UNIVERSITY OF CALIFORNIA
SANTA CRUZ

CASUAL CREATORS: DEFINING A GENRE OF AUTOTELIC CREATIVITY SUPPORT SYSTEMS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Katherine Compton

June 2019

The Dissertation of Katherine Compton
is approved:

Prof. Michael Mateas, Chair

Prof. Jim Whitehead

Prof. Katherine Isbister

Quentin Williams
Interim Vice Provost and Dean of Graduate Studies

Copyright © by

Katherine Compton

2019

Table of Contents

List of Figures viii

Abstract xxxii

Dedication xxxiv

Acknowledgments xxxv

1 Introduction 1
- 1.1 Mission of this dissertation . . . 5
  - 1.1.1 Definition of a casual creator . . . 6
  - 1.1.2 Examples of casual creators . . . 7
- 1.2 Research Questions: . . . 13
- 1.3 Contributions: . . . 13
  - 1.3.1 Structure of the dissertation . . . 15

2 Motivation 18
- 2.1 Labelling an old genre . . . 20
- 2.2 Of genres and usefulness . . . 22
- 2.3 Effects of having a community . . . 23
- 2.4 Casual creators as an intervention in Computational Creativity and Artificial Intelligence . . . 24
  - 2.4.1 Showcasing Computational Creativity . . . 25
  - 2.4.2 Turning an intimidating algorithm into a trusted collaborator 27
- 2.5 Related fields . . . 29

3 Central Themes 35
- 3.1 Power vs. Control . . . 35
- 3.2 Maps and Spaces . . . 41

4 Creativity 45
- 4.1 Creativity inside and outside of production . . . 47

4.2 Environmental creativity . . . 52
4.2.1 A digression on the "Wundt curve" . . . 55
4.3 Embodied creativity . . . 55
4.4 Social values, economics, and creativity . . . 61
4.5 Situating casual creators (self and society) . . . 66

5 Creativity Support Frameworks 69
5.1 Creating creativity . . . 69
5.2 Flow . . . 70
5.3 Improv, randomness and creativity cards . . . 73
5.4 "Creativity Support Tools" . . . 80

6 Existing Frameworks of Generativity 86
6.1 The case for frameworks . . . 86
6.2 Generativity across domains . . . 90
6.3 Terminology for "generativity" across fields . . . 91
6.3.1 Generative art frameworks . . . 93
6.3.2 Generation-in-games frameworks . . . 101
6.3.3 Frameworks of natural history . . . 104

7 A generative framework of generativity 107
7.1 Generative methods (components of a generative pipeline) . . . 110
7.1.1 Content selection . . . 111
7.1.2 Tiles . . . 112
7.1.3 Grammars . . . 113
7.1.4 Parametric . . . 116
7.1.5 Geometric extrapolations . . . 118
7.1.6 Distribution . . . 122
7.1.7 "Agent" simulation . . . 125
7.1.8 Machine-learned statistical models . . . 129
7.2 Consumers of generative pipelines . . . 132
7.2.1 Navigation Methods . . . 133
7.2.2 Search-based techniques . . . 133
7.2.3 Constraint solvers . . . 135
7.3 Example generative systems . . . 137
7.3.1 StippleGen . . . 137
7.3.2 No Man's Sky and Spore . . . 138
7.4 Generominos: turning a framework into a physical toy . . . 141

8 Frameworks of Interactivity 147
8.1 Direct manipulation . . . 148
8.1.1 Going Deeper in Direct Manipulation . . . 151
8.1.2 The Gulf of Execution and Evaluation . . . 152

8.2 Reflection-in-Action . . . 160
8.2.1 Models of learning: Single and double loop learning . . . 162
8.3 The site of interaction . . . 163
8.4 The Grokloop . . . 165

9 Possibility Spaces **170**
9.1 What are possibility spaces? . . . 171
9.2 Translating from model to artifact . . . 173
9.3 Describing a possibility space . . . 176

10 Narrow and Shallow Possibility Spaces **197**
10.0.1 Simple spaces . . . 197
10.1 Four simple-space creators . . . 199
10.2 Super Sequin Simulator - DIY Flip . . . 202
10.3 Strategies for simple spaces . . . 204
10.3.1 Progressively-expanded tools . . . 205
10.3.2 Intentionally slow grokloops . . . 208

11 Parametric Spaces **211**
11.1 "Pure" and "lumpy" generators . . . 211
11.1.1 Displaying and mapping parametric possibility spaces . . . 217
11.2 Moving through possibility space . . . 219
11.2.1 Animation . . . 221
11.2.2 Optimization . . . 222
11.3 Hill-climbing as a casual creator . . . 227
11.3.1 Ownership in genetic algorithms . . . 229
11.3.2 Parametric patterns in Machine Learning . . . 232

12 Drawing Tools **235**
12.1 Drawing on different perspectives . . . 235
12.1.1 Mark-making . . . 239
12.2 Modes of drawing in casual creators . . . 248
12.2.1 Conformance . . . 249
12.2.2 Operational . . . 249
12.2.3 Accumulative . . . 251
12.2.4 Transformational . . . 256

13 Slow Creators **262**
13.1 Music . . . 263
13.2 Generative literature . . . 271
13.3 Slow creators: Game-makers . . . 279

14 Casual Programming **289**
14.1 A Few Existing Casual creator languages and IDEs 290
14.1.1 Logo 290
14.1.2 Scratch 296
14.1.3 Casual languages and the Gulfs of Execution and Evaluation 300
14.1.4 Features of a casual creator language 302

15 Tracery and Chancery **309**
15.1 Tracery Timeline 312
15.2 Tracery language features 316
15.2.1 Domain model 317
15.2.2 Tracery 2 Syntax 323
15.3 Tracery specification levels 324
15.4 Tracery as a little language 325
15.4.1 Scaffolded 326
15.4.2 Unfoldable 328
15.4.3 Little 330
15.5 Community ecosystem 331
15.5.1 Tracery Ports 331
15.5.2 Languages Inspired by Tracery: 332
15.6 Chancery 336

16 Pattern - Entertaining Evaluations **342**
16.1 Instant partial feedback 343
16.2 Fictional feedback 347
16.3 Human social feedback 358

17 Pattern - No Blank Slate **362**
17.1 Examples of blank-slate solutions 364
17.1.1 Starter artifacts 364
17.1.2 Inspiration 366
17.1.3 Structure and Templates 369
17.2 Case Study: Spore 371

18 Pattern - Safety **375**

19 Pattern - Possibility Space Annotation **385**
19.1 Automatically mapping generators 388
19.2 Social cataloging 391
19.3 In-game mapping 399

20 Pattern - Self-promotion **404**
20.1 Social organization . . . 407
20.2 Famous explorers: from "merely-mapping" to "creativity" . . . 409
20.3 Contests . . . 413
20.4 Alternative channels . . . 415
20.5 Fame and Money . . . 421

21 Pattern - Repetition, Repetition, Repetition **429**
21.1 Precision, symmetry and interaction . . . 432
21.1.1 Gestural symmetry . . . 432
21.1.2 Manipulating user-provided photos . . . 436
21.2 Embedding symmetry in external systems . . . 438

22 Pattern - Transformation **443**
22.1 Coloring-in a drawing . . . 448
22.2 Traveling to a new medium . . . 458

23 Pattern - Make It Real **463**
23.1 Driving interest in personal fabrication . . . 463
23.2 Casual creators for "personalization" . . . 467
23.3 Social affordances of "real" objects . . . 468
23.4 Artifacts as memory . . . 471

24 Pattern - Signifiers Add Power **473**
24.1 Semiotic Theming . . . 474
24.2 From K'nex to Lego to Playmobil . . . 477
24.3 Juxtaposition and the Imagination Playground . . . 483
24.4 Pure refined semiotics: Emoji . . . 488

25 Pattern - Expressing Identity **493**

26 Expert evaluation **503**

27 Conclusion **521**
27.1 Future work . . . 526

28 System Index **528**

Bibliography **616**

List of Figures

1.1 (Left) Isaac Karth and I creating paintings in our ScholarsPlay livestream of *Eastshade*. (Bottom) Phillipa Warr's printed photos of *Firewatch*. (Right) GTA photos by users *SnowGoose95* and *Alex Crowley* 12

1.2 A few casual creators and their artifacts, to show the range of the software (and non-digital) systems that we will see in this dissertation 17

2.1 An assortment of assisted-creativity toys. A vintage paint-by-numbers kit[s159], a knitting nancy, a modern day *Rainbow Loom*[s184], the Mattel Knitmagic[s116] which was one of my favorite childhood toys (from Flickr user *x-ray_delta_one*, from Flickr user *sallysetsforth*, image from Wikipedia, `https://theknittingneedleandthedamagedone.blogspot.com/2013/01/knit-magic-isnt.html`) 21

2.2 Janelle Shane's interrogations of image-labeling algorithms, using out-of-context sheep to explore the edges of the algorithm's abilities 28

2.3 "Casual creators" in relation to other fields of study 29

2.4 Samples explorables from the *Explorable Explanations* site (left): *The Evolution of Trust*[s70], *To Build a Better Ballot*[s24], *Pink Trombone*[s172], and *Earth Primer*[s59] 33

2.5 A preview for an abstract music-making game Sentris identifies a common creative problem among casual creative users, and advertises a "fix" to attract them. If this kind of software could identify itself as a "casual creator", would give us a shorthand for this headline, while also linking it to others of its kind? (https://www.polygon.com/2014/8/4/5968415/if-youve-ever-been-too-scared-to-make-music-try-sentris) 34

3.1 "Creation" apps on the app store. 39

4.1 A collage of creativity posters from Pinterest 47
4.2 Maybe we need a novel and useful definition of creativity. 51
4.3 Accominotti's chart of the painters, their active years, their peaks (by artwork price) and their movement, demonstrating the remarkably tight clustering of individual behavior in each "cohort" . . . 53
4.4 Wundt's curve, decay of nerve response in response to stimulus repetition over time[441]. 55
4.5 Warren's curve, linking intensity of stimulus to pleasure, showing a sweet spot of pleasure between apathy and pain[431]. 56
4.6 Berlyne's curve:applying Wundt's curve as a metaphor to describe response to aesthetic stimulus. 57
4.7 Saunders's curve:compressing Berlyne's axis to "familiarity", not complexity and familiarity[344]. 58
4.8 Environments which students considered most conducive (left) and least (right) conducive to creativity. It is unclear if they controlled for the presence of seating[261]. 58
4.9 A Ladies Home Journal page advertising knitting tutorials, watercolors, crafting tools, and instructions for the hot new trend of "crazy quilts". With updated graphics, this would be a familiar front page of Pinterest. From https://babel.hathitrust.org/cgi/pt?id=mdp.39015012341544;view=1up;seq=20 63

5.1 The "channel of flow" from [110] 72
5.2 Partial image of a voting page for Bar Karma, archived from `https://web.archive.org/web/20110312010512/http://current.com:80/studios/create-episodes/102/develop-the-story-of-the-autho/newest/0/voting/` 75
5.3 Images from the "Game Design: A Book of Lenses" Android app. It has a Tarot-like "divination" mode where four cards are randomly drawn and juxtaposed together. 79
5.4 @sambreed's @ObliqueBot posts cards from the Eno/Schmidt deck, @norareed's @botblique and my own @artideabot create generative riffs on the Eno cards, doubling down on the divergent properties of the original deck 80

6.1 The Grokloop, which will become much clearer after the next two chapters. 89
6.2 The table of notable frameworks of generative art from Dorin et al. Don't worry about reading them individually, just note how *many* there are. 93

7.1 Tarot spreads (from Flickr user Atell Rohlandt). The tarot reader selects a spread type (content selection), then fills the spaces with cards (tiles), which are randomly selected from a deck (content selection). 113
7.2 (bottom)In *Ring Designer*[s190], the user selects a layout and band style (content selection). For each gem type, they select the size, style, and color of the gem (content selection), but these choices are used to fill in sockets in the ring's band in a fixed pattern (tiles, grammar). *Barbie Magic Fashion*[s10] uses a simplified version of the same mechanic decorate tiaras. (top) Constrained tile-placement in *Diablo II* and *Betrayal at the House on the Hill*`https://diablo.gamepedia.com/Map_Generation`, Flick user *meoplesmagazine* . . 114

7.3 Two revisions of an array-of-floats tutorial and platform I provided for users to play with or modify. This is part of a long-running[72] set of tutorials and code I have developed to help developers better understand parametric methods, which is one of the most approachable, powerful, and easy-to-get-wrong generative methods categories. .9513.6 119
7.4 My projects, *Idle Hands*[s97] and *Stjärna*[s220], which both use Voronoi diagrams. *Stjärna* uses them to create generative paper lanterns *and* a small Swedish town to install them in. 121
7.5 Brushes I made in *Adobe Illustrator*, and their deformation/application along a curved path 122
7.6 *I/O Brush* (left) and Google's *Tiltbrush* (right). Both tools use extrapolation along a curve, with different types of generative/emergent pattern. In IOBrush's case, the pattern is sampled with a webcam (content selection). For *TiltBrush*, the patterns are textures or geometry provided by the system, but are animated or decorated with particles and glow effects 123
7.7 Decals in *DIY Fashion Star* can be selected from the user's photos or pre-created assets(content selection), then placed, scaled, and rotated on the clothing 124
7.8 *The Powder Game*, with assorted user-made levels that simulate volcanoes, engines, and electrical systems (`https://dan-ball.jp/en/javagame/dust/`), and *The Sandbox:Craft, Play, Share* commercialized version of the same gameplay on Android 126
7.9 I often use particle physics with spring-forces in my prototypes, rather than inverse kinematics, for softer and more expressive generative animation. *Monster Mirror* (left) pulls particle-based costumes around according to motion-tracked skeleton data. *Teuhana* (right) tugs at the limbs of generative particle-based dolls around in time to music to create virtual dances. 128

7.10 *Loopy*, *The Wisdom and/or Madness of Crowds*, *Neurotic Neurons* and *Machinations*, all of which use graph-based simulations, expressing wildly different phenomena. 130
7.11 Comparing the pipelines of the *Spore* and *No Man's Sky* terrain generators. 140
7.12 The basic shape of a Generominos card and pipeline. 142
7.13 Positive student feedback on the Generominos, from Eddie Melcer's evaluation. 145
7.14 The Generominos went out into the world and immediately were put to use in classrooms and design studios. 145
7.15 Instruction pamphlet for the Generominos. 146

8.1 Loops from Argyris[15] and Hutchins[204], showing the same loops of iterative learning. 167
8.2 The "grokloop", a process of exploring a generative space through interactions with an artifact 168

9.1 A spirograph drawing resulting from the previous instructions, drawn with the Inspirograph online simulator[142][s101] 174
9.2 Some of the Spore terrain scripts I wrote. Each script was a set of "brushes" written in the Maxis in-house particle system language Swarm. These would rain particles down on the planet, move them according to some set of choreographed forces, and leave a different kind of stamp (riverbed, lake, alpine mountain, round dome, e.g.) multiplied by different height values. Because of the stochastic nature of these brushes, some bad behavior was common, so I preran dozens of planets for each script to save a catalog of good seeds that could be used in-game. 177
9.3 "Bowls of Oatmeal" as a headline. 178
9.4 Models, artifacts, and properties for three examples of typical game systems and three real or hypothetical generative design examples, like generative chairs[126] and antennas.[196] 179

9.5 Expressive range(right)[383] and Danesh[105][103] both map properties of generated artifacts onto a 2 dimensional space. Danesh elevates the expressive range into a casual creator for generators by adding slider-tuning, autotuning and browsing content with rollovers. 180
9.6 Movement through states in a musical work (in a field of potential next states), and the *horizon* of memory and prediction from a single state[11]. 181
9.7 The loop of navigation from Darken's later work[116](right), adapted from Jul and Furnas[212]. The navigator continuously acts on hypotheses and updates their mental map with the results. Compare this to the grokloop and you will find a very similar structure. These loops-of-learning haunt every chapter in this dissertation. . 183
9.8 *Dryad*'s possibility space contains many aesthetically displeasing trees, but by identifying user-preferred landmarks (top left) they can identify "good" spaces (bottom left). *Petalz* divides its space in to computationally-distinct regions (top right), which are sometimes aesthetically related, to make a continuous/toroidal 2D-remapped possibility space of 80 collectible "species" (bottom right). 185
9.9 Not every local space is continuous, not every continuous space is local. Navigation qualities can differ over short and long scales, and can be different for different parameters of the model. 187
9.10 Chaim Gingold's presentation at GDC 2007 illustrated how a designer of a generative system carves up the possibility space with their generator into which artifacts are Possible, Probable and Desirable. Some bad artifacts may be *possible* to generate, some *desirable* artifacts may be *impossible* to generate, but the designer should build a system such that it generates some *probable* varied selection of mostly-good artifacts.[158] 188

9.11 Different kinds of distributions are acceptable in different use cases. The far left example, with no broken artifacts but no exceptional ones, would be good for an on-line generator with no filtering, where hard failures are unacceptable (like an impossible platformer level) but it's ok for the artifacts themselves to be unmemorable (*Diablo II*[s48] map generation falls into this space). A space like the one on the far right, with a rich exceptional space among many bad and disastrous artifacts can be an exciting space for users to search for, and "own", the gems among the dross. Likewise, the middle two columns are common distributions, with a few broken outcomes and a few exceptional artifacts. These would require some filter, either by the user or the system, with a reward of the occasional rare find. 192

9.12 In a movie example of a procedurally generated scene, the directors cast twins and triplets in the background of a scene in The Matrix to make an argument about not noticing small differences when presented with a much larger difference. One character is *characterful*, the others are *differentiable* but not unique. The character who created the scene mentions "the woman in the red dress" after Neo exits, and asks him if he saw her (the characterful artifact is *named* and *externally remembered*).[92] 195

10.1 *Super Sequin Simulator - DIY Flip*, *Pixel.ly 3D*, *Pixel Art: Color by Number*, and *DiamondArt* are all shallow or narrow creators. . 204

10.2 *Cake Pop Maker - Cooking Games* and *Real Cake Maker 3D - Bake, Design & Decorate* are apps marketed to children to get them to watch ads while waiting to use a casual creator. On the left, *some* of the two dozen steps required between creative-control stages of *Real Cake Maker 3D*. 209

10.3 It's impossible to say how many color-by-number apps there are on the app store. This is as many screenshots as I cared to take, but the list keeps scrolling for as long as I refreshed. 210

11.1 *Ring Designer*'s model of a ring. Even in this simple choice picker, some of the choices are interdependent, making the possibility space more complex than a simple combinatorial problem. 213

11.2 A parametric space modeled as a 3-dimensional cube (representing an N-dimensional cube). The flower generator shown is a mapping between an N-dimensional vector (representing flower properties of varying directness), which is a point in the cube, and an output artifact, a flower. Because these vectors are spatial representations of the inputs, spatial operations in N dimensions work the same way they do in 3 dimensions. We can lerp from one artifact to another, or sample points and create an N-dimensional bezier curve and sample along it. 215

11.3 In a continuous space, models *very* close to each other will look more similar, but over larger scales, they will diverge rapidly. In a system with locality but non convergence, models close to each other will produce similar artifacts, but may have small discontinuities between them. The space is only ever *sampled* though, so technically non-continuous spaces (like fixed-precision numbers) will be perceived as continuous as long as they are local on that scale. 216

11.4 Allison Parrish's word visualization (top-right) and *Robot Mind Meld* (left) use vector-encodings of English words. Compare these to decidedly non-casual word-encoding explorers like the official Tensorflow projection viewer (`https://projector.tensorflow.org/`). 217

11.5 Dryad's multidimensional "landmark" trees. 219

11.6 *Slidertoy*[s211] is a platform I built as a way for users to explore parametric spaces with a variety of interactive techniques. They can slide sliders to find a particular point/artifact in the space. They can set it to perform an animated randomwalk through the space, which morphs through the continuous spaces of artifacts. Or they can hook the inputs of the parametric system up to several different input pipelines that create arrays of numbers, like music frequency or the Leap Motion.(`http://www.galaxykate.com/apps/aof/`, `https://github.com/galaxykate/slidertoy`)). . . . 223

11.7 Gameplay in *Funky Ikebana*. Seven flowers are animating attractively (offsetting the L-system angle by a sine-wave of the time). The user clicks at the base of their favorite, which creates seven "offspring" with slightly-varied "DNA". They can do this very quickly, because the process of visual evaluation, clicking and refreshing is nearly instantaneous. This makes for a fast grokloop and an effortless journey through a wide range of floral possibility space. . . . 224

11.8 Genotypes transform into phenotypes, which transform into evaluations, which pair with the original genotypes to become a new set of genotypes. Each one of these transformations can be a generative pipeline(shown here as a pipeline of *Generominos*). 226

11.9 (left) The muscle editor and simulator in *Evolution by Keiwan(, and the simulator in Walkers, which has no editor. (right) The simulator, parametric editor, and high-scoring user-discovered artifacts from BoxCar2D.* 228

11.10One of the images with the most descendants on PicBreeder (`http://picbreeder.org/search/showgenome.php?sid=4041`). 230

11.11Family tree of a sheep, with its creator, its parents, its descendents, and users who rendered one of its frames/ (`http://electricsheep.com/archives/generation-244/dead.cgi?id=23652`). 231

11.12Scenes from *Mendel*: naming flowers, collecting them, and seeing them in relation to named specimens. 232

11.13A joint project with Microsoft, MIT, and NY's Metropolitan Museum of art, *Gen Studio*[s84][180] maps out artifacts from the Met's collection as a wonderfully Lynchian map of landmarks. The user drags a point spatially, which creates a vector point somewhere between the sample *real* artifacts, which drives a GAN to make a synthetic artifact that reads as that percentage of each artifact. . 233
11.14Interaction-wise, *GanBreeder*[s80] is almost an exact clone of *PicBreeder*. Users can breed or mutate GAN-generated images, and crowd-voting surfaces the current most exciting "landmarks" to evolve.[373].234

12.1 Interactive mark-making installations are popular in museums. These provide a wonderful space of casual creator examples, as they have to balance *some* creative expression on the part of the participant with automation or structure that creates a cohesive whole for the museum space. Yayoi Kusama's "Obliteration Room"[s151][230], Joseph L Griffith's "Drawing Machine 1"[s53][170], Karina Smigla-Bobinski's "ADA"[s3][171], Ryan Feddersen's "Coyote Now"[s37] coloring book mural at the Santa Cruz MAH[134]. 238
12.2 Diffusion brushes in *Kid Pix*. 244
12.3 Conformance drawing in *HelloChinese*, *Okami*, *Calligraphy Handbook*, and *Learn to Draw Glow Cartoons*. 250
12.4 Lava flows, erosion, and footprints are a frozen record of forces and dynamic movement. Even if we can't "read" the geology, their shapes implicitly tell us about the directions and collisions that happened here (CC-licensed photos from Pxhere, Gina Dittmer, and Tari Noelani Mattox). 252
12.5 Action Painting Pro and Joy Exhibition. 253
12.6 *Pax de Deux*, Matt Bell's Kinect experiments, and *Happy Action Theater*[178][28][265]. 254

12.7 *Paintthrush* is a mixed particle-system/accumulative drawing tool. This image shows a number of prototypes that I went through, including some where there were no birds, but the acceleration vectors were drawn *accumulatively* onto the canvas, creating an appealingly shaggy record of hand motion. 255
12.8 Brush types (from [239]. 258
12.9 The *Pendant Maker* UI. It became quickly apparent that users didn't even look at the drawing buffer, they could draw while *only* looking at the extruding cloned 3D paths. 259
12.10An earlier prototype of *Scribbles*[s197] (`http://www.galaxykate.com/apps/Prototypes/Scribbles/`), an earlier spring-particle drawing tool using accumulative drawing to show its motion over time (left), and several drawings from *Jazz Composition*(right). 260

13.1 *Music Construction Set* (top-left)[421], *Google Song Maker* (bottom-right), the original *Mario Paint Composer*(top-right)[s131][417], DanielX's *Mario Paint Music Composer*) with templated sections. 266
13.2 *WolframTones* and its many indirect controls 267
13.3 (left)*BubbleHarp* and (right) *Ethereal Dialpad*, two semi-autonomous gesture-based music apps. 268
13.4 (top-left) Some of *Panoramical*'s many worlds. (right) *Patatap's* geometric shape animations for a single setting (it has multiple color and sound profiles to toggle through). (bottom) Four of *Bloom*'s ten worlds, which are the least complex and least interesting of these visualizations, perhaps to not outcompete Eno's music. . . . 269
13.5 Google's *Otto Fischinger doodle.* 270
13.6 (right)Max Kreminski's blackout-generator bookmarklet and poems (`twitter.com/maxkreminski/status/941466797420638208`), (bottom) Emma Winston's blackout poetry maker [437], (top) Erasures. 272

13.7 *Auctor*, an Android app for generating and tracking collections of characters. 276
13.8 "Coen brothers movie"-making social game *Fiasco*[s73] and *Lasers & Feelings*[s118], a one-page story game about Star Trek fan fiction. . . 277
13.9 *Tanagra* and an image from my master's thesis on generating platformer levels. *Tanagra* is the smarter system of the two, but it has fewer ways to prove that visually to the user. (sadly, all the color images of my original system have gone missing, but the trajectories were colored to show successful and unsuccessful jumps). 288

14.1 *Code with Anna and Elsa*, Google's Frozen-themed *Scratch* programming tutorial inspired by Logo. 294

15.1 Railroad diagrams for Tracery 1 (CBDQ edition) and Tracery 2 . 338
15.2 (left)A variety of visualizations of Tracery grammars with syntax coloring, graphs, colocation diagrams (by Ben Kybartas), frequencies of leaf-node selection, and automated Tracery error-checking. None of these have been as successful enough to add to the editors, except hierarchical decomposition of output (top right two) and rules (bottom right two), which is especially important as Tracery 2 syntax gained complexity. 339
15.3 *HipChef* and *Revisions*, two of my own games with embedded Tracery as a core feature of the experience. 340
15.4 Users playfully creating scenes with bots. A few "honeypot" bots trick ill-tempered users into repetitive conversations with them, but most bot conversations are knowingly using the bot as the straight-man partner in an improv conversation. 341
15.5 The original open-sourced Google code as "Bottery" (left), and my recent experiments at making a multi-bot casual-creator platform for writing bots in Chancery (formerly "Bottery"). (right) 341

16.1 (top) Google's Perspective API and Text.IO sell companies a promise of "scientific" confidence in writing.(bottom). Three gender-bias tools return diverse values for the same sample text: the TotalJobs Gender Bias Decoder (`https://www.totaljobs.com/insidejob/gender-bias-decoder`), Kat Matfield's Gender Decoder (`http://gender-decoder.katmatfield.com/`), and Tom Forth's Gender Bias (`https://www.tomforth.co.uk/genderbias/`). 344

16.2 A few of the "beauty scanners" on the Android store. App notes include: "WARNING: Please do not use this app if you have low self-esteem or confidence problems." and "gives the results of the scan randomly, so do not expect it Scanner realistic assessment of your appearance." These *may* be examples of a "dark pattern" for "entertaining" evaluations. (`play.google.com/store/apps/details?id=com.prettyscale.PrettyScale`), (`play.google.com/store/apps/details?id=com.schnapps.face.scanner.beauty`). 347

16.3 *BECOME A GREAT ARTIST IN JUST 10 SECONDS* and *Art Sqool* 352

16.4 *Human-Free Photobooth* tested in the EIS lab and at a party. While the AI wanted to describe the local furniture it could be certain about, it was also willing to guess at the identity of the human objects. Humans especially like lying to the AI, by yelling "no!" when it asked if there were any humans there. 354

16.5 Bots in Techne. In the top row, the metagrammar generates a bot with some subset of tools (grammar expansion rules) for a given domain (i.e., only red, muted colors, spiky shapes, and spiral layouts). In the middle row, bots compare their art to the art of their creative community. If they find one they like better, they will frenetically try out their idol's tools, hoping to "improve" their art. In the bottom row, many bots were released onto a virtual platform to try to "talk" to other bots of different domains (mostly, the mixed-media bots stole lines from the poets to use in their posters). 359

17.1 Left: The *Doodle Pad*, a mid-century book of "sexy lady" templates to fill in, reinterpreted as action heroes by David Jablow [140], Right: The *Heartless Machine Guide to Drawing* (`https://www.kickstarter.com/projects/494572698/the-heartless-machine-guide-to-drawing-fill-in-the`). Note the caption: "Starting with a blank page is terrifying! This book alleviates that fear by starting you with a partial drawing. You just fill it in." 365
17.2 Pictures for quilting inspiration and a finished quilt:Quilt inspiration and finished quilt by Anne Sullivan. 367
17.3 Create-a-Sim challenges:Youtubers and Twitch streamers seem particularly fond of Create-a-Sim challenges, performing difficult creative tasks in front of an audience. 368
17.4 6 word short stories:Microfiction stories from IO9's Six Word Scifi story challenge[275]. 368
17.5 Art memes of Twitter. With #Dinovember, #Junicorn, and #Mermay, there is a drawing challenge month for every taste. 369
17.6 (left) Multiple drawings of the same character in different styles:Yoshi Yoshitani's original drawing as reinterpreted by dozens of Twitter artists for the #drawinmystyle challenge, where one user posts a character, and everyone reinterprets it. (center)Artists draw their surroundings in #myartistdesk. (right) Not *every* challenge meme is for visual art: the scifi site IO9 hosted a "six-word story" challenge for its readers[275]. 370
17.7 Templated creativity activities only require the user to make one or two choices (instead of generate whole new ideas). "#ChangeA..." memes let users figure out which word in a movie or book title to change for humorous effect. The "emoji sheriff" meme creates sheriffs a range of colorful expressive purposes, while only changing the emoji used. 371
17.8 Christmas buildings:a holiday contest for scenes made in the building editor[404]. 373

17.9 Challenges and creations from *@instrumentbot* and *The Best Art*, as fulfilled by their human partners. (`https://the-best-art.computer/`). 374

18.1 The casual creator embedded in the word-search game *Alphabear* lets users fill in templates with the words they found, and encourages them to post them to Twitter. Also shown: *AbyssRium* dress-up, which also has Twitter snapshot sharing, *Petalz* collectible pots[321], *Pixel.ly 3D* menu replacement 378

18.2 Creative risk-diffusion in *Tee K. O.* and *Coffee: a Misunderstanding*'s puppets/puppeteers/director/audience setup (not shown: awkward laughter). 381

18.3 Gingold's illustration of the *Spore* vehicle editor's space during development (left), how they tried to shift it (center), and comparing it to the space of a tool like *Maya*[s132] (right). 382

19.1 The 80 machine-learned categories of Petalz flowers (from [323]). . 389

19.2 On the top row, artifacts randomly selected from the whole tree space. On the bottom, artifacts randomly selected from *high density* tree space, which are more reflective of real-world trees (from [401]). 391

19.3 Keyforge deck composition. 394

19.4 394

19.5 A variety of *Galactic Arms Race* bullets (from [185]). 396

19.6 The Elite Dangerous Database catalogs planets and content, but also metagaming patterns that emerge from them, like arbitrage loops (from (`https://eddb.io/`)). 398

19.7 A medieval villager's thoughts in a galaxy somewhere in Orteil's *Nested* (`http://orteil.dashnet.org/nested`), and the villager specification, as documented in the Wikia for Nested[292]. 399

19.8 Images from the now defunct Athena DataBase of named *No Man's Sky* entities. 401

19.9 Smallest and largest creatures found by users. 401
19.10Collecting and cataloging plants in Mendel. You can see the family tree of any plants you have bred. 402

20.1 Mailboxes on the Sporepedia. 409
20.2 (left)A *Legends Viewer* view of one character's progression through their *Dwarf Fortress* lifespace. (right)Output from Jacob Garbe's narrative search-and-retelling algorithm. (both photos used with permission of Jacob Garbe) 412
20.3 Example "community innovation" contests (from [45]) 413
20.4 Official *Spore* contests with a range of rules, prize styles, and motivations. 415
20.5 Unofficial Spore contests, with trophies invented and awarded by users. 416
20.6 A Christmas contest for *Oculus Medium*, keeping the *Spore* traditions alive. 417
20.7 A creature save file. Go on, drag it into the Spore editor. It should load *from an image embedded in a dissertation.* 418
20.8 Solutions posted to Twitter. 420
20.9 A creatively-terrible *Opus Magnum* solution(from user @britishgaming (`twitter.com/britishgaming/status/922022684116045825`)). 421
20.10An *Opus Magnum* leaderboard (from [420]). 422
20.11Books and a magazine cover from *Second Life*'s zenith as a money-making platform. 423
20.12A sampling of Twitch livestreams active at a single moment on Jan 11th, 2019. We can see a gameplay challenge (have 100 babies), a fund-raising challenge, a viewer service challenge (fan portraits), and creativity challenges (making Peter Pan, a portrait, a castle). 426

21.1 Mechanically-assisted repetition in Italian millefiore and Japanese yosegi woodwoorking; both are done through bundling long strips then thinly slicing the bundle to create identical tiles. (from Wikimedia user *Captain76* and Ulrich Mayring) 430
21.2 a page from computersandautomation, aug 1965:The winning artist's statement from A. Michael Noll: "The positions of the vertical and horizontal bars have been chosen at random with the constraint that the positions mus fall inside a circle. The length and width of the bar was chosen at random within a specified range. If the position of the bar fell within a parabolic region in the upper half of the circle, the length of the bar was shortened by a factor proportional to the distance of the position from the edge of the parabolic region. The motivation for this type of pattern came from Piet Mondrian's "Composition with Lines", 1917, now in the possession of the Rijkmuseum Kröller-Müller, Otterio, The Netherlands." . . 431
21.3 Making a hexagonally-tessellating shape with the online *Interactive:Tessellate!* editor[s202]. 433
21.4 (left) Example of *Spiro* drawings. (right) Symmetries in the *Silk* app (none, mirrored, spiral, and spiral-mirrored). 434
21.5 *Tilemaker*(right) and *Engare*(left) are two systems which use constrained user gesture (or physically-mediated gestures) to create patterns from repetition and symmetry. 436
21.6 Symmetrical photobooths are common on the app store. They combine immediate feedback with the actractions of symmetry, and the interest of seeing one's own face in a new way (Android app store shown). 437
21.7 *Mirror Lab*, the most complex of the photo-symmetry apps, has a number of different symmetry and mathematical distortion filters. We can see a few ways of looking at one of my cats. The results are *more interesting* than they would be if the manipulated image were not of a cat. 438

21.8 439
21.9 :A variety of mandala (rotational symmetry) drawing apps 441
21.10Sonia Harris's sardonic linework, User "blagh"'s *Monument Valley 2* mandalas, with appropriate commentary[416]. 442

22.1 A variety of coloring books for sale on Amazon.com, with heavy use of psychologically or culturally meaningful signifiers (see chapter 24).444
22.2 A paint-by-number of President JFK, uploaded by Larry Miller (Flickr user larry&flo). 447
22.3 *Zentangle* patterns shared by users on Pinterest. 448
22.4 A *Zentangle* made with *Kid Pix*'s fill tool, set to randomly select a fill pattern. Made in a few minutes with the Internet Archive's DOSBox version of *Kid Pix* (1991) (`https://archive.org/details/KID_PIX_DOS`). 449
22.5 Exploring the use of *Photoshop 6*'s[s168] content-aware fill on a coloring book page. While it makes transformational magic across large single areas (first and second column), it can't reason about regional fills (third column) and the image becomes noise. (public domain image from (`https://picryl.com/media/mandala-coloring-page-coloring-for-adults-backgrounds-textures-89ba6f`)) 450
22.6 An example of the *Style2Paints* algorithm, almost exclusively used to recolor line drawings of anime-style girls [250]. 451
22.7 My attempt an a *Style2Paints* artwork, using a turn-of-the-century woodcut of Urania and some rough scribblings to indicate color regions. With little guidance, it admirably keeps hair, costume, and detail separate from skin tone. 452
22.8 *Style2Paints* is one of the only coloring apps that I've seen that explicitly recognizes flat colors, gradients, and region shading as steps in the process, giving users control and access to each one[396].453

22.9 In addition to multiple "AI artists" styles, *Style2Paints* can render in "Careful" or "Careless" mode, providing a high-level stylistic control[396]. 453
22.10*Scribbler* allows iterative modification of the base sketch. 454
22.11*DeepDream Generator*[s46], another *DeepDream* casual creator, takes a picture of my cat and a style to create new art. They don't say what images they've used to train their style networks. [1] 456
22.12In the top row, the user annotates regions of a style image, then draws a new source image with those same colors to create regions. Below, the network is recursively iterating to match the region style of the style image[56]. 457
22.13Examples of user creativity and playfulness with *Edges2Cats*[101]. 458
22.14Crayola also has a coloring-to-3D app, but its very low app store reviews (and angry posts) suggest that this technique, while promising, can lose its magic quickly if the camera recognition isn't easy or reliable. 462

23.1 Several interactive creators for 3D printing by *Nervous System*. *Cell Cycle* creates a double-looped ring or bracelet, *Kinematics* creates a hinged Voronoi necklace. *Kinematics Cloth* creates a 3D printable dress. *Radiolaria* create a flat celled object which can be used as earrings or a trive. *Kinematics-at-home* creates a jointed bracelet that can be printed on an inexpensive home 3D printed (unlike most of their designs which must be ordered from a professional printer). 466

23.2 (top) *Broderie* scarves being traded at XOXO for a variety of personally-created goods. The physicality and limited-edition nature of these scarves seemed to make them more interesting than other experiments in physicalizing *Broderie* patterns, like making a digital art projection or printing them on tshirts for sale (`https://society6.com/galaxykate`). Users also really enjoyed that I was trading the scarves for other hand-made objects, linking them into the community of makers at XOXO. 469

24.1 "Downton Abbey" and "Outlander" coloring books, two media properties that share a large marketing-demographic overlap with coloring books. 475
24.2 Add rhinestones to Rapunzel's hairbrush, excavate grubs from a tree stump, design your own mermaid tail[443]. 475
24.3 So many Indian wedding apps. More that western-style wedding apps. I'm not sure why these are *this* popular, and nothing online has been written about them (in English, that I've found, at least). 477
24.4 Recontextualized Lara Croft needlepoint by Becky Schaefer (from [206]). 478
24.5 A *Spore* creature becomes more creature-like with the addition of eyes, mouth and limbs. ([386]) 479
24.6 Anything with eyes on it is a character.(photo used with permission of Marie Huynh). 480
24.7 Semiotically-heavy Lego sets from Lego.com (`https://www.lego.com`). These sets carry semiotic meaning from books, movies, videogames, and cultural knowledge. 481
24.8 The submarine from the movie, reconstructed by fans (`www.thebrickfan.com/the-lego-movie-master-builders-submarine/`). 482
24.9 The *Lego*, *K'Nex*, and *Playmobil* product pages 484
24.10A full output of (`https://galaxykate.itch.io/eternal-night-vale`). 485

24.11A selection of possible first lines of the Nightvale generator. Given a large number, it is easier to see the structures and the semiotic components. 486
24.12People occasionally try to solve bafflingbot's murders. 487
24.13magicRealismBot, thinkpiecebot, twoheadlines, and the customer generator from *HipChef*, my unreleased hipster cafe sim. 488
24.14Semiotically-heavy terms, but only for game studies programs. Placed in whiteboard-style juxtaposition structures to create an imagination playground of research topics, for the game *Revisions* (`http://www.galaxykate.com/revisions/`). 489
24.15Emoji bots everysheriff, tinyprotests, emojifart, tiny_gardens, choochoobot, bot_ideas, and thetinygallery. 492

25.1 IMVU, a long-running avatar-based chat app and virtual goods platform. Users can buy clothing and hair, or create and sell them (via pro-creative tools like *Photoshop* or *Maya*). All options are select-only: there are no sliders or color-pickers here, as many objects are user-created and non-editable. Everything in IMVU is designed around participating in social situations, like buying the ability to create custom emojis or gestures, or post to a virtual in-game Instagram-like feed[205] 494

25.2 A range of dress-up and avatar apps on the Android play store. (top-left)*DIY Fashion Star* is a fairly full-featured casual creator for outfits (notice my cats' picture on the skirt). The avatar then poses and is rated on a virtual Instagram clone (see chapter 16). (top-right)*Barbie Magical Fashion* is a typical narrow-possibility content-selection creator, but includes two socketable-gem jewelry creators. (bottom-right)*Live Portrait Maker* and *Monster Girl Maker!* both have large content-selection spaces, with complex multidimensional color-pickers, and in *Live Portrait Maker*, the ability to break left/right symmetry. (bottom-left)The app stores contain a spectrum of human avatar-makers, furry avatar-makers, and animal creators, with no clear distinction. *Create Your Own Unicorn* follows many of the patterns of avatar makers, but for an animated 3D unicorn instead. 496

25.3 The avatar-sticker-maker in Google *Allo*. 497

25.4 *The Soul Caliber* editor has the most interesting casual creator features. While other creators have more options, this editor is one of few that lets the player browse their options in ways other than manually setting a slider. After selecting a base character, they can use the editor normally, randomly roll a look, or jump to a "landmark" in the space, like pirate or knight (row 1). Randomness can jump to any point in the space, or can be constrained by era and region (row 2). As usual in these editors, accessories from a range of semantic domains can be purchased (row 3). Individual items can be colored, or the whole outfit can be color-changed as one (row 4). Accessories can be further spatially manipulated (with a gamepad), and personality, e.g. outbursts, can be changed (row 5) (`https://www.youtube.com/watch?v=ukpE7ERWtSY`). 500

25.5 Character creators to promote a voter registration campaign (https://voterhero.com/), South Park (http://southpark.cc.com/avatar), Harry Potter (http://www.harrypotterhogwartsmystery.com/), the Power Puff Girls (https://powerpuffyourself.com/#!/en/my-powerpuff/create), and the Simpsons (https://disneyandmore.blogspot.com/2008/09/are-you-ready-to-simpsonize-yourself.html). In each, the creator includes signifiers that are important to that world, like patriotic symbols or in-world gags, as well as items that are identity signifiers like glasses and hairstyle. 501

25.6 Users do not only represent themselves in casual creators. They often attempt to make caricatures of acquaintances or celebrities. (right)Ronald McDonald and Bojack Horseman in *Soul Caliber 6*'s editor[122](left)Barack Obama in the wrestling game *WWE 2K16*[167], and candidates for the 2012 election that I made with the *Spore* spaceship editor for marketing purposes for Spore[391]. 502

26.1 *Binary Fission*'s "Iris view" for changing the long process of manually inputing and evaluating an equation into a process of browsing and selecting generated equations nearly instantaneously. 505

26.2 Results of the survey. Most respondents would recommend the zine to junior or senior colleagues, and found it relevant to their work, even those who had not heard the term before. 510

26.3 511

26.4 512

26.5 513

26.6 514

26.7 515

26.8 516

26.9 517

26.10 518

26.11 519

26.12 520

Abstract

Casual Creators: Defining a genre of autotelic creativity support systems

by

Katherine Compton

Artists, musicians, writers and designers use tools to be creative, whether they are designing a personal opus or producing work for hire. There are many more people who are neither paid professionals nor historic geniuses, but who also enjoy being creative in a casual way. Can we design systems to help these casual users engage with their creative sides?

In this dissertation, I define and describe a genre of systems, "casual creators". Casual creators are interactive, generative systems in which the user explores a possibility space of artifacts, in a way that feels easy, pleasurable, and expressive. Casual creators do not provide the user with the total control of a "traditional" creativity tool like Maya or Photoshop. Instead, they empower the user with some combination of automation, generativity, surprise, or social/community features. Crucially, casual creators are not a genre of *new* systems, but a lens for looking at many existing (and emerging) systems that share these properties, including digital systems like *Kid Pix* and *Spore*, as well as non-digital systems like the *Spirograph* and *Mad Libs*.

Contemporary creativity theory provides clues about *where* creativity happens ('environmental' and 'embedded; theories of creativity), and *when* creativity happens ('flow' theory, improv). These theories have already been used to design creativity-augmentation systems in the field of *creativity support tools* and *computational creativity*. This dissertation combines these theories of creativity with theories of interactivity and generativity, to anticipate and catalog common patterns that can be seen when a user is creatively exploring a generative space.

The dissertation concludes with an initial set of ten patterns, including patterns that emerge from technical aspects of the system's design and others that emerge from the system's position in a social environment. Along the way, a couple hundred systems that are casual creators *in some way* are noted and analyzed. The primary contribution of this dissertation is the term "casual creators" (and its associated frameworks), which has been shown useful for understanding existing and new systems, and has seeded a community of practice around such systems.

To my parents, Russ and Joan.

Acknowledgments

Thank you to my parents and family for your patience and support. Thank you to my classmates and colleagues for the inspiration, pushback, and evolution of my ideas. Thanks to my advisor of 15 years, Michael Mateas, as well as all the other faculty who have guided my path and shown me new horizons.

Chapter 1

Introduction

> "A good way to rid one's self of a sense of discomfort is to do something. That uneasy, dissatisfied feeling is actual force vibrating out of order; it may be turned to practical account by giving proper expression to its creative character." - William Morris, quoted perhaps apocryphally in the workingman's magazine "The Iron Age", 1909

Why is **creativity** important? We can see the importance of *professional creativity* in the sales figures for Pixar movies, or the return on investment in fashion or video games. Workplace creativity is likewise promoted in best-sellers and magazine articles. Museums and libraries are dedicated to the output of *historically creative*[1] people. If tools can assist this kind of creativity, it seems worthwhile to build them. However, there is a whole different kind of creativity that can be called "everyday" or "little-c" or casual creativity. This dissertation will define and explore a category of systems that I call **casual creators**. Professional creativity software that is designed to support the needs of productive creativity[2], but casual creators are systems designed to support *autotelic creativity*.

Casual creators exist halfway between productivity software and entertainment

[1] "Historically creative" is a common term in creativity studies to denote field-changing people of historical significance, which can include people in the modern era as well.

[2] For example: systems like *Maya*, *Photoshop*, *Garageband*, and *Autodesk Inventor*.

software: like productivity software they are designed to create artifacts (virtual or otherwise) during their use, but like entertainment software, the purpose of the experience is primarily diversion or pleasure, and the production of artifacts is only secondary to that purpose. These systems privilege enjoyment of *process* over quality or control of the final artifact that is created. As Craig Hickman notes in the first two of his 15 guiding principles for designing the early and influential casual creator *Kid Pix*[s113]:

> "1. (The Prime Directive) The program should be extremely easy to use. No manual should be needed and program features should "explain themselves" through use. All tasks should be able to be performed in the simplest, most straightforward way. The program should go out of its way to meet the user.
>
> 2. As long as The Prime Directive is not violated, every opportunity should be taken to make the program surprising and satisfying to use. No opportunity should be missed. The process of making a picture should be as important as the picture produced." [193]

There are many[3] casual creators in app stores, but they are variously categorized as games, entertainment, or productivity. Casual creators may be designed to produce any kind of artifact: physical crafts, digital images, animated characters, melodies, 3D printed sculpture, or story ideas all currently have successful creativity systems designed to help users produce them, though they are scattered across a wide range of fields, and rarely compared to each other. In the course of this dissertation it will become clear that, despite these differing categorizations, when examined as a collection, very distinctive and particularly effective patterns are shared by all casual creator systems. Furthermore, these patterns are very different than the design patterns we are familiar with from productivity-focused creativity software.

[3]Thousands at least, and more than I've been able to directly survey.

It seems self-evident that historic and professional creativity is important. Why is **casual creativity** important? For an example, we can look at the advantages that creativity provides in medical and psychological intervention outcomes. Creative work can be used to improve cognition and mood, as well as distract from negative emotions or pain. A qualitative study of hobbyist needleworkers suffering from depression found that creative work both distracted the crafters from negative feelings and created a context for positive experiences (like improved self-esteem, "a sense of empowerment", and social contact that further defended against depression). [317]. A similar study of a population of aging 60-93 year olds found comparable effects. The arts distracted them from physical and psychological pains of aging, and provided higher meaning and purpose to actions: "When a person is creative, he opens up doors and it's important he expands, and creativity is the means for that expanding. While you are creating you are taking your mind and building a wall between you and the undesirable things in your life." [136] This quote suggests that we may also turn to philosophers and humanitarians to, if not understand creativity, appreciate what it gives us that *isn't* measurable in a study.

The act of creating something can be seen as a creative act, a productive act, or both. A factory worker in a 19th century factory (or a modern one) might have produced something useful, or created something that wasn't in the world before, but was not being 'creative'. To quote the early humanitarian John Ruskin,

> "You can teach a man to draw a straight line, and to cut one; to strike a curved line, and to carve it; and to copy and carve any number of given lines or forms, with admirable speed and perfect precision; and you find his work perfect of its kind: but if you ask him to think about any of those forms, to consider if he cannot find any better in his own head, he stops; his execution becomes hesitating; he thinks, and ten to one he thinks wrong; ten to one he makes a mistake in the first touch he gives to his work as a thinking being. But you have made a

> man of him for all that. He was only a machine before, an animated tool."[334]

If one performs work precisely according to an external specification, even if the specification was originally creative, there is no creativity — no chance of mistake or failure, but also no chance of expression or greatness. As we will see later on, this presents us with a potential spectrum of creativity, from no choices to only one or two choices, to a broad and branching path, to an open and markerless field.

Productivity and creativity should both exist in labor, to improve the human condition of the laborer. "We are always in these days endeavoring to separate the two; we want one man to be always thinking, and another to be always working, and we call one a gentleman, and the other an operative; whereas the workman ought often to be thinking, and the thinker often to be working, and both should be gentlemen, in the best sense." Ruskin concludes. To do mindless work is damaging to the human spirit, but so too is the retreat into unproductivity. Ruskin's contemporary, William Morris, among many others, holds that creating connects one with the whole of humanity, becoming more human through the process:

> "A man at work, making something which he feels will exist because he is working at it and wills it, is exercising the energies of his mind and soul as well as of his body. Memory and imagination help him as he works. Not only his own thoughts, but the thoughts of the men of past ages guide his hands; and, as a part of the human race, he creates. If we work thus we shall be men, and our days will be happy and eventful." [279]

Creativity, then, is useful. It produces economically valuable goods, it makes us happy, connected, and fulfilled. Those reasons alone could justify investigating methods to improve and encourage creativity.

Jerome Bruner, writing in *On Knowing: Essays for the Left Hand*[40] points out the "shrillness to our contemporary concern with creativity". As humans we reasonably know that there is something important and ennobling about being creative, but this is too soft a reason for hard scientists to be interested in studying it. To make up for the deficit, one can appeal to the usefulness of creativity in therapy, or to the increased productivity that creativity will produce in science and the arts [368] or even to create better CEOs, as the introduction to the Handbook of Creativity suggests[394]. Bruner reminds us, while we can still consider these arguments, there still exists non-scientific, non-practical reasons that drive us to study creativity, and its worthwhile remembering these.

> "Yet there is good reason to inquire about creativity, a reason beyond practicality, for practicality is not a reason but a justification after the fact. The reason is the ancient search of the humanist for the excellence of man: the next creative act may bring man to a new dignity."[40]

To be human is to create, and we create to further humanity.

1.1 Mission of this dissertation

Creativity is one of humanity's defining traits. We create to make useful or necessary things, to make pleasing things, and sometimes for the pleasure of creation itself. We have made tools to support creativity, digital and non-digital, perhaps for nearly as long as we have had creativity. And we have studied those tools, but much of that study has focused on the tools that help us make the beautiful or useful things, and less on those that support creativity for its own sake. This dissertation is dedicated to the tools of creativity-for-creativity's sake, and to the support of joyful, aimless, indulgent, exploratory creativity and the celebration of that as its own source of value. **To that end, this dissertation**

presents a new genre of interactive systems called "casual creators", as a recognition of the joy of creativity and the importance of that joy within our interactive systems.

Casual creators are interactive systems that encourage and privilege fast, confident, and pleasurable exploration of a rich possibility space; leading the user to experience feelings of creativity, both in a sense of ownership of what they make, and of their participation in an environment of creativity.

In one sense, this genre already exists. Hundreds (or thousands) of casual creator systems are already part of people's lives. This encompasses systems like *Lets Create: Pottery*[s123], a virtual pottery wheel for mobile phones and physical pottery wheels. It includes the 1989 program *Kid Pix*[s113], where children could draw with generative virtual brushes that came to life as they drew into a pixel buffer, and virtual reality drawing tools like *TiltBrush*[s235], and *Oculus Medium*[s153] where 3D brushes come to life in midair. In this dissertation, the reader will see tools to playfully create poetry, abstract art, songs, robot-made paintings, self-portraits, jewelry, virtual cars, and monsters. We will explore how physical crafts and toys like the *Spirograph*[s214] and Kaleidoscope[s110] share patterns (and even technology) with modern casual creators using advanced geometry simulations and machine learning.

1.1.1 Definition of a casual creator

A casual creator is a system that

- privileges enjoyment of the creative process above productivity
- produces artifacts within a limited-yet-meaningful domain space, enabling automation and support, both passive (encoded into the domain model and system constraints) and active (responding to user actions)

- supports a state of creative flow by restricting choice and preventing hard failures while allowing rapid iteration
- results in the user's feeling of pride and ownership toward the produced artifact, and sense of pride in their own creativity.

1.1.2 Examples of casual creators

Many systems exist which are casual creators *in some way*. Many such systems were designed with casual creator goals (without knowledge of the term), and achieved these goals with varying degrees of success[4]. A few systems were not intended as casual creators (e.g. traditional games or professional creativity tools), but were picked up by users who played counter to the expectations of the system, hacked in new features, and adapted these systems into being casual creators.

To see how the casual creator terminology helps us understand such systems, let us examine several examples of systems with *some casual creator properties* that all feature a similar mechanic of exploring an landscape and capturing images[5].

Eastshade[s60] is a quest-based first-person exploration game, where the player is a shipwrecked artist trading favors and fetch quests for the ability to explore more of an island full of whimsically-Victorian talking animals. The player can gather raw resources (cloth and wood) to create blank canvases. These can be combined with "inspiration" (a resource renewed by finding new map areas) to enable a "painting mode", where the player frames some portion of the landscape, and it is rendered onto the canvas. These paintings can then be used to fulfill requests of the townspeople, if they contain particular elements (portraits, pictures

[4]One contribution of this dissertation is to provide a post-mortem "forensic" framework for understanding why some systems succeeded and some failed at these goals.

[5]Proposed genre name: "walk'n'painters"

of beaches, particular in-game objects). One creator of *Eastshade* describes the evolution of the painting mechanic, and the need to make it feel "juicier" than a screenshot:[430]

> ""The first time Jaclyn proposed the painting thing, it sounded really hard to me, so I like basically took what she said and took all the good stuff out,' laughs Weinbaum. 'I initially put the paintings in like pickups. So we would have these paintings in the world and they were like little pickups. This is so embarrassing to even say, and even she at the time was like, 'That's not painting. That's totally dumb. Don't do that under any circumstances."' Having to rethink how painting was incorporated, the team settled upon a screenshot mechanic instead. Though this would mean players would have less control over how they could interpret their surroundings, this mechanic would ensure the game was still accessible for even the least talented of painters, allowing individuals to simply stretch the parameters of the canvas on the screen and capture the image held within. [...] "We knew we had to come up with something that was a little bit cooler than just the thing popping up in front of you," states Weinbaum. "Because I mean it already is dangerously close to just like a photo or a screenshot, so we wanted at least to support the fantasy with some sort of animation.""

From these quotes, even though the designer has probably never heard the term "casual creator", he identifies a relationship between the loss of fine-grained control and resulting ease of use, as well as the need to have *some* creative control over creating paintings, and some generative component to make the paintings feel more *special* than the player could normally achieve.

This kind of painting mechanic requires a painting "object" that is comprised of two components: the visual representation of what the player has "captured", and a representation of what is in the painting (a list of entities calculated to be "in" the frame). Regardless of the visual splendor (or lack thereof) of the painting, the game must provide feedback based only on the list of entities. This mechanic can be seen in past games like *Dead Rising*[291] and *Pokemon Snap*[395], where the player gains points by "capturing" different in-game objects or mo-

ments. Given the definition above, are these systems casual creators? There is certainly "automation and support" in these systems, as the games engine renders a photo-realistic scene that the player could not themselves draw. The space is constrained, yet rich: while a player *could* take a photo of a blank wall, there is little repercussion for that error, and the environment is designed to provide many more interesting things to draw the camera instead. The question, for these systems, is whether the process is one of playfulness or productivity, and whether the player feels a sense of creative ownership over the artifact. In *Pokemon Snap*, the player's movement is on rails, and the challenge is to click the shutter at the right moment to capture a moving Pokemon while it is on screen. There are few creative options here, only an objectively better or worse time to shoot. *Dead Rising* lets the player walk around, explore, and find new scenes to photograph, though its calculations for point-scoring and photo evaluation are so specific that a player might experience it as a utilitarian point-maximizing exercise, rather than playful creativity.

Eastshade extends this mechanic in a few ways that push it towards being a casual creator. Players can drink several different kinds of mind-altering potions which apply a lightly stylistic filter to the captured scene, giving slightly more control (and generativity!) to the mechanic. While the paintings are still evaluated based on whether they contain the necessary entities, the requirements are simple yes/no, limiting the need to optimize for maximum points, and leaving a large possibility space of equally-valid pictures that the player can explore. I asked a friend (Aaron Reed) and enthusiast for the game, about his experience:

> "I think some of the stuff like teas that filter your vision, the ability to control framing, and the fact that he environment is deliberately designed to contain lots of hidden nooks and crannies, is enough to make you feel like you're "finding" good shots to take.[...] I think it's also important that the characters give you lots of positive feedback about

> your paintings, telling you how much they liked them, hanging them on their walls, etc. The game makes them feel special and important."

Despite the game not being able to distinguish a beautifully composed shot from an indifferent one, it is willing to celebrate and elevate the player's output, through visual framing, generative effects, and in-world commentary. As we will see in chapter 16, these fictional encouragements can have an effect on creative confidence, even when they aren't based on actual knowledge or evaluation of the artifacts.

Firewatch[s74], like *Eastshade*, is a first-person game about exploring an attractive environment: in this case, a sprawling 1970s National Park of towering trees and craggy canyons (and a few drunk teens). The player (a ranger named Henry) finds a virtual disposable camera, and can take a limited number of photos. These can only be viewed online after the game is completed, and even printed out in a vintage envelope of photos. While this lends intensity, realism, and mystery to the experience of taking photos, it also raises the stakes so that players may not be comfortable making risky or playful photos, so as not to waste a limited resource, and raises the "price" of mistakes, as one reviewer noted:

> "I think the only way in which they sort of fit the conceit of having a film roll is that you get no do-overs so my own images are ones I'm not entirely happy with – a sensation I know well from film photography. The way they're similar to most of the prints I own is that I dislike them. Or at least feel ambivalent about them."

She also describes a separate experience, where she uses the in-game shots, but edits them in contemporary photo-editing software, and feels a greater sense of ownership:

> "Perhaps it's because I got to crop them and that does apply an edit function, shifting a butterfly out of the centre of the shot and into the one third position which I far prefer. Perhaps this set of shots let me

> be myself whereas in the others I should have been role-playing Henry for it to work properly."

Firewatch's photo mode, like *Eastshade*'s painting mode, is only one part of a larger game. But the creative experience in *Firewatch* is less clearly a casual creator (despite still being a creativity tool) because of its high risk and limited explorations of the possibility space[6] It does however, use an uncommon but powerful pattern seen in casual creators, the instantiation of the user's artifact into a physical object, as we will see in chapter 23. This pattern can turn a moderately interesting artifact into something profoundly "magical" and increase the sense of creative pride and ownership.[7]

Not every casual creator is intended as such. Later in this dissertation, we will see ways in which decidedly non-casual systems like *No Man's Sky* and *Dwarf Fortress* spawn subcommunities for casually-creative counterplay within advanced technological systems. *Grand Theft Auto V*[s91] (and its ancestors) is *primarily* a game about driving fast and shooting people. It is likely that most players never take a screenshot, or, if they do, they are trying to showcase in-game virtuosity, rather than creative composition. Yet this game has seen a lively community of players self-organize to create new ways of using the game to create visual art.[450] The game contains an in-game scene editor which can be used to stage and focus new scenes, and track-building tools that users re-purpose to create virtual photography sets like pirate ships. Rockstar, the developer of the *GTA* series, also provides an online social network[8] where users can share resource requests and share (and "like") their photos. *GTA* is a much larger and more popular game than either *Eastshade* or *Firewatch*, so its community of creative counterplayers is

[6] *Eastshade* also artificially limits creative exploration with its resource constraints.

[7] `https://www.reddit.com/r/Firewatch/comments/9u03hf/has_anyone_used_the_photo_mode/`

[8] `https://socialclub.rockstargames.com/photos`

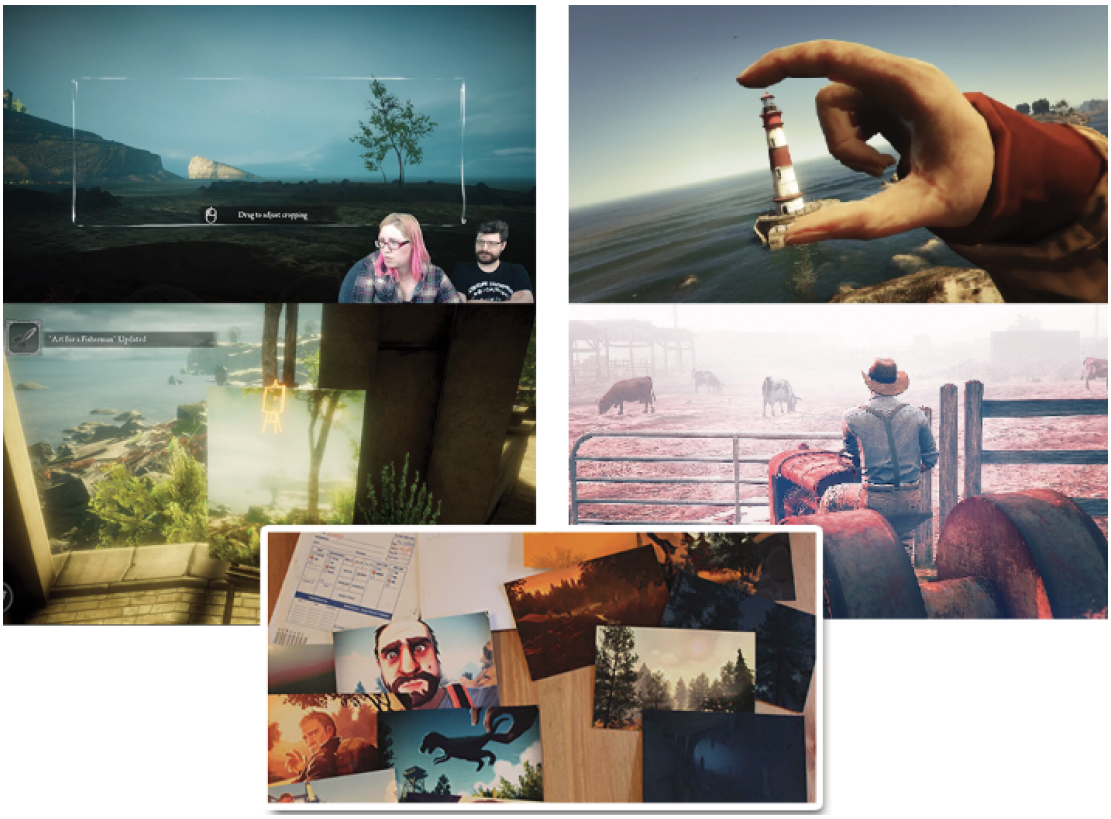


Figure 1.1: (Left) Isaac Karth and I creating paintings in our ScholarsPlay livestream of *Eastshade*. (Bottom) Phillipa Warr's printed photos of *Firewatch*. (Right) GTA photos by users *SnowGoose95* and *Alex Crowley*

larger, and yet it is still surprising how well-developed the creative community is and how well-built Rockstar's community-creation tools are. Each of these three games is a flawed casual creator in its own way, with some features that limit the player's creative playfulness, and other features that create creativity safety, surprise, generativity, or creative community for casually-creative players.

In this dissertation, I will explore ways to understand systems like this, and to enable those building new systems to think critically about the design decisions that support or hamper casual creativity.

1.2 Research Questions:

- What defines a "casual creator", and what new experiences do casual creators enable?
- How are those experiences predicted or explained by existing theories of creativity?
- What frameworks do we need in order to understand and analyze casual creators?
- Are there subcategories within casual creators that have shared properties?
- What set of design patterns provide prescriptive guidance for designing new successful casual creators?
- If these design patterns are used to design new casual creators, do they ease and improve the design process?

1.3 Contributions:

This dissertation attempts answer these questions, as well as to provide a useful starting point to a potential community of "casual creator practitioners". To this end, the dissertation will achieve the following, in language that is accessible to a range of communities with differing (academic/industry/hobbyist, expert/novice, theory/practice) backgrounds:

- define what casual creators are (see above)
- identify how casual creativity differs from task-oriented creativity, and why we expect to find different patterns to support it than those that have been identified and studied in professional and virtuosic creativity

- define a starting set of terms and frameworks (chapter 6 and chapter 8) for understanding casual creators, to establish a shared foundation upon which we can build additional knowledge of casual creators
- lay out the principles that underlie casual creators: interactivity, generativity, possibility spaces, navigation, and social interactions, and examine how they work in interactive or generative systems in general, and also how they combine in casual creator systems
- identify the shared properties of casual creators across many diverse genres and mediums, from character creators and game-makers, drawing tools and pottery wheels, poetry bots and programming languages
- propose a handbook of patterns, a non-exhaustive menu of tools that can be applied to casual creator systems, or systems that seek their properties
- motivate the identification, study, and creation of casual creators, and catalyze the formation of an interested new community of research, development, and construction of shared knowledge

The dissertation will start with a few chapters on why we care about "casual" creativity, as well as what we know about creativity (and assisting creativity). It will then explore some frameworks of interactivity and generativity, and propose some new language to talk about those fields. The second half of the dissertation will be devoted to outlining common properties of casual creators and cataloging examples. Some of these properties are determined by the kind of possibility space that a creator uses in its generative pipeline, while others are "general purpose" patterns that can be applied to many kinds of casual creators. Finally, I conclude with a brief (in length, scope, and duration) evaluation of these concepts by a few interactive-creativity practitioners.

1.3.1 Structure of the dissertation

- **Motivation:** I hope that the term "casual creativity" (and its associated frameworks) is adopted by practitioners and researchers. What do I expect to happen next if it is?

- **Central themes:** Casual creators trade *power* for *control*, and are interactive tools to navigate a generative possibility space. This dissertation will revisit these two themes repeatedly, so I give an introduction to what they mean.

- **A summary of theories of creativity, especially for autotelic creativity:** What do we know about creativity, and how it is a product of not only the individual, but also the embodied interaction with the artifact, and the social environment of creativity that they are embedded in.

- **A survey of systems to support creativity:** If we know *what* creativity is, how do we create ways to support it?

- **Frameworks of interaction and generativity:** I review existing frameworks for generativity and interaction to answer the question "how do we understand a generative possibility space through interacting with it". I then suggest two new frameworks to describe interaction and generativity in casual creators (the "grokloop" and the "generative framework of generativity").

- **Ways of categorizing casual creators:** The next seven chapters deal with the different kinds of possibility spaces that casual creators have, and how that affects how users can navigate through and understand them.

- **A general introduction to possibility spaces:** and some terminology ("expressive range", "locality") that we can use to discuss the properties of possibility spaces.
- **"Vectorspace" creators:** Creators with an N-dimensional cube-shaped possibility space have some unusual affordances, and ways of navigating them (like genetic algorithms).
- **Drawing tools:** Making a gesture to draw a mark (or drawing) is a strangely common casual creator subgenre. This chapter discusses why.
- **"Slow" creators:** Some casual creators have slow feedback and iteration loops. How do we make systems that speed up editing the artifact or interpreting the results of those edits when the domain is *slow* (like novels)
- **Casual programming languages:** These are interesting subcategory of slow creators, with their own problems. What makes a programming language (or rather, its environment) a casual creator? A whole chapter is then devoted to Tracery as a casual programming environment.

- **Collections of design patterns:** The next ten chapters are sets of *design patterns* that we see across many casual creators. Unlike the previous chapters which create *subgenres* of casual creators by their possibility space properties, these patterns can often be applied across many types of casual creators.

- **An evaluation of the zine version of this dissertation:** Are these frameworks and new terminology of use? Will "casual creators" catch on as a term of art or useful research area? I survey several industry experts on

their opinion.

Along the way, there will be short case studies of some of them many casual creators I have made as part of this dissertation's research. Each one of these case studies focuses on some property or pattern of casual creators to explore their use in *creating* a casual creator from scratch. There are also a *lot* of systems cited in this dissertation; you will find them all listed in the back, and linked to from their analysis in the text ("S5" for the fifth system, etc).

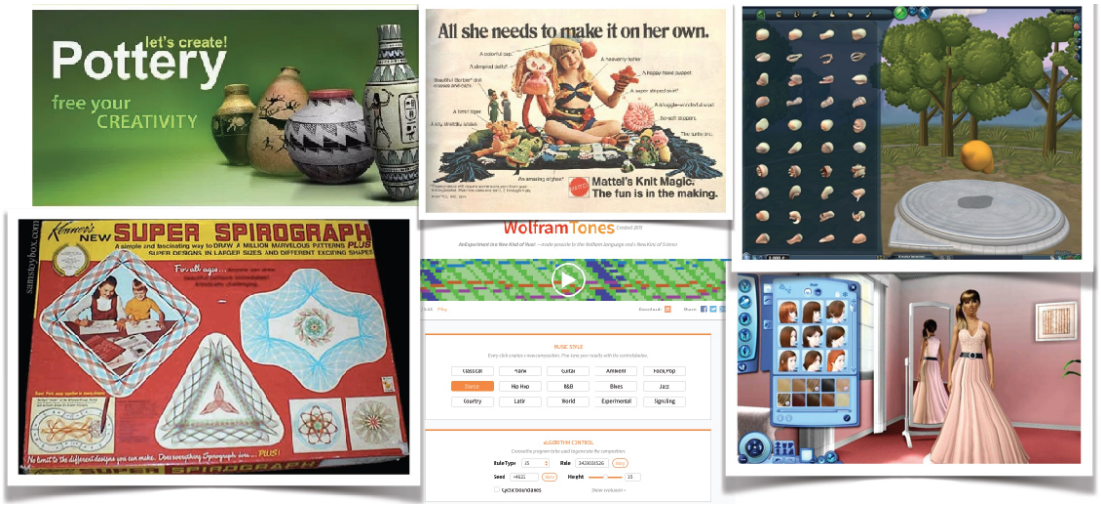


Figure 1.2: A few casual creators and their artifacts, to show the range of the software (and non-digital) systems that we will see in this dissertation

Chapter 2

Motivation

Casual creators exist.

You can discover this for yourself by searching "create" or "creator" in any app store. You can compare the character makers across dozens of games from *Stardew Valley*[s219] to *World of Warcraft*[s244] to *Miitomo*[s134]. Consider why are there are so many sound apps where you paint noises into a grid (*Mario Paint*[s131], Wolfram Tones[s243], *Chrome Music Lab Song Maker*[s88], Google's *Oskar Fischinger Doodle*[s156]). Is it a coincidence that the 80s drawing toy *Kid Pix*[s113] is cited in so many reviews for Google's VR drawing app, *Tiltbrush*[s2351234]? Why is one of the most popular creativity apps in the Apple and Google app store a *virtual pottery wheel*[s123]?

Creativity systems for casual users are a huge field, and yet, as I began this dissertation, it seemed that no-one else could see the relationship between the

[1] https://www.reddit.com/r/oculus/comments/67wzdn/tilt_brush_v10_sharing_to_the_online_gallery/

[2] https://arstechnica.com/gaming/2016/04/learning-how-to-vr-with-tilt-brush-htc-vives-killer-app/

[3] https://www.theverge.com/2014/9/24/6835271/painting-in-virtual-reality-will-blow-your-mind

[4] https://medium.com/@davidaslanfrench/a-brief-review-of-quill-vr-fcb876b2a7df

character creators, the music makers, and the drawing tools; and certainly not the connection to non-digital creativity systems like pottery wheels, spirographs, and MadLibs. But this connection seemed obvious to me, and not only obvious, but productive. If we understand the connections between all these disparate systems, we can learn about what patterns *work to support* casual creators, and what *problems we may encounter* or *false paths* we must beware. How much design knowledge is known by all the creators of these systems? How little of that knowledge has been communicated?

This dissertation itself was inspired by working on *Spore*[s215], which shipped with *Creature Creator*[s216] editor inspired the best of user's creativity, yet also released an Adventure Creator that inspired little more than frustration.[5]

My strongest motivation in this dissertation[6] is to create tools for this new potential community. These are reflected in the research questions from chapter 1, but are grounded what *effect* they have on the community.

- a name to organize around, to find shared goals and fellow travelers amid otherwise disparate domains [7]
- a shared set of terms and frameworks, so that these new colleagues can communicate usefully

[5] One memorable experience while working on the *Spore Galactic Adventure*'s pack was when they sent a few of the *Spore* developers (including me) to L.A. to work with the Robot Chicken creative team, to build some humorous sample adventures. With some of the best comedic minds, and some of the best people at using the system (we'd built it, after all), we could barely put together a half-dozen functioning adventures in 8 hours of work. In hindsight, we should have compared this with the apparent ability of most 5 year olds to create Spore creatures in a few seconds, and recognized the sinister omen. For more on *why* it didn't work, turn to the chapter on "Slow Creators". Clearly, the same team could make a casual creativity system that was wildly successful at its goals, yet we didn't have the knowledge, or the language, to repeat that success in a slightly different domain, or even explain why it failed.

[6] Aside from earning the degree.

[7] Goal conversation: "Oh, you do sound art?" "Yes, mostly casual creator sound toys" "Hey, I do casual creator story systems! What channels do you use for community artifact sharing?"

- a starting set of patterns (and examples) drawn from across domains, showing that there is value seeking out systems and knowledge from casual creators of other domains
- best practices and known difficulties, to help those building new systems avoid the errors and build upon the advantages of past systems
- a manifesto and motivation that casual creativity *is* of value, and a reminder that unskilled, playful, unserious users can be an audience as valid as the virtuosos and professionals targeted by other creativity software

2.1 Labelling an old genre

I should note that the genre that I'm defining already exists, or, the works that fall into the genre already exist, and many are older than software. Craft kits have existed since the Victorian era saw the rise of 'crafting' as a pastime rather than an obligation [154]. Central to the design of a craft kit is the trade-off of limiting the range of possibility space in exchange for the assistance that can be provided in a space of precise specifications.

A dedicated artist's studio would have many colors of paint and kinds of brushes, capable of producing any kind of image, and a blank canvas. In contrast, a paint-by-numbers kit includes only the colors necessary for the painting supplied, with guidelines and coloring instructions to follow, capable of producing only one kind of painting. Similarly, a knitter might have enormous varieties of yarns and needles to make hats, blankets, scarves or socks. The complexity that is beneficial to an expert is overwhelming to a novice user. We can see this principle in the long history of children's knitting products, designed, like a casual creator, to limit the domain space while providing assistance and automation and

preventing failure states, in Figure 2.1.



Figure 2.1: An assortment of assisted-creativity toys. A vintage paint-by-numbers kit[s159], a knitting nancy, a modern day *Rainbow Loom*[s184], the Mattel Knitmagic[s116] which was one of my favorite childhood toys (from Flickr user *x-ray_delta_one*, from Flickr user *sallysetsforth*, image from Wikipedia, `https://theknittingneedleandthedamagedone.blogspot.com/2013/01/knit-magic-isnt.html`)

The products made by these craft toys are restricted enough in scope that a lot of assistance, and even some automation can be provided. Making it much easier to use, and harder to fail at than free-form knitting, at the price of limited control.

2.2 Of genres and usefulness

Why identify and label this genre? If there really were a genre here, wouldn't the market have identified it already?

Genres are very useful in marketing, both in identifying and selling to a target market. Identifying role-playing games as a genre creates a targetable audience of those who buy and enjoy RPGs. Although the genre's name presumably specifies the properties of the works in it, the current expectation of a role-playing game is not 'playing a role', a description that applies to almost all games, but a very specific set of expectations of regular forward progression of stats, an expansive and complex set of power-ups and abilities, and a fairly narrow set of canonical settings (medieval fantasy). This narrow focus allows fans to self-identify as RPG players. Marketers and developers can provide a predictable set of features to sell to them, and makes a whole field of games more unified and economically viable.[235].

As a way to identify marketable populations, genre often becomes bound up in demographic considerations; genres are frequently tied to gender (dating sims vs RPGs), nationality (Japanese and American RPGs) and age, with a different genre "kids games" used to market to children regardless of gameplay. Genre is effective for economic purposes, but can be frustrating for academics or designers looking to use genre as a lens on design.

Genre can also include a set of design patterns. For RPGS, many patterns have been established over time as ways to support the core mission of RPG gameplay, like numerical experience points, shopkeepers and 'trader trash', crafting, and parties of NPC assistants, as well as known local maxima for optional features, like choosing between permadeath, autosaving, or multiple saves.

A design-focused genre can be defined as a set of mutually-supportive and co-

occurring design patterns[235], that is, given the goals of a work, what patterns support those goals, and what patterns support and strengthen the effect of *those* patterns? As a design pattern is often considered to be a solution to a problem, the genre needs to (explicitly or implicitly) also encode the goals of the genre.

So, in defining "casual creators", I will need to specify them as both the *properties* of the genre, and the *goals* of the genre. For the sake of simplicity the previous definition encompasses both—"privileges enjoyment" is a goal, but "limited domain model" is a property, for example. But it is useful to break them down farther, especially for purposes of identifying and critiquing current examples, and importantly, building more effective future works.

2.3 Effects of having a community

An enlightening parallel is the nearby area of video games. Games and casual creators are closely related: both are autotelic experiences. As an object designed for pleasure, they often focus on spectacles like appealing graphics and juicy animation. They don't support a *productive* purpose, [47], and are sometimes denigrated or ignored as unserious pursuits unworthy of serious study [64].

The now vast field of video games research eventually grew out of this understudied space. A few blogs and books and meet-ups [175][392][143] became immense conferences, competing journals, games studies and game-development university programs, diverse and thriving ecosystems of games writing and games research, within just a decade. How did games develop in this way? Why? What would it take for casual creators to see a similar set of community structures develop? What is gained from these structures, and what will happen to the field of casual creators without them?

A curious novice practitioner has their choice of tutorials, frameworks, text-

books and societies available to guide their further steps. This dissertation, and my other outreach work[8] are one attempt to build resources for new designers of casual creator systems, but my hope is that others will also use the language and frameworks seeded by this dissertation to start creating their own resources. While there are no Casual Creator conferences yet (nor workshops, journals, or even a jam) the language is getting adopted by several academics, and it may be starting to inform developers (see the chapter 26 for examples of its adoption).

2.4 Casual creators as an intervention in Computational Creativity and Artificial Intelligence

Casual creators may provide a solution for some issues in Computational Creativity and Artificial Intelligence. Adding a user (or in the case of AI, considering the mental state of the user) can be an intervention that increases understanding and trust of these otherwise unknowable and monolithic systems. Computational Creativity, as a field, is deeply interested in the question of *whether* a computer can be creative (as well as how and if humans perceive that creativity)[306]. The work from this community is often (or perhaps always) generative in some way. As we will see later in chapter 6, these generative systems can be viewed as pipelines: from some input, to some output. In almost every system, there *is* input from a human, whether the input comes from the user of an interactive system, or the data hard-coded by the original system designers, or from 'accidental' or 'unaware' users like Twitter feeds or news stories or remotely-tracked crowds[9]. Thus, even

[8] See galaxykate.com

[9] This excludes a few systems that take input from non-human agents like waves or bacteria, but even then, we can often usefully speak of them as 'a user'.

in systems which assume that the system itself must be creative, because creative work is being generated in the "absence" of a human interactor, we can see the system as interactive *as it is being designed*. Its (probably) human designer is *in dialog* with the system: changing the code, its source materials and methods, even the random seeds, and then evaluating the results and modifying the system. The process of designing a generative system is an *interactive* movement through a *possibility space of possibility spaces*, even if the "final" system is not interactive. Thus, building a computational creativity system could be done through a casual creator, with the same issues (generativity, interpreting results, navigating a possibility space), and could use some of the tools of this dissertation to analyze and improve systems in progress.[10]

Additionally, casual creator ideas provide a bridge for evolving non-interactive generative systems into interactive systems. The variety of structures with which casual creators implement "interactivity" (from simple immediate-feedback editors, to semi-autonomous simulations, to crowd-controlled multi-agent systems, or communally navigated spaces) could also give computational creativity systems more models to implement their own interactivity.

2.4.1 Showcasing Computational Creativity

Why would a computational creativity project need interactivity? As we will see in this dissertation, interaction allows users to navigate and map a possibility space defined by a generative pipeline. If a user cannot interact with a system, they cannot possibly understand the space simply by being given a selection of pre-generated artifacts (or even a single artifact).

Computational Creativity uses models of creativity to produce artifacts that

[10]We will see Michael Cook's *Danesh*, a casual-creator-for-generators in the later chapters about possibility spaces.

mimic the products of human creativity, such as music, poetry, portraiture, abstract art, recipes, and memes. Many of these systems [62][374][181] function as artifact-on-demand blackboxes: a button is pressed (or otherwise invoked) causing the system to crunch through its computations and eventually output an artifact. Interaction with the system is limited to setting preconditions or selecting seed artifacts and deciding when to push the button again. Partly this may be a symptom of the long runtime of the algorithms: many of these systems were designed in a slower era of computation, and some modern systems [57] require laborious machine learning on the provided seed artifact. But it is certain that there are smaller, lighter versions of algorithms that could be run in real-time (or older generators that could take advantage of modern hardware speed) to achieve interaction within seconds, or even milliseconds in response to continuous user input. A recent project at ICCC presented both a traditional generator, and an interactive version[219]. Deciding *how* to respond to user input may be the limiting factor, and one I hope to lift with the design patterns in this dissertation.

However, if they interact with a system, they can build their understanding of it, know what it does, and have confidence and trust in their model of what it does. As the user learned about the algorithm's inputs and outputs, they would build an ever-improving working mental model of the system. This, in turn, would allow them to find paths to better parts of the system's possibility space, discovering better content. Applying casual creator design patterns to computational creativity turns users into agents for finding the best and most interesting content, further improving the output of the system. An additional benefit is the set of casual creator patterns for allowing users to share, curate, and self-promote through content that they discover or create. By cataloging the generated content from the system, users are also discovering and publishing

interesting *features* and *affordances* of the system, many of which may not have occurred to the original system builders. Additionally, each time a user posts content, self-promoting themselves as the creator/discoverer, they also promote the system that created it. This very lucrative side effect can be see in the games which can be most entertainingly played by Twitch stars becoming commercial hits, like *Minecraft*[s136] and *Goat Simulator*[s86]. The effect can even be seen in the re-invigoration of older games like Facade as a performative "Lets Play" platform.

2.4.2 Turning an intimidating algorithm into a trusted collaborator

Artificial intelligence methods, and recent advances in machine learning, have led to the creation of many extremely powerful tools [349]. Many of these tools run autonomously with no interactivity, operating on user's data with no controls other than to enable or disable the algorithm's access to data. Since these are non-interactive systems, users cannot interrogate or explore the systems to understand them, and misinformation is allowed to persist. Users develop very inaccurate and often unpleasant and unflattering models of what the algorithm is doing. They don't *trust* these systems, and don't trust their limited knowledge of them. This results in negative user experiences and general increased distrust in Artificial Intelligence. [33].

When these systems become interactive (in a more intentional way than providing or withholding data) they can take advantage of the mastery that comes with allowing the user to interactively test the consequences of a system. In this case, "mastery" has two equally useful meanings: users can build a predictive model of the system that is accurate enough for them to express intention through it, and *also* users will feel that they can *control* the system, if it is a sys-

tem that they can prod, poke, experiment with and examine at will. We see this at work in systems like *Youragebot*[s246], *Writelike*[s245], and Microsoft's dog breed guesser WhatDog[s242]. Users intentionally submit odd or confusing information to the system to get a sense of what it can and can't do, and also to playfully exert dominance by "putting one over" a so-called "intelligent system".



Figure 2.2: Janelle Shane's interrogations of image-labeling algorithms, using out-of-context sheep to explore the edges of the algorithm's abilities

Janelle Shane [356] has become a celebrity in AI by playfully interacting with intimidating or hyped AI systems. Her many works include using LSTMs to create new paint colors [359] and knock-knock jokes[11], experimentally interrogating photo labeling software about its sheep obsession [357], and encouraging an image-based chatbot to interpret scenes from Star Wars[12]. Aside from the humor, why is this useful work to be doing in Artificial Intelligence? Many people take AI claims at face value, whether these claims are made by companies, researchers, journalists, or journalists uncritically or mistakenly interpreting researchers and companies [268]. From personal experience, even diligent explanations of how things work don't shake people's media-influenced perceptions of AI as opaque and threatening and smart. *However*, Shane's work of playfully and non-technically (she does not identify as 'a coder' [360]) interacting with these systems suggests to

[11] `https://twitter.com/janellecshane/status/892155008216473600`
[12] `https://twitter.com/JanelleCShane/status/1010324383921868800`

her readers that they, too, can interact with these systems. She provides a model for such interaction: finding edge cases, incongruities, feeding them unexpected inputs and picking apart unexpected outputs. Shane can't perform this service for every AI system, now and in the future, but her work shows what an influence this sort of interaction can have on our relationship to AI, both as designers of interactive systems and as an audience.

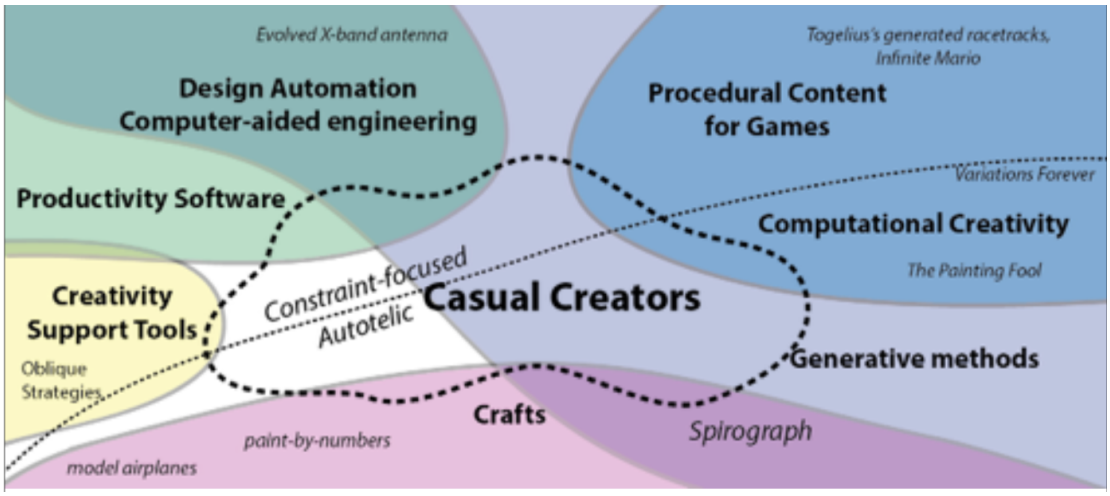


Figure 2.3: "Casual creators" in relation to other fields of study

2.5 Related fields

Do we need a new field of "Casual Creators"? What neighboring fields/concepts exist already? In this section, I briefly describe neighboring fields of study. I identify how they are similar to, and inform, my research about casual creators, but also how they have differ, often having different tools, focuses, or concerns from it.

Here are a few of the fields that we will be drawing from in this dissertation:

- Creativity (4C/"Big-C little-c"/P-H Creativity, environmental creativity)

- Computational Creativity
- Creativity Support Tools
- User Experience/ Interaction Design (Direct Manipulation, Flow, Reflection-in-Action)
- Game Studies
- PCG/Generativity/"Magic Crayons"
- Generative and Algorithmic Art (digital and non-digital)
- Evolutionary Algorithms, A-Life, and agent-based algorithms
- Machine-learning for creativity
- Fab-at-home, maker movements
- Computer aided design, generative design
- Explorable Explanations, museums
- Commercial apps, interactive websites

The related fields come from both computational and non-computational areas of work. In many cases, one area of study is the extension of another into a digital space, such as Direct Manipulation's application of Reflection-in-Action to user interfaces, or **Computational Creativity**'s exploration of concepts of **Creativity** through the lens of digital algorithms. Digital tools can be designed to support the user's ability to fulfill a creative task (**User Experience**), or to enhance their creativity (**Creativity Support Tools**) while doing so, or to enable a computer to accomplish something analogous to the human's activity (**Computational Creativity**).

We are also interested in the *autotelic* experience, rather than task focused, so we will expect to find guidance in how **games** let a player creatively explore a possibility space (such as crafting systems [173]). But games also have *generative content*, some of which is non-interactive, and some of which *is*. Chaim Gingold's "Magic Crayons" masters thesis defines an interactive system of "magic crayons" to be accessible, sketchable, computational, and expressive [157]. The "Magic Crayons" idea makes *use* of generativity, to bring content to life and create surprise and delight in the user. But it does not provide guidance for how to accomplish this technically, or what influence different technical choices would have on the user's delight.

What is generativity, and how does it work? There are many fields interested in this question. Though my own work grew out of my time making procedurally-generated content (PCG) for *Spore*, many diverse fields (with no contact with the games industry) have been working on the question in parallel, and arriving at interestingly divergent conclusions based on their backgrounds. **A-Life**, **evolutionary algorithms**, and other biologically-inspired approaches are committed to modeling interesting biological behaviors, with the goal for making art, but their interest is in the process, less in the product of the systems, and of course, they have little interest in bringing a human creative into the loop. Likewise, **Machine Learning**'s promise is to create something "artificially intelligent" that can create art without human intervention. The history of **Generative** and **Algorithmic art** can show us how these non-interactive, emergent systems can be enjoyed and appreciated by humans, and participate in the larger environments of human creativity (galleries, commissions, art movements, and more).

Other fields have a more *practical* set of concerns for their practice. **Fab-at-home** or **personal fabrication** tools, used by casual and professional makers,

have enabled new ways for humans to make art with algorithms, yet we still see echoes of established patterns of interaction for algorithmic art-making throughout history, from pottery, weaving, and lathe-spinning to Italian millefiore glassworking and Japanese yosegi marquetry. Custom fabrication has opened the door for computational assistance for design, as computers can help humans design for new forms of manufacturing, as well as one-off or custom works. **Computer-aided design** can help parametric architects design a 3D printable skyscraper, but can also power a website where a casual user can design a 3D printable necklace.

Users come to these systems from a range of different perspectives. Do users experience these new techniques as consumers, an audience (or gallery patrons), game players seeking challenge or advantage, artists looking to express their creativity, scientists seeking answers about a phenomena, or learners seeking knowledge of a system? If these systems help us deeply learn a system, then do we care about the system that they teach us? Some systems use creativity as a lure to encourage interaction with a system, but the system is one which the user has some extrinsic motivation to learn, like hard-to-understand phenomena of psychology, geology, or music. Several designers of systems like these have rallied around the term, "**Explorable Explanations**". These works take some complex system (like Fourier transforms or voting systems) and create an interactive activity in which the user *explores* the phenomena, iteratively deepening their understanding of the topic, while scaffolded by the guidance of the system. Some of these Explorable Explanations[13] are installed in science museums, but can also be found in commercial or governmental education apps, or hosted online by those interested (for their own reasons) in guiding the user to a deeper understanding of some complex system.

[13] `https://explorabl.es/`

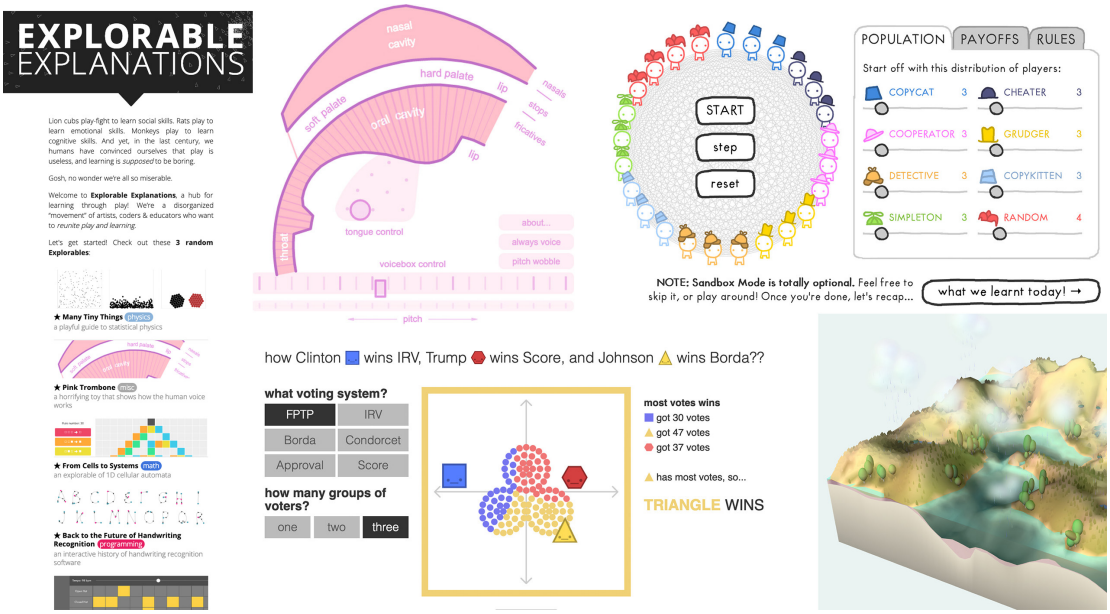


Figure 2.4: Samples explorables from the *Explorable Explanations* site (left): *The Evolution of Trust*[s70], *To Build a Better Ballot*[s24], *Pink Trombone*[s172], and *Earth Primer*[s59]

Finally, as we will learn from **creativity** studies in chapter 4, creativity does not happen alone. We create within an environment of creativity, whether we are making *Minecraft*[s136] castles to fight off attackers, *Scratch*[s195] games to share with classmates and teachers, clever designs for a Maker Faire or Ravely audience, 3D printable objects for commercial manufacture, an experiment for a conference paper, or works for an international art show. What "value" do we create? Who decides that value, who curates, or signal-boosts or critiques value in our community? Who have we learned from, and how do we contribute back to the community?

Casual creators exist.

But so too do many fields of studying how creativity works, how to enhance it with computation, and what to do with the output. This dissertation pulls ideas from many different fields. Even though they differ in their goals they share tools

with casual creators, or for other fields, vice versa.



Figure 2.5: A preview for an abstract music-making game Sentris identifies a common creative problem among casual creative users, and advertises a "fix" to attract them. If this kind of software could identify itself as a "casual creator", would give us a shorthand for this headline, while also linking it to others of its kind? (`https://www.polygon.com/2014/8/4/5968415/if-youve-ever-been-too-scared-to-make-music-try-sentris`)

Chapter 3

Central Themes

A reader of this dissertation will notice two recurring themes that pop up in many chapters. These themes are **tradeoff between power and control** and **maps and spaces**.

3.1 Power vs. Control

This was one of the first things I noticed when looking at this topic, and the one that first motivated the creation of the Casual Creator idea. A piece of software for a professional user will *always* privilege control. In the mind of the software designer, the ideal user for the software is either:

- a professional creative, performing work to conform to an external specification, with some demanding and particular customer
- a virtuosic creative, the DaVinci or Beethoven of the software, with a grand and history-changing vision which they must be able to execute to the full extent of their creativity

If you like, you can blend these two customers into one by imagining the virtuoso's self as the demanding customer. Regardless, the user **must** be able to exercise full control over the system. Whether the boss wants the logo to be in a brighter shade of red and 23% larger, or the composer wants to be able to change the key and inflection of the music within a single bar, the user's control must not be hampered by avoidable restrictions in the software (even unavoidable restrictions must be considered a loss to creativity). As we will see in the chapter on creativity chapter 4, these two archetypes of creativity are *not* the only ones available to us, but *are* the ones with which creativity study has traditionally concerned itself.

If we consider a third archetype, one who is creating for the pleasure of creating (autotelic creativity, or creativity "for its own sake"), some of the constraints suddenly fall away. If the user is not concerned with the particular outcome of the artifact they are creating, then they do not require the full control of the professional or virtuosic creative. However, they *are* concerned with their psychological and emotional state while being creative: if they are doing this for the pleasure and not the artifact, then the experience must feel good! We can expect the virtuoso and the professional to struggle through a painful-to-use tool (the virtuoso for art, the professional because they are being paid), but for the autotelic user, a loss of pleasure in the experience is a loss of the purpose of the exercise, and they have little reason to continue engaging with the tool.

On the other hand, if the system generates something of surprise, or power, or novelty for this user, it is unlikely to generate what they might have had in mind. But unlike our virtuoso or professional, they can often accept and enjoy the surprise. If they had a grand vision, they can be surprised into trading it for the system's accidental new direction. This trade works best if the system can provide

something that the user could not, or would rather not. This might be randomness or incongruity (socially-trained humans have a rather difficult time supplying either). It might also be effort or diligence: a system laboriously and patiently extrapolating one line into a thousand, or straightening and cleaning a thousand crooked lines, for example. Or it could be any of the other things which systems can do better, faster, or with less boredom than a human (spell-checking, lookup tables, geometric calculations, massively data-intensive statistical inference, basic arithmetic). If one or more of these is true of the computer's contribution, it has offered *power* to the user.

Of course, few casual creators have *no* interest in the eventual artifact they are making, but at the very least, the outcome is flexible in some ways, and the experience is valued on par with the artifact outcome. Some casual creator systems offer the user no more control than a "reroll" button, while others blend generative power with more traditionally user-controlled tools. Like most properties, control-vs-power is a spectrum, but it gives casual creators ample room to play within the broad range available.

One other outcome of this tradeoff is the restriction of possibility spaces. The possibility space of a piece of professional-grade software (think Maya or Garageband or Netbeans) must be able to support nearly *anything* the user wants to do in that space, to support the aforementioned need for control. Maya must be able to create: a virtual house for an architect's prospective clients, a special-effects-laden abstract animation, a 3D printable teapot, or a fully-rigged and textured game character. It also needs to support any artist who starts making a teapot, and realizes later they need to make it an animated character with flame effects. Maya, as creativity software, can make few assumptions about its user's needs.

However, if we have an autotelic user who is willing to trade power for control, the system can greatly restrict what it makes (3D printable teapots), and use the assumptions that it can make within that space (watertight mesh, thick sides and bottom, has a spout, handle can fit a hand, needs a top hole and fitted lid) to give additional power to the user. Our hypothetical teapot maker could allow the user to specify the volume and material, and adjust the body and thickness automatically without making the user resculpt the form. The user could make a larger handle, or a smaller spout, but the software could use its own calculations to limit the sizes available. "I want to put a hole in the side", the user could think. "That is not a part of this system", the software could reply, metaphorically, through its lack of UI for arbitrary hole-making. "How about a flower-pattern painter instead", it could say, again, metaphorically, through the presence of a "select paint pattern" menu. "Ok," responds the user, distracted by a nice rose pattern. The system can use its exciting generative power to redirect the users away from where they don't have control, and the experience still feels powerful and not unbearably constrained.

General purpose casual creators are rare, and if they exist, are usually quite shallow. More commonly one will find very limited-purpose creators. A quick search (on 7/23/18) of the Google Play app store for the word "create" finds creators for:

- 3D printable pottery (*Let's Create! Pottery*)
- unicorns (*Create Your Own Unicorn*)
- Voronoi-generated geometric art (*PolyGen - Create Polygon Art*)
- emoji (*Emojily - Create Your Emoji*)
- interior home layouts (*Planner 5D - Home & Interior Design Creator*)

- interactive pixel simulations (*The Sandbox Evolution - Craft a 2D Pixel Universe!*)
- graphic design banners (*Canva: Poster, banner, card maker & graphic design*)
- music (*Music Maker JAM*)
- multicelled organisms (*Cell Lab: Evolution Sandbox*)
- 2D physics simulated cars (*Create Your Car*)

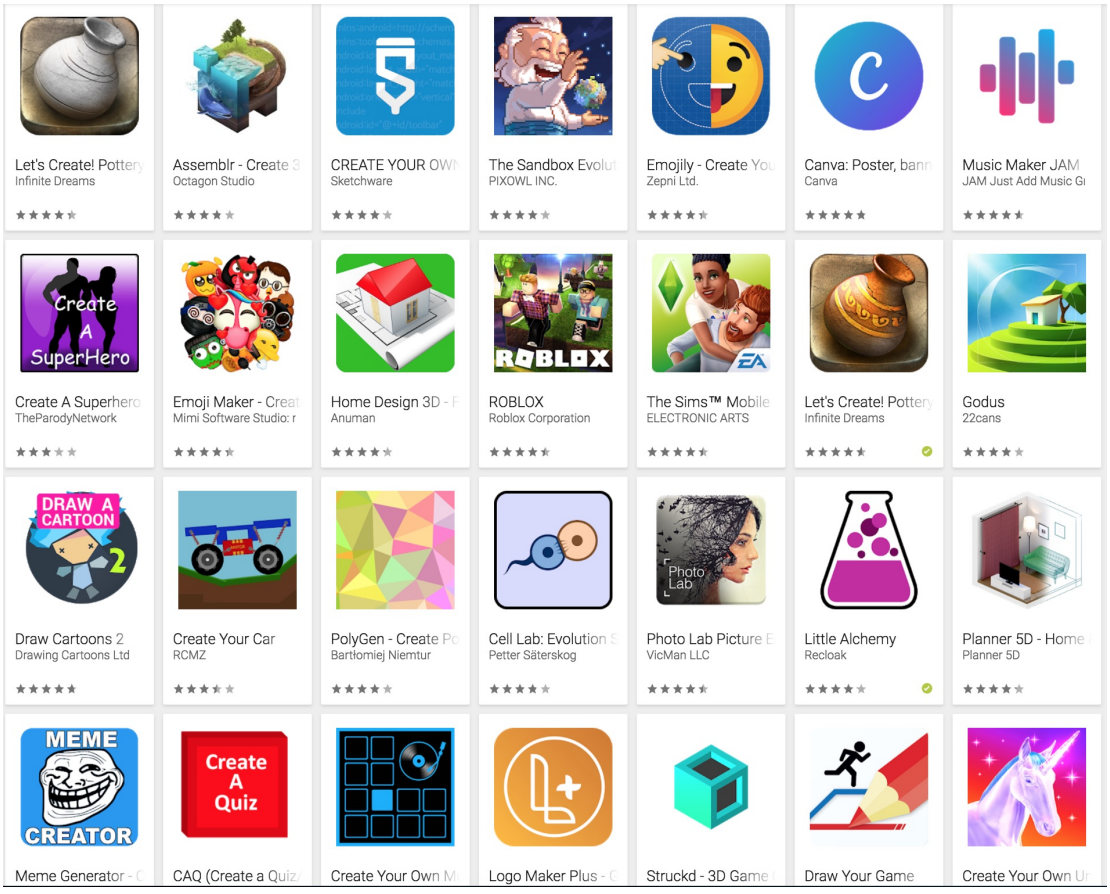


Figure 3.1: "Creation" apps on the app store.

Only a few apps seem as general purpose as a system like *Maya*[s132], such as *Assemblr - Create 3D Models (Sandbox AR)*[s8]. But even this app is a casual

creator: one can make 3D models, yet its targeted at a casual audience, and its models are made from stacking and recoloring geometric primitives. One game maker, *Draw Your Game*[s54], can make any game, as long as it is only built out of a few kinds of restricted 2D collision logic and movement, and another which creates 3D games (*Struckd - 3D Game Creator*) out of simple objects with built-in 3D collision logic. There is even a creator for making apps, *CREATE YOUR OWN APPS*[s39], which advertises easy-to-make apps with "Block programming similar to Scratch from MIT".

Many of these apps advertise ease-of-use, variety and power (within a limited space), and several explicitly mention creative community, sharing or remixing:

>"PolyGen is the app for "low poly" art. It lets you generate crystal patterns automatically or manually by hand. Sometimes, apps don't give you many options to play with, thus limiting your creativity. Because of this, we're getting accustomed to picking predefined wallpapers or using the same old filters. If only there was a tool that would let you create pretty designs by yourself effortlessly... Enter PolyGen, the app that lets everybody be an artist."[s1771]

>"We make music creation easy! Pick from thousands of studio-quality loops, beats and samples to start mixing your new track – then share instantly with a worldwide audience." [s1442]

>"With Let's Create: Pottery making ceramics has never been more simple and fun! Become a true artist and create "one of a kind" pottery items and share them with your friends! Throw the clay on the wheel, release all your artistic talents and take benefit of dozens of materials available in order to create your own design! Even when you glaze and fire your very first pot you will feel accomplished and relaxed as pottery is the best way to relieve your everyday stress

[1] `https://play.google.com/store/apps/details?id=com.bitbotany.polygen&hl=en`
[2] `https://play.google.com/store/apps/details?id=com.magix.android.mmjam`

and find your inner peace. An amazing, therapeutic and uplifting experience you can enjoy with your family and friends!" [s1233]

> "Canva makes design amazingly simple [Emoji:Raising Hands] (and loads of fun). Create beautiful designs for work, school and play in minutes with Canva – no design skills or complex software needed." [s294]

This is only a small selection of creators from a single Google Play Store search, but even without a defined community of practice (like the community around casual games), these apps have all developed a very similar language of casual, welcoming, and constrained-yet-powerful creativity.

3.2 Maps and Spaces

The other recurring theme that I will highlight in this introduction is the concept of maps and spaces. This theme is much more wonky and technical than control-vs-power (it was never mentioned, and barely alluded to, in any of the app summaries above), yet it underlies much of the power and usability of casual creators. In the chapter chapter 6, I explore the concept of what generativity is by defining is a property of *generative pipelines* built from connected *generative methods* that recombine and transform data structures. All of the above examples employ some generative methods, and define a generative space (even the banner maker and the unicorn color-changer). At the beginning of this research, I noticed the difference between how the user understood the space (their mental map) and creations the space is capable of representing. This dichotomy lead to many interesting properties of casual creators, such as their ability to suddenly "unfold" the

[3] `https://play.google.com/store/apps/details?id=pl.idreams.pottery`

[4] `https://play.google.com/store/apps/details?id=com.canva.editor&rdid=com.canva.editor`

perceived possibility space for users who learn a new technique. Even designers of generative system don't have a "true" map of the space, a property which necessitates tools for the designers (like *Danesh* [105] and Expressive Range[383]). This property also creates moments of surprise (for designers and users) when users discover and map unexpected new spaces, as when Spore users discovered glitches to create body-less and asymmetric creatures. No-one, neither designers nor users, has a finished, accurate map for any possibility space (of more than trivial complexity) and yet, making useful and improved mental maps of a generative space is one of the great pleasures of exploring a casual creator.

Only later in the research did I discover that it is also useful to treat the "actual" algorithmic possibility space of the generator as *two* distinct spaces: a *model space*, and an *artifact space*. This is a subtle distinction: we don't have a real-world equivalent difference for, say, a pot on a pottery wheel. When we modify the pot, we touch the *actual* artifact. **But in a generative space, we touch the artifact only through the generator that generates it.**

In relatively direct generators, such *Microsoft Paint*'s[s141] which models a pixel image directly as an array of the pixels that appear on screen, it feels as if by dragging the mouse, we are changing the screen's image directly. Yet what we are doing is *changing the computer's model*. Somewhere in memory, there is a 2D array of numerical values, which is interpreted by MSPaint into the pixels on screen. No matter how we "touch" the image, we are constrained by that possibility space. We can't change a numeric value to a value outside its range, or to a word, or to somehow represent a circular or even a transparent pixel. The model space of MSPaint (bitmaps) defines the artifact space (pixelated pictures). The artifact space has its own interesting properties, because it is the artifacts that interact with the human/physical/social world. We can losslessly represent

the possibility space of a 64x64 pixel image as a 32768 bit number (if each color is 8 bits), or as a continuous 12288-dimensional cube (three-dimensions representing a floating-point RGB value of each pixel, and the remaining 4096 dimensions representing which pixels are which colors) in which each "image" is modeled as a point in the cube. These numeric representations don't *mean* anything; there is nothing to say about whether or not one point is "similar" to another, other than to compare the vector distance between them. But if one image is of a yellow dog on a blue background, and another is a brown dog on a red background, those are more similar in *artifact space* than in *model space*. A picture of random noise, or a fish in space, or the word "dog" in yellow on a blue background would have very different relations to either image in artifact space, regardless of their positions in model space. Readers who use genetic algorithms (GA) will recognize the phenotype/genotype division analogy. To perform evolution, a GA system must evaluate the phenotype, but it can only ever modify the genotype.[5]

None but the most mathematically-minded user would think to make a mental map of a model space rather than an artifact space, but using a system fluently requires them to map the translation. Consider a user thinking: "I can draw lots of dogs in this program, but to make this look more like a terrier, I need to change the eye pixel to black". They are planning and mapping in the artifact space, but they need to move backward through model space to be able to encode it in the system. For systems with a less-direct model-to-artifact translation, this task becomes even more important to recognize. For a system in which the user specifies only a random seed (or can only press "reroll" to generate a new seed), the model is a single number or string, and the artifact is whatever (game level,

[5]If you are more comfortable using this more biological language, feel free to mentally substitute "phenotype/genotype" for "artifact/model" when reading. From teaching this topic, I find many users intimidated or confused by the scientistic language, so I have avoided it except when talking specifically about GAs.

song, or tweet) the generator translates that into. There is little useful to be said about that model space! On the other hand, in a generator which extrudes a metaball surface along a 3D graph (the Spore Creature Creator), the model is the 3D graph, and the artifact is the lumpy creature body that is generated through the Marching Cubes algorithm. There is a relationship between them, and though it is mathematically complex, it feels intuitive for the user to do the translation. The user can create a complex mental map of the real perceptual artifact space, according to any arbitrary aesthetic, functional, or social structure ("things that look like dogs", "things that look like birds", "things that are Pokemon", "in-jokes for my friends", "good things", "bad things", "things that won't work in the game"), and translate it, sometimes, into modifications in the mathematical, computer-interpretable model space.

By separately considering these three spaces (model, artifact, and mental map), we can discover all manner of interesting phenomena when translations (and mistranslations) are performed between them, and we can identify the patterns that improve the accuracy and speed (or fun) of those translations.

Chapter 4

Creativity

> "I like to just make things... If I have the TV on, I'm not just going to sit there. I want to do something with my hands; I've always got a project. Or I get a small group of people over, and I say, 'Let's make these things today." - Amy Sedaris

The field of creativity is immense, and well-studied through the perspectives of neuroscience, sociology, psychology, history, and philosophy. This chapter will provide an overview of several disciplinary perspectives on creativity as well as several frameworks on creative experience that I have found useful for this dissertation's goals.

What does it mean to be creative? We can consider the great creative geniuses that spring to mind at the term "creativity": Prokofiev, Tolkien, Tesla, Kahlo, Balanchine. Perhaps also entrepreneurs, entertainers, investors, or even corporations [120]. We have invented many ways to define creativity, each drawing a line around "who is creative", and by extension, who is not.

When Csikzhentmihalyi writes on creativity[111], he prefaces it by defining a few popular ways people define creativity (social creativity and elan in conversation, creativity of personal experience, and creativity of culture-changing production), before announcing that he will only address the third:

"All three kinds of creativity enrich life by making it more interesting and fulfilling. But in this context I focus primarily on the third use of the term, and explore what is involved in the kind of creativity that leaves a trace in the cultural matrix" [111]

This attitude reflects a common thread in studies of creativity, the discarding of the everyday or non-productive creativity in favor of one which, to students of industry or history, **matters more.** This dichotomy is addressed further in this chapter. The valuation of industrial and cultural levels of creativity has leaked into software development as well: while there are studies on software support for industrial creativity (see the next section on Creativity Support Tools), there are few or none on tools to support casual creative. When telling others in my field about this dissertation project, I have received support, but also moments of disappointing pushback about why I'm wasting users time when I could be making *real* tools.

But we also speak of children as being creative, and of regular people being creative, often without recognition or payment. People gather and celebrate each other's creativity on DeviantArt, Etsy, Archive of Our Own, and Pinterest. Craft stores, art stores, and woodworking stores are all big businesses. Creativity is accessible and important to all people. It provides entertainment, but also enough emotional and psychological value that it can even be used as therapy, and enough social value that it can be used as a basis for group bonding. It seems that humans enjoy being creative, and perhaps even that humans need to be creative.

The answer of "why am I studying creativity" that creativity is known to be good and valuable, for purposes both personal and social (as well as industrial and cultural). One goal of this dissertation is to rehabilitate and celebrate the role of *casual creativity* in software.



Figure 4.1: A collage of creativity posters from Pinterest

4.1 Creativity inside and outside of production

> "I think there's an artist hidden at the bottom of every single one of us." Bob Ross

What is creativity? The most widely accepted definition in both computer science and psychology seems to converge on a declaration that creativity is that which creates ideas that are novel and appropriate (or useful) [394]. After concurring on that point, agreement diverges on what conditions cause creativity, and how novelty and usefulness are defined.

Whether an idea is novel or useful is not a clearcut division: if Leibnitz or Newton discovered the calculus first, was one less creative? Or if a musician invents a melody, 200 years after it was first hummed and lost, is that still creativity? If an artist paints a landscape, but the art world never takes note, is

that creativity? There are no shortage of edge cases, so these questions provide reason to split the common-knowledge concept of creativity into multiple species, like "personal" and "historical" creativity [34], or somewhat apocryphally "big C"-"little c" creativity[269].

Kaufman and Beghetto[216] further break it down into 4 total categories:

- Big-C: "eminent accomplishments" (Beethoven, DaVinci, et al)
- Little-c: "everyday innovation", unexceptional art made by unexceptional people
- Mini-c: "transformative learning", the creative insights people have about their lives, "the novel and personally meaningful interpretation of experiences, actions, and events"
- Pro-c: the fuzzy middle-class of professional creatives who are more creative than everyday people, but not significant on a historical timeline

Of the little-c's, they say:

> "If a friend shared a painting with you, he or she would not expect you to begin your critique by comparing it to Van Gogh or Monet. Rather, you would be expected to explain whether you found it to be unique and aesthetically pleasing (within the context of what you know about that particular style of painting and your friend's current level of artistic accomplishment)."

There is still a "useful and novel" evaluation, but it is done *relative* to the viewer and the artist's place on the creative ladder.

Each of these are variants on the idea that some creativity produces ideas that are locally-novel and useful only to the individual having the creative insight, and other ideas are universally-novel and useful globally, changing entire fields of practice on historical scales. The latter is the group with which Csikzhentmihalyi

is concerned in his book on creativity[111], but even then, some of the concepts are useful to "little c" creative practice(as we will see in the section on environmental creativity. Communities of practice are fractal: the patterns we see among the feuding French Impressionists may be seen as well in any local quilting group.

Only rarely do psychologists explicitly study "everyday" creativity *as it is experienced*, as Ruth Richards does:

> "One may ask: Would we humans value everyday creativity more – the "originality of everyday life" – if we knew how much it could do for us, for example, improve our physical and psychological health, boost our immune function, and give us greater life satisfaction and meaning?" [319]

In doing so, she still creates a revised "novel and useful" definition: "originality (or relative rarity of a creation within a given reference group) and [...] meaningfulness (being comprehensible to others, not random or idiosyncratic, and thus being socially meaningful)", but also acknowledges that "everyday creativity is not only about product but is also about process, about how one does a task." She lists many, many studies of benefits of creativity. But I am left wondering, would these studies see the same effects from activities that had low originality (ie, a coloring book vs unstructured drawing)? Are these benefits due to the process or the product?

Creativity-as-production has an economic and industrial impact, so perhaps it is unsurprising that studies of creativity often view it through a product-focused lens, and ask the question: how can we produce **more**, **better**? Even user-experience research often uses task-completion as a primary metric. There are fewer studies on how we *feel* when being creative. US-based military funding structures like DARPA are understandably disinterested in whether a battleship designer is having a nice time. We can see this distinction in the difference between usability, measured with: *"effectiveness; flexibility; ease of use and learnability;*

and the user's attitude"[106] and user experience, measured by *"fulfilling the human needs for autonomy, competency, stimulation (self- oriented), relatedness, and popularity (others-oriented)"*[184].

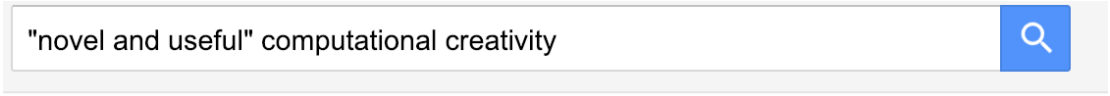
Hassenzah defines these two different forms of metrics as "do-goals" and "be-goals":

> "Pragmatic quality refers to the product's perceived ability to support the achievement of 'do-goals', such as 'making a telephone call', 'finding a book in an online-bookstore', 'setting up a webpage'... In contrast, hedonic quality refers to the product's perceived ability to support the achievement of 'be-goals', such as 'being competent', 'being related to others', 'being special"'

Casual creators are implicitly defined by a be-goal (to **feel** creative) even as the name suggests a do-goal (to have **created**). The heuristic of a casual creator is about the **autotelic** experience. Even when speaking about the artifacts of casual creators, we will speak of them in reference to their autotelic properties: "is this a good ending to the journey of my session" or "am I proud of this?" or "does this make me feel connected to others", instead of "is this good?" or "is this commercially/culturally valuable?".

This framing *rejects* the popular definition of creativity: "creativity as the production of novel and useful ideas in any domain."[12], a definition that has become *de rigueur* in computational creativity. Casual creators can be a part of "Pro-c" or even "big-C" creativity, as brainstorming or prototyping tools. While some casual creators may seek to make grand works of art *possible*, they must be primarily concerned with the "little-c" experience.[1]

[1]Daniel Newman, gained in-game fame as 'dananddna', one of our most exceptional *Spore* users. We met when he visted the Maxis offices as a reward for his creations and continued correspondence afterward. His abilities eventually outgrew *Spore*'s possibility space, and he switched to tools like *Maya*, becoming a professional illustrator. Casual creators can be a *part* of someone's creative growth, rather than an end point.



About 2,850 results (**0.17** sec)

22 Fifty Years of **Creativity** Research
RE Mayer - Handbook of **creativity**, 1999 - books.google.com
… The underlying view is of **creativity** as mental **computation**: **Creativity** can best be repre- sented … On balance, **computational** method- ologies seem to offer a unique source of evidence that … the classic definition of **creativity** as the con- struction of **novel and useful** products needs …
☆ Cited by 860 Related articles All 3 versions

[PDF] **Computational** approaches to storytelling and **creativity**
P Gervás - AI Magazine, 2009 - aaai.org
… over the recent years that address the study of **creativity** from a **computational** point of … Jennings (2008) introduced **computationally** plausible modeling of the fact that most human **creativity** takes … is addressed), and maps of physical space are kept and used to **compute** the story …
☆ Cited by 125 Related articles All 14 versions

Supporting **creative** design in a visual evolutionary **computing** environment
H Liu, M Tang, JH Frazer - Advances in Engineering Software, 2004 - Elsevier
… It is not realistic to simulate **creativity** by **computational** tools, but it … generate many surprising and inspirational architectural forms, and how **novel and useful** structures could … Evolutionary **computation**, artificial intelligence, integrated and interactive system techniques, and virtual …
☆ Cited by 63 Related articles All 5 versions

Creativity syndrome: Integration, application, and innovation.
MD Mumford, SB Gustafson - Psychological bulletin, 1988 - psycnet.apa.org
… Although each creation reflects a recognized **novel and useful** solution, only those that meet the additional … Specifically, it is unlikely that **creativity**, in the sense defined above, represents a … Rather, **creative** behavior is likely to be determined by a complex interaction between the …
☆ Cited by 2003 Related articles All 8 versions

Affective **computational** priming and **creativity**
S Lewis, M Dontcheva, E Gerber - … on Human Factors in **Computing** …, 2011 - dl.acm.org
… This study is the first to empirically demonstrate how **computationally** induced affect … to **creativity** support tool research, demonstrating the effectiveness of affective **computational** priming, and … BACKGROUND **Creativity** is defined as the generation of **novel and useful** ideas [2 …
☆ Cited by 64 Related articles All 10 versions

[PDF] Mixed-initiative co-**creativity**.
GN Yannakakis, A Liapis, C Alexopoulos - FDG, 2014 - autogamedesign …
… **useful**) is ultimately a result of iterative co-creation. The autonomous **creative** system, in that case, finds novel ways to navigate a search space, by eg looking at orthogonal aspects of the human **creative** process, which are suggested back to the human. **Computational creativity** …
☆ Cited by 88 Related articles All 13 versions

Figure 4.2: Maybe we need a novel and useful definition of creativity.

4.2 Environmental creativity

"Where is creativity?" asks Csikzhentmihalyi in his exploration of great creative individuals[111]. In our collective imagination, we often picture tortured geniuses operating alone in an attic somewhere, isolated in their creative spaces, only emerging to push a finished artwork out the door. Yet this pervasive myth is only that, a myth, and artists in an attic tend to have a creative community more like the artists of La Bohême. Even in Van Gogh's seclusion in the French countryside, he maintained a lively correspondence with Paul Gauguin about the evolving art movements in Paris, even exchanging paintings.[149]

All creativity happens in an **environment**, whether that means the physical space around the artist, the expectations of their chosen domain, or the evaluations of their peers. Csikzhentmihalyi identifies this environment as

- the "domain" (a material/culture with rules),
- the "field" (experts in that domain who are both knowledgeable and listened to by others)
- the individual (our 'lone' genius)

Even when the individual is creating, they are pushed around (and push back on) the rules of their domain. They are aware of their field, and how their work will be received in it. Their work may go on to change the rules of domain or the opinions of their field, but they also work against the backdrop of all the past changes caused from previous artwork.

Fabien Accominotti[7] uses computational statistical methods to model the Impressionists (and their many offshoot movements) as a network that not only encompasses the individuals careers and works, but their ever shifting allegiances

to new movements. By doing so, he can see that, at least in this one art scene, we can see the lone-genius myth busted as small groups of multiple artists synchronously reach their career peak.

> "Over time, organizing into movements increasingly became a strategy for artists eager to supplant older, more established ones. This strategy spread after the success of the impressionists, once it had become clear that at least some contemporaries regarded movements as the place where significant contemporary art was being produced. As an organizational form, movements thus became appealing to young, ambitious artists. Such a mechanism, as I have shown, also has the advantage of accounting for the decreasing steadiness of creativity over the lifecourse in later generations of painters owing their peak to movements."

Table 1
Overall creativity careers and 5-year periods of peak creativity over historical time (1845-1984) for the 41 French artists in Galenson (2001)[a].

Movement (if possibly accounting for creativity peak)	Name	1845-1849	1850-1854	1855-1859	1860-1864	1865-1869	1870-1874	1875-1879	1880-1884	1885-1889	1890-1894	1895-1899	1900-1904	1905-1909	1910-1914	1915-1919	1920-1924	1925-1929	1930-1934	1935-1939	1940-1944	1945-1949	1950-1954	1955-1959	1960-1964	1965-1969	1970-1974	1975-1979	1980-1984
-	Bissière																							P					
-	Bonnard																		P										
-	Cassatt										P																		
-	Cézanne													P															
-	Chagall														P														
-	Gauguin										P																		
-	Herbin																			P	P	P							
-	Manet								P																				
-	Modigliani															P													
-	Redon												P																
-	Rouault																			P									
-	Rousseau H.											P			P														
-	Seurat									P							P	P											
-	Soutine																												
-	Toulouse-Lautrec										P																		
-	Van Gogh										P																		
Impressionism	Caillebotte							P																					
Impressionism	Degas							P																					
Impressionism	Guillaumin					P																							
Impressionism	Monet					P	P																						
Impressionism	Morisot						P																						
Impressionism	Pissarro						P																						
Impressionism	Renoir							P																					
Impressionism	Sisley						P	P																					
Impressionism	Whistler							P																					
Nabi Movement	Vuillard											P																	
Fauvism	Derain													P															
Fauvism	Dufy R.												P					P											
Fauvism	Matisse													P															
Fauvism	Vlaminck													P															
Cubism	Braque													P															
Cubism	Delaunay R.														P														
Cubism	Gris														P														
Cubism	Léger															P													
Cubism	Picasso													P															
Dada	Arp															P													
Dada	Duchamp M.														P														
Dada	Picabia															P													
Surrealism	Masson																	P											
Surrealism	Miró																	P											
Surrealism	Tanguy																	P											

Figure 4.3: Accominotti's chart of the painters, their active years, their peaks (by artwork price) and their movement, demonstrating the remarkably tight clustering of individual behavior in each "cohort"

We make art relative to the rules of our domain: both intrinsic rules like how paint can reflect light, or extrinsic rules like what subjects can be painted. We also have a community, whether our friends that we share personal projects with, or the world of professional art critics, and we create art in relationship to them.

Systems (including casual creators) that adopt this framework usually use human social collaboration, review and external judgment as part of the creativity process. A reified example of social creativity judgment can be found in games like *Apples to Apples*[s6] or *Cards Against Humanity*[s25], in which players attempt to construct the most clever juxtaposition of cards, to be judged by other players. No card combination, no matter how creative, 'wins' unless judged to do so by the current expert.

One computational creativity project which explored Csikszentmihalyi's domain-field-individual creativity model is Rob Saunder's *Digital Clockwork Muse*[s43][343]. Saunders created a community of digital artists that could produce virtual "paintings" and share them with each other. Each one had their own style, and they could even self-organize based on their style and what they liked. But what they like was *relative* to the work in their domain. Saunders modeled their preferences with a function called the Hedonic Curve, which says "I like things that are different from what I've seen before but **not too different**."[2] Like Csikzentmihalyi, Saunders' system encoded that creativity is not an abstract quantity that can exist outside a society, but a property of the relationships between artists, their opinions, and their past work:

> "The aim was to show that agents are not recognised as creative when they fail to innovate inappropriately. Agents can innovate inappropriately either by producing "boring" images that are too similar to images previously experienced by other agents, or by producing "radical" images that are too different for other agents to appreciate." [344]

[2]More about this experiment can be found in the chapter on Entertaining Evaluations.

4.2.1 A digression on the "Wundt curve"

Saunders' Hedonic curve is a wonderful and useful illustration of a "common sense" insight. He cites Berlyne's 1970 study[31] on the hedonic value of complex/simple and novel/familiar abstract images. This study demonstrated that either complex-familiar or simple-novel images maximize hedonic value, while simple-familiar images are boring, and complex-novel images are threatening. Berlyne cites Wundt, so I went on a digression to find out what, exactly it was that Wundt said about the curve, and was surprised to find that the reference was at best, metaphorical, as Wundt was studying the loss of nerve response to continued stimulus. I also found the first use of "hedonic curve" in reference to the curve, in "Human Psychology", by Howard Crosby Warren, 1919.

We come closer to the movement in nervous substance itself when we stimulate the nerve at two different points of its course, the one remote from the muscle, the other as near to it as possible, and when the experiment is so arranged that the stimulation is timed to occur in both cases at the same point upon the axis of abscissas above which the curve of contraction is described. If the two stimuli have the same intensity, and the nerve is kept in as constant a condition as possible, the resulting curves evince a twofold difference. In the first place, as HELMHOLTZ discovered, the curve of contraction given by the more remote stimulus begins later—has



FIG. 25.

Figure 4.4: Wundt's curve, decay of nerve response in response to stimulus repetition over time[441].

4.3 Embodied creativity

Creativity happens within the relationships between an individual, a domain, and a field. Yet in addition to zooming *out* from the individual, we can also

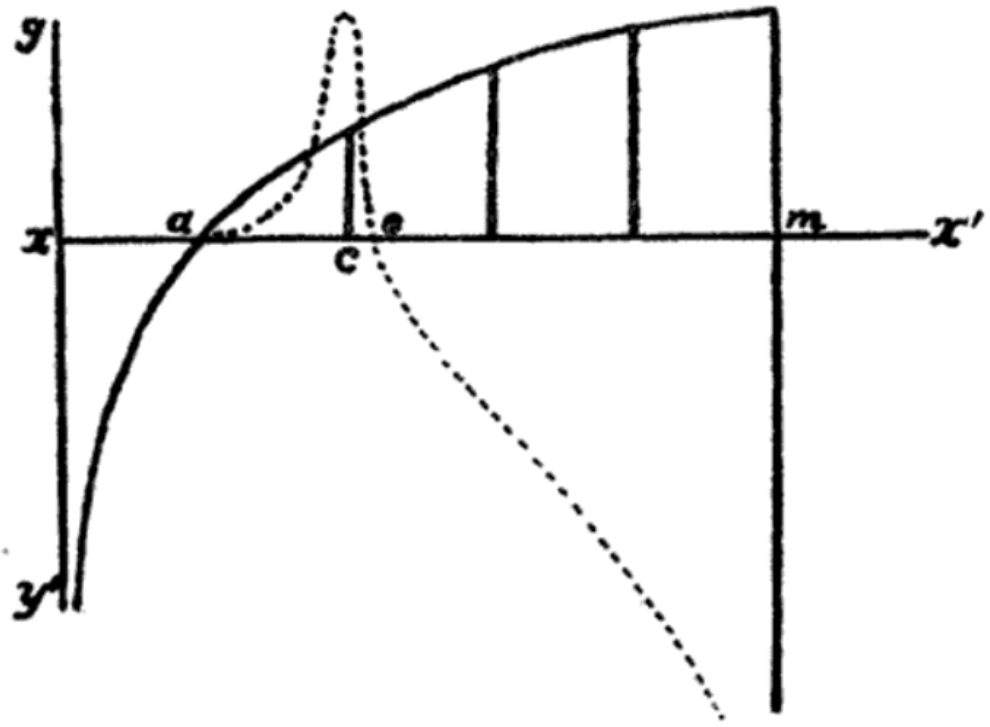


FIG. 68. — HEDONIC CURVE.

Dotted curve shows change in degree of pleasantness (above x axis) and unpleasantness (below x axis) with increase of stimulus (left to right along x axis). The corresponding changes in intensity of pressure sensation are shown by the unbroken curve. [From Pillsbury, after Wundt.]

Figure 4.5: Warren's curve, linking intensity of stimulus to pleasure, showing a sweet spot of pleasure between apathy and pain[431].

zoom *in* to see a different network of creativity. This individual contains a mind, in which they store a model of the domain and field (and their collected expectations), but they also are in a *physical space*, with *tools*, and a *physical body* connecting them to their tools and the space. They also have an *artwork*, which is separate from all these, including being separate from the mind.

Studies have attempted to show that one kind of office environment or another will spark creativity in a workplace[261][228]. We can again see the industrial

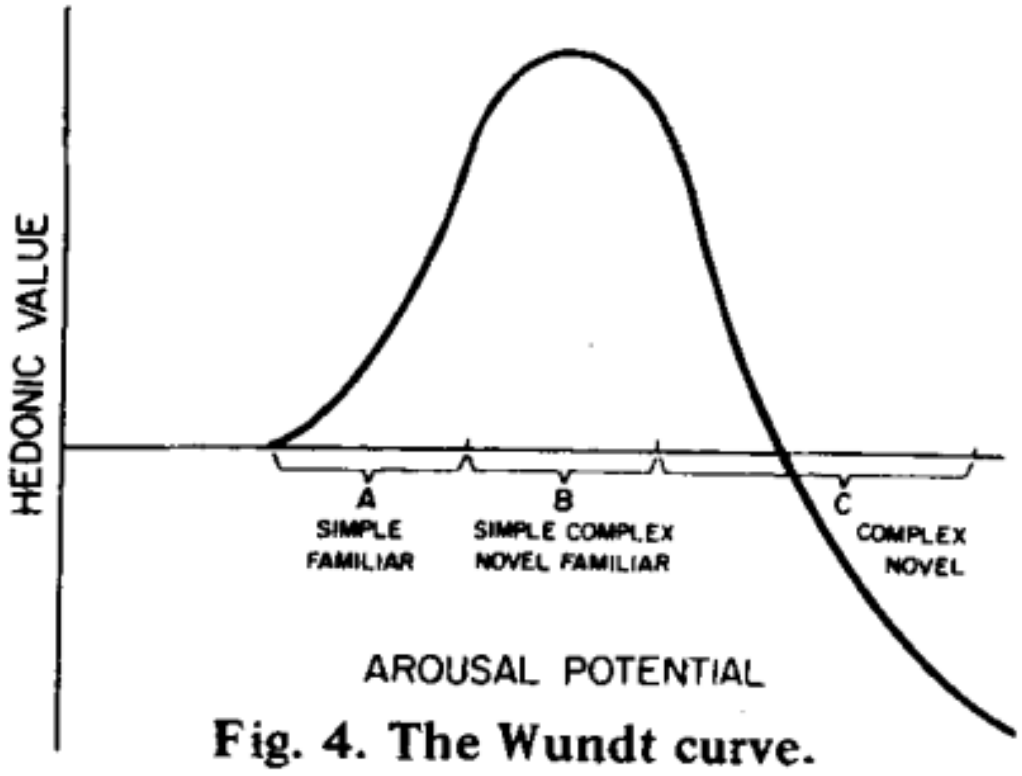


Figure 4.6: Berlyne's curve:applying Wundt's curve as a metaphor to describe response to aesthetic stimulus.

framing of creativity in these studies (they seek to extract more value from employees or students with inexpensive and reusable resources like beanbag chairs and pictures of the sea). But it is hard to deny that local environment, whether loud or quiet, social or isolating, stark or over-stimulating, can have an effect on creativity, even if it is not clear what it is, or whether is is consistent across different individuals, populations, or even moods.

To understand the effect of tool-use on cognition, it is very common for UI designers and user experience researchers to cite Heidegger's "hermeneutic phenomenology" and theories of tool use[128][152][211]. Heidegger's theories describe

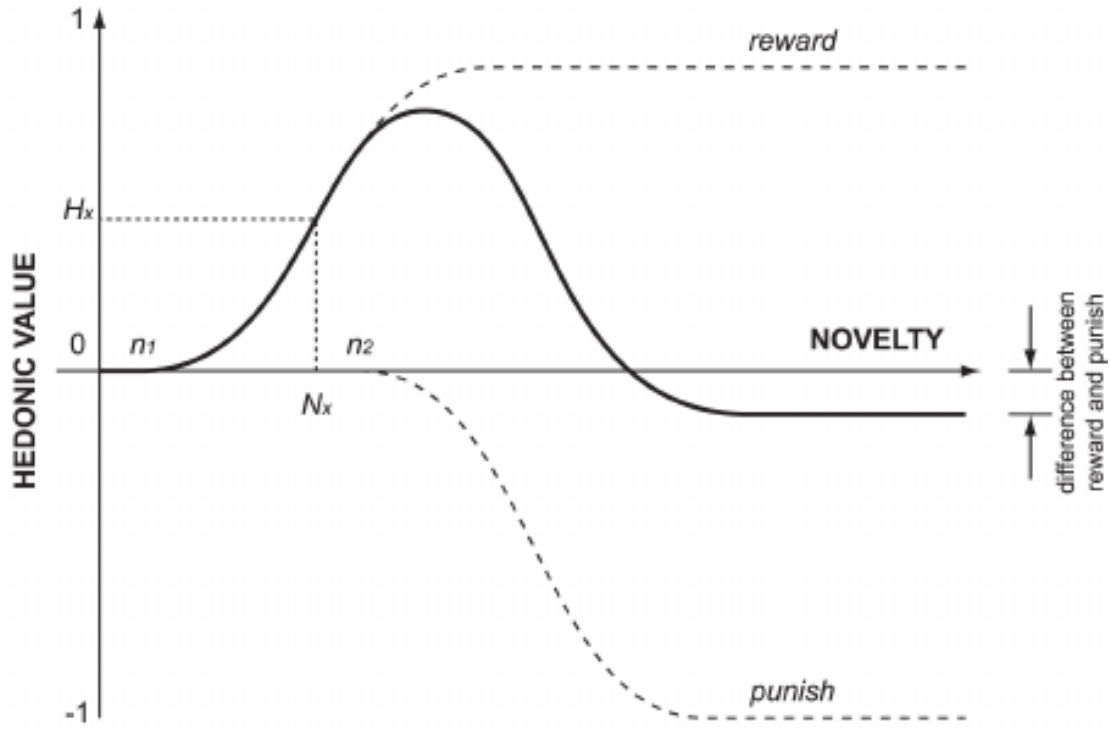


Figure 4.7: Saunders's curve:compressing Berlyne's axis to "familiarity", not complexity and familiarity[344].



Figure 4.8: Environments which students considered most conducive (left) and least (right) conducive to creativity. It is unclear if they controlled for the presence of seating[261].

how we relate to tools and how when we use them, they become invisible to us as our proprioceptive focus moves, from the tip of our fingers, to the tip of the

tool in a moment of "tool absorption"[220]. The theories also address the moment when, in their malfunction or our misjudgment of their properties, they become visible again. This principle relates to software tools just as well as it does to the physical tools Heigegger would have been writing about:

> "In daily life, the use of tools is second nature to us. We seamlessly conduct our goal-orientated activities via physical objects without the slightest awareness that we are doing so. So accustomed are we to the use of knife and fork, computer keyboard, can-opener and door-key, that the only times we become aware of their presence is when they malfunction and interrupt our activity." [211]

Importantly, these Heideggarian frameworks of interaction privilege **interaction** as a way of understanding the phenomena and answering the question "How does the world reveal itself to us through our encounters with it?"[128] We don't just observe the world, or our artwork, we understand it through repeated interactions with it, and we understand those interaction through the probes of our tools.

The paintbrush (or the casual creator software) is part of the artist, except when it also embodies the expectations of the field of painting, or is a probe to understand the flow properties of the paint. The painting is a physical object that the user can look at, but they also observe it from the perspective of their field of peers (or their model of them). It is also a *tool* with which the user understands *the painting they want to make*:

> "When an object has a complicated spatial structure, or is highly detailed, it is often easier to simulate outcomes by manipulating either the real physical thing, or an appropriately simplified physical model of the thing than to simulate manipulating that thing through mental imagery"[220]

This thinking-with-tools paradigm is referred to as "embodied cognition"[203], and lets us inspect the relationship between the individual and their tools-at-hand.

A piece of casual creator software often contains a visible environment, tools, the artifact, and ways of knowing the artifact. Our interactions with the artifact in a casual creator may be profoundly strange: in this dissertation we will see modifications of "living" creatures, impossible paintings drawn in the air, and 4 dimensional explorations. While these tools are stranger and less familiar than a paintbrush or screwdriver, embedded cognition thinkers are already attempting to find the limits of Heidegger's tool absorption:

> "How different can our remote "body parts" be from our own before we cannot assimilate them? [..] But what about twelve-fingered claws, and what about having the fingers work in continuous fashion? Probably not impossible; but clearly an interface challenge. And then there is the question of how different a scene in a virtual world can be before it shatters our situated grasp of things? Can we cope with a world that runs at clock speeds fifty times our own?"[220]

Fortunately, in this dissertation I find few limits on the *range* of what strange tools can be absorbed, but do find limits on the properties that they must have in order for this to happen (see chapter 8).

Interestingly, enabling new forms of embodied action, through diverse tools, does seem to trigger different kinds of cognition. Experimental subjects who were encouraged to make "on-the-other-side" gestures with their hands generated more ideas than those who were prevented from gesturing[236]. We can all remember times when we thought differently once we put our ideas onto post-it notes, whiteboards or index cards that we could reposition on a table. This suggests that tool-assisted creativity will be *different* from non-tool assisted creativity.

Bringing environmental and embodied creativity together, it is possible to consider the domain and field of the artwork as necessary tools to understand the artwork itself:

> "The work of art cannot be interpreted for what 'it' knows. Rather, the listener or viewer may be compelled to engage in tacit processes of

> contemplation, reflection and consequent knowledge production, interwoven with context, aesthetic materials and participating individuals. "[398]

Again we see the implication that knowledge of the artwork is a process of continuous action, not an insight. However, it does not need to be a *burden* that we need a community to understand and process (and create) art. Like any casual creator tool, the joy is in using the tool, and there is pleasure, value, and growth in using a community as tool to make art.

4.4 Social values, economics, and creativity

It would be easy to write this dissertation looking exclusively at pieces of software and the creativity that happens within them. But one fascinating property of creativity, especially of casual, "everyday", little-C creativity is that everyone possesses it and can experience it, and that means that the communities of creativity can grow much larger than Accominotti's handful of Impressionists. It can also encompass all of a community, even those who are not skilled or particularly passionate about crafting. This property has been used to create activities that can unite broad groups of people around an activity that none of them are "dominant" at. Researchers and community activists have successfully used this to help diverse populations like refugees and locals find common ground with shared creative experiences outside of either the locals or the refugees culture, researchers write: "The ambiguous nature of events provides a contested domain of new community formation, integration and adaptation"[238]. The absence of experts can be an *advantage* in some kinds of social creativity.

While some forms of casual creativity are intensely private and personal, even invisible to outsiders, casual creativity is often situated in a social context. The

most common form of casual creativity is the idea of "crafting", a sometimes pejorative, consistently controversial, and continuously redefined term applied to autotelic creative labor by non-professionals[3].

In **Hobbies**[154], Steven Gelber tracks the middle-class American/British trajectory of hobbies. He cites their beginning at the industrial revolution, where people en masse left home to work for the day, and thus work/home lives became considered as two different worlds, and "leisure" became a class of activity:

> "Most commentators assumed that free time spent at home was superior to that spent at public amusements, and prescriptive literature favored useful domestic occupations as appropriate leisure activity. Consequently the Victorian period became a golden age for household entertainment. Amateur music making was ubiquitous, and when the family tired of playing and singing, there were parlor games. The rise of hobbies was part of this upwelling of home-based leisure, and it worked to reintegrate the divided spheres of work and leisure. When paid labor left the house, it had taken its values with it. [...]The values that governed the workplace were, for the most part, left at the factory gate—but not entirely. One could avoid the temptations of idleness not only by working at work but also by working at home." [154]

Around this same time, Ruskin and his fellows were extolling the virtues of the handicraft as moral virtue, and urbanization and crime sparked a moral panic in the Victorians. What better solution to moral decay of the street life could there be than to engage in virtuous (indoor!) labor? These hobbies often took on other moral characters (manliness, motherhood, thrift), and continued as a leitmotif into every moral panic onwards (including the present-day version of repairing your little ones screen-damaged minds with wholesome craft kits).

Aside from the moralistic element, people gain satisfaction from their hobbies:

> "In fact, several studies suggest that people identify physical and abstract qualities in their leisure that are missing in their work. These include a sense of achievement, exercise, creativity, status, competition, and leadership"

[3]Unless used by Ruskin and his contemporaries, where it means the exact opposite.



Figure 4.9: A Ladies Home Journal page advertising knitting tutorials, watercolors, crafting tools, and instructions for the hot new trend of "crazy quilts". With updated graphics, this would be a familiar front page of Pinterest. From `https://babel.hathitrust.org/cgi/pt?id=mdp.39015012341544;view=1up;seq=20`

The idea that you can express yourself, in simple ways that matter to you (and aren't out of your reach) is a profound one, and we do well to remember that even activities that are not "creative" on the surface are often creative outlets:

> "The question of how to arrange stamps in an album was in fact a visible symbol of the meaning of the collection. "To put a lot of stamps into an album, beginning with the smallest value and following it with

> the next higher value may do very well for a miserly collection," explained one woman collector, "but to arrange a collection of stamps to be attractive, as well as valuable, takes the fertile mind of a woman." To women, the stamp collection, like the trade-card collection, was a medium through which they could express their creativity; it was not a compendium of economic value."

Casual creativity movements are perpetually interwoven with the tension between industrial productivity, "great man" hagiographies, and social casual creativity. Is creativity a way to make money, to become famous (and culturally significant), or to spend time? And who's time is being spent? Though this dissertation rarely discusses explicit gender issues, why have video games been the subject of hundreds of dissertations, but creativity apps that fill up the app store are so grossly unstudied that this dissertation attempts to categorize them for perhaps the first time? Maker Faire is a community founded to provide community for the casual inventors (tinkerers)[127]. But in the time I have been attending and presenting there, the mood has changed from what felt like a county fair to something more akin to CES, as it began focusing increasingly on the money-making industrial potential of the exhibitors[255]. It also suffers from an oversized gender imbalance, crafts-for-sale were exiled away from the main showroom to the "Craft Barn", exacerbating the issue, and only recently moved the CNC sewing groups into the main expo hall. But Maker Faire (and the Maker movement) is one of many casual creativity groups. Pinterest, DeviantArt, Ravelry, and Tumblr are all sites used to share creativity up and down the skill ladder. But even these site have grown out of long histories of creative communities, from professional guilds to quilting bees.

The book *Making is Connecting* [150] addresses these contradictions in an overview of the many social functions that craft plays in communities, through a lens of personal creativity, but also of social, economic, and political organization,

as "this book is about the idea that making and sharing is *already* a political act". Like this dissertation, this book is concerned with "everyday" creativity and what it provides outside of (and even in opposition to) industrial value. David Gauntlett proposes a new definition of creativity in contrast to the "novel and useful" default:

> "Everyday creativity refers to a process which brings together at least one active human mind, and the material or digital world, in the activity of making something which is novel in that context and is a process which evokes a feeling of joy."

He concludes with five principles that tie together the themes of the book:

- "A new understanding of creativity as process, emotion, and presence
- The drive to make and share
- Happiness through creativity and community
- A Middle layer of creativity as social glue
- Making your mark and making the world your own"

He then proposes a manifesto for what the future of creativity should hold:

- "People should be given opportunities to express their creativity through tools which do not seek to shape or determine the outcomes; and which enable people to express their unique presence in the work; and which mean that their contribution is distinct and recognizable"
- "People should be able to share the fruits of their creativity simply, and without unreasonable restrictions or gatekeepers"
- "Communication, exchange, and collaboration should be enabled and encouraged, to foster engaged communities of interested people who can help and encourage each other. Collaboration should be at the level which participants in a project are most comfortable with, which could be in tiny bits, or substantial chunks."[150]

These points are influenced by the book's history of the loss of freedom in Web 2.0 (as homemade blog posts and self-organized web-rings became subsumed

by corporate-owned content curation). Creativity tools are not empty pieces of software (or machines), but political choices that influence how people can be creative and share that creativity, and exist within a social and economic space that has negotiated (and imposed) limitations.

4.5 Situating casual creators (self and society)

> "Everything is connected to everything else. Fortunately, not all connectivity is equally dense. The non-uniformity of connectivity makes science possible."[203]

Creativity happens in communities, in fields and domains, in the minds of experts and novices, and in the interaction of an artist with their tools. A tool that ignores the greater context of this creative ecosystem does so to its loss. So it is to our advantage in this dissertation to consider the full range of these "sites of creativity", which are:

The creative community:

- Domain (properties of the medium, expectations of the creative form)
- Field (experts and audience, the "art institute")

The individual:

- Their experience/mood (flow state, fear, enjoyment)
- Their personal model of the domain and field (preferences, beliefs, skills)
- Local environment (desk, room, a view, loud noises, co-workers)
- Tools (to make an artifact)
- Probes (to *read* an artifact)

- The artwork itself (and models of the artwork, like sketches)
- Their body and its limitations, abilities, and senses

Social, political, and economic community:

- Social valuations of that craft/art
- Economic values (startup prestige, work-for-pay, cottage crafts, non-valued labor)
- Control over creativity (who owns the tools/art, control of communication channels, censorship)
- Community (friends, neighbors, non-"field" communities)

Not all casual creator systems engage with all the areas of the ecosystem, but as we will see in subsequent chapters, there are dozens of casual creators making use of each way of looking at creativity. Casual creators usually have automated or generative support which is only available through focusing the tool on a well-studied and narrow domain (and embedding its rules and aesthetics in the pipeline). These tools may help users show their work to the field of related practitioner, or participate in economies (toy ones, with online currency, or real ones). The experience of a casual creator is in using the tool as a probe to explore the possibility space, and understanding one's movement through that space by engaging with the artifact itself, not just an abstraction in the user's mind. And finally, we have seen different values placed on different kinds or "levels" of creativity. Casual creators exist within a society that values "little-c" creativity less, and looks at hobbies and non-monetized creativity as a potential waste of productivity. Several subcategories (see chapter 10) of the casual creators I look at in this dissertation are full of so-called "trash apps", which disrespectfully

waste users' time and dole out little creative control in exchange for demanding in-app purchases (or viewing ads). Some forms of creativity receive respect from society, while others do not, and we can see these prejudices express themselves in creativity tools as well. Creativity happens in the user's mind and in their environment: so too are casual creators situated in the algorithms they use, and in the society that builds and uses them.

Chapter 5

Creativity Support Frameworks

> "Self expression, the search for novelty — also called "innovative design" — the demonstration of one's personality, the search for originality, trying to create something that's different, are all motivations which do not lead to variety. They are all negative impulses, urging you not to do something causing the designer to hold back, rather than to give. All negative attitudes inhibit the process of creation; they stop the joy of aesthetic play. [...] Only a positive attitude of generosity, a friendly dialogue with your work from the beginning to the end of its development, will produce pleasant variety." - Eva Zeisel[447]

5.1 Creating creativity

It is well-accepted that certain kinds of software improve cognitive abilities. Video game playing has been show to increase, or at least correlate with, greater precision and speed in surgery[328], and first-person shooters improve attention and multitasking ability, at least in the short term[217]. Schools are adopting *Minecraft*[s136] as a classroom activity and after-school program for its benefits in enabling student creativity[320] and fulfilling a variety of pedagogical and social needs[282].

These games were not *intentionally designed* to improve cognitive ability; this

feature emerged as a result of the gameplay that they implemented. If there exist clear methods for influencing a user's psychology, then presumably software could implement them to intentionally create similar effects. Though the human mind is not a particularly well-documented system, there are known ways to influence perception and behavior[179], and some software designers have used them as rulebooks for designing psychologically fine-tuned systems, superior for their intended tasks. Bruce Tognazzini, for example, used the known principles of stage magic to control users' perceptions of the behind-the-scenes activity of Apple software, creating the illusion of more intelligent or foolish AI, suggesting activity that did not really exist, or hiding activity which did[412].

The question for this dissertation, then, is "how can we create creativity", or more practically, if we accept John Ruskin's hypothesis that the default human condition is one of creativity, "how do we create a space in which creativity can occur?"

5.2 Flow

Some experiential frameworks of creativity *are* measurable and actionable (or at least, have been measured in studies). Csikszenmihalyi's theory of "flow"[110] defines a psychological state of pleasurable focus, a "flow state" of the experience of "joy, creativity, the process of total involvement". Flow is defined as specific properties of the activity, as well as their effect on the participant:

> "a sense that one's skills are adequate to cope with the challenges at hand, in a goal-directed, rule-bound action system that provides clear clues as to how well one is performing. Concentration is so intense that there is no attention left over to think about anything irrelevant, or to worry about problems. Self-consciousness disappears, and the sense of time becomes distorted."

The system **is** *goal-directed* and *provides performance feedback*, and it **causes** *intense concentration* and *a distortion of time*. When we tease apart Csikszentmihalyi's descriptions to separate the *results* from the *conditions* of flow, we find that he presents the conditions that support an autotelic experience.

Here are his conditions for flow, as listed in his later book on creativity[111].

- There are clear goals every step of the way ⋆
- There is immediate feedback to one's actions ⋆
- There is balance between challenges and skills ⋆
- Action and awareness are merged
- Distractions are excluded from consciousness ⋆
- There is no worry of failure ⋆
- Self-consciousness disappears
- The sense of time becomes distorted
- The activity becomes autotelic

I have added a star to the ones that, as system designers we have control over. You could tell a designer to "reduce worry of failure" or "add clear goals", but not "remove self-consciousness" or "merge action and awareness". While the outcome is important (we *do* seek autotelic creativity in casual creators), this chapter is focused on identifying *actionable* design frameworks.

The conditions for flow are created by balancing skills, challenges, anxiety and boredom to maintain a psychological centering **in the channel of flow**. Maintaining a balanced position in the channel, he argues, allows a person in the flow state to maintain focus and gain higher levels of ability and achieves "a creative feeling of transporting the person into a new reality".

Flow state could be attained through any activity, physical or mental, though "to enjoy a mental activity, one must meet the same conditions that make physical activities enjoyable. There must be skill in a symbolic domain; there have to

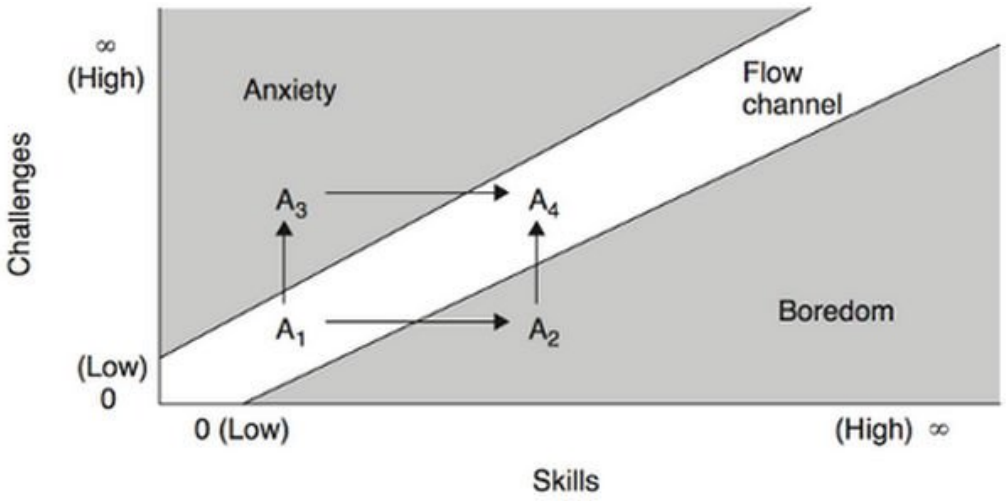


Figure 5.1: The "channel of flow" from [110]

rules, a goal, and a way of obtaining feedback." He notes that challenge and goal-based activities like rock-climbing or tennis are *designed systems* which have the requirements of flow *built into their structure*:

> "When describing optimal experience in this book, we have given as examples such activities as making music, rock climbing, dancing, sailing, chess, and so forth. What makes these activities conducive to flow is that they were designed to make optimal experience easier to achieve. They have rules that require the learning of skills, they set up goals, they provide feedback, they make control possible. They facilitate concentration and involvement by making the activity as distinct as possible from the so-called "paramount reality" of everyday existence."

Open-ended creative pursuits like art or music will require different approaches to the 'rules and goals' requirement, by adding internal structure (like personal preferences and aesthetic heuristics) or external structures (like the Zentangles and paint-by-numbers we will see later in this dissertation). Regardless, the activity must have these structures innately or provide an opening for participants to provide them on their own.

> "In some creative activities, where goals are not clearly set in advance, a person must develop a strong personal sense of what she intends to

> do. The artist might not have a visual image of what the finished painting should look like, but when the picture has progressed to a certain point, she should know whether this is what she wanted to achieve or not. And a painter who enjoys painting must have internalized criteria for "good" or "bad" so that after each brush stroke she can say: "Yes, this works; no, this doesn't." Without such internal guidelines, it is impossible to experience flow." [110]

"Flow" has become one of the most productive frameworks in software design and provides guidance to a wide range of applications, perhaps because it seems so straightforward to enable the conditions for flow in a totally-artificial system like software. Flow theory's impact is perhaps seen most clearly (and explicitly) in video games. Game designer Jenova Chen took it as such a strong influence to his work that his first breakout game was titled "Flow", and a subsequent one was titled "Flower" in homage[59]. These games, and many others likewise inspired by Csikszenmihalyi, use the framework as a set of principles and interventions to keep the user in the 'flow state', offering choice and challenge only in ways that preserve the flow.

Flow has been studied in industrial creativity[232], and self-reported flow states were found to be "significant predictors of employees' creativity." If it has been used in games (an autotelic activity) and professional creativity (a creative activity), I think we may conservatively hypothesize that it will be effective as well in casual creators, an autotelic creative activity.

5.3 Improv, randomness and creativity cards

Randomness can inspire creativity. When thinking or creating, humans have a hard time "setting down" their existing knowledge of the domain or their expectations for "what comes next" in a work. When watching a child tell a story or assemble an outfit, you can see the lack of relational expectations, as they pair

fairies with cardboard boxes, rain boots with tutus. An adult with a few dozens of years more experience will find themselves making choices based on the knowledge of what is "right". Will Wright's interactive TV prototype "Bar Karma"[148] allowed users to submit and vote on plot points, but as I followed the voting back in 2011, I was frustrated that users seemed to vote for the "right" answer, the one that was least divergent from what would be familiar in a plot trope, and thus the least surprising for a television episode.

Bill Gaver, known for his unusual ways of asking HCI questions, warns against the creativity-dampening effects of clarity and certainty and total control, suggesting that instead, unusual or constrained ways of asking questions (like giving users a 15-second non-editable recording toy) can divert people out of their creatively paralyzing expectations and enable creativity.

> "Asking unambiguous questions tends to give you what you already know, at least to the extent of reifying the ontology behind the questions. Posing open or absurd tasks, in contrast, ensures that the results will be surprising. Summarizing returns tends to produce an "average" picture that may not reflect any individual well, and that filters out the unusual items that can be most inspiring."[151]

This principle of "expectation" or "routine" as an enemy to creativity has led to word association as a long-running test of creativity[164], with more divergence in word association being linked to higher creativity. Conversely, the child (or random generator) that pairs ideas because they *don't* know about their connectivity (or lack thereof) is not an ideal model of creativity, so it is suggested that creativity requires both *divergent* thinking, and *associative* thinking (finding commonality and pulling up associated concepts)[29]. Thus, if **divergent** and **associative** thinking are key to creativity, tools that enhance those abilities *should* enhance creativity.

We can see this principle very clearly in improvisational theater. Many of us

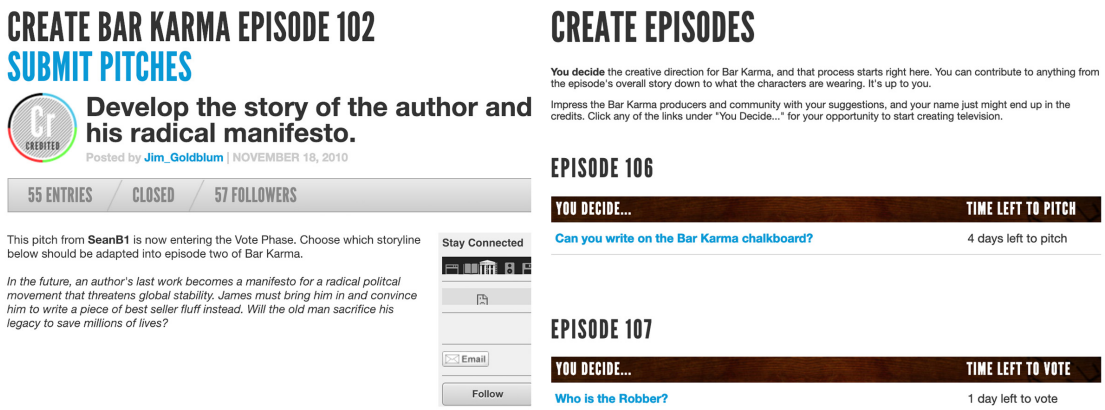


Figure 5.2: Partial image of a voting page for Bar Karma, archived from `https://web.archive.org/web/20110312010512/http://current.com:80/studios/create-episodes/102/develop-the-story-of-the-autho/newest/0/voting/`

may have seen improv troupes draw ideas out of a hat (or have done so ourselves). Techniques like this often have their roots in Keith Johnstone's very influential "Impro" theory books. In **Impro for Storytelling**, he identifies the difficulty of balancing ideas that are *associative* (what should logically happen) with those that are *divergent* (what is interesting or strange).

> "Beginners feel uncreative when an activity nears completion. Some compensate by choosing a 'clever' activity like scuba-diving in boiling custard - but the audience says to itself, 'So they scuba-dived in boiling custard - what's the point?' To solve this problem I defined anything that the improvisers were doing as a routine - poking the fire, reading this book, tracking a moose (whatever) - and I argued that the audience pays to see routines being broken. The routine of 'watering a flower' is broken when it says, 'Thank-you!' The routine of waking up in the morning is broken when you discover that you're in the wrong house."[210]

These breakages are called "tilts" in Impro. The players must exercise judgment in knowing when a tilt is needed, and must use a combination of associative and dissociative thinking to identify a viable tilt. But if they need a creativity aid, he states:

> "Tilts can be added arbitrarily. This encourages stronger platforms (absolving the players of the need to think ahead, and to 'be clever'). Players can easily make random tilts seem as if they were generated by the platform. For example, a director in Gorilla Theatre asked for a parent-child scene, and we saw a bad poet having his verses corrected by his daughter. Then a tilt written on a strip of paper was taken from a hat. It read: 'It's time I told you about the family curse!', so the father explained that for hundreds of years the men in the family had been doomed to be bad poets"

This augmentation of divergent thinking is done by a "machine" (tilts written on a piece of paper in a hat), but the associative thinking is done by the human. They are tasked with weaving the incongruous bit of divergence into an associative whole. This seems to be an ideal division of labor: not only are both the "machine" and the human playing to their strengths, but the humans really seem to enjoy this activity (see the later chapter chapter 24 for more on the "imagination playground" that this creates). *Mad Libs*[s128] is a classic, popular, and long-standing form of this machine-divergent/human-associative game. We suggest words into the slots with the obscurative assistance of the *Mad Libs* form, then amuse our friends by reading and imagining the resulting story. Perhaps it should be no surprise that we see this pattern again and again, and adopted into digital forms. Writer, artist and bot-ist Harry Josephine Giles describes their Tracery bots:

> "We see the human in the robot and the robot in the human [...] They are the best comedians: they repeat a joke until it isn't funny, then until it is again, then until it isn't again"[156].

When Brian Eno and Peter Schmidt created their *Oblique Strategies*[s150] deck of cards for music and art inspiration they were inspired by the a colleague's use of the *I Ching*[s95] divination tool as an art provocation.[352]. The *Oblique Strategies* cards are described as "idea-generating tools and tactics designed to break routine thinking patterns", and so we see the themes of associative (generating)

and divergent (breaking) ideas rising again. Decks of cards, dice, ideas in a hat, and random number generators all have randomness built into them.

In *Fiasco*[s73] by Jason Morningstar , a group storytelling game, dice are used to help limit options, by acting as a limited resource that can be spent by the improvisers to "buy" characters and motivations. Later on, they select random tilts, which are then woven into the story by the players as "the exact interpretation of the outcome is not predetermined but created through play in the moment"[318]. Is it a coincidence that the terminology of the "tilt" is the same as Johnstone's, or that both dice and a randomly-drawn suggestion are used to keep the players *in the moment*? Whether or not Morningstar has a copy of Impro on his shelf or not, these two systems share a similar set of needs (augment partially-creative people in improvisational storytelling) and seem to have found a similar solution.

Cards, especially, seem to be useful for creativity, as they allow a variety of randomization affordances (shuffling, drawing), as well as ways to juxtapose cards (laying them next to each other, tucked under one another, or suggesting a connection) and social maneuvers (sharing cards, giving cards, holding a card and shaking it at another person while yelling). Adam Rothstein proposes an "Open Card Deck Standard" unified format for card decks (including both collectible sports cards and *Oblique Strategies*)[330], and notes some inherent properties of cards and how they change how we think:

"(B)y breaking text into smaller chunks by printing it on physical cards, we can more easily violate the categories of information, while not breaking the rules of language or intelligible thought. This allows our brains' creativity to come in and repair the categories in an ad hoc manner, or manipulate them according to predetermined rules."[330]

The juxtaposition of cards may play a role in adding a mechanically-assisted

"associative" property to the cards, as many inspirational creativity decks [248][252] have relational elements to them that encourage multiple cards to be considered *in relation to each other*. *Generominos*[s82][88], a deck of generative art design cards I designed[1], guide users in creating valid pipelines of generative methods by matching up input/output symbols that represent data types (like numbers, soundwaves, or 3D meshes). The *Generominos* cards have very specific connective constraints that represent the constrained ways that real generative algorithms can connect to each other. These constraints offload some of the associative role from the player: they can check their connections are valid or sort through the deck for cards that could connect to a given input or output. Yet for all that constraint, playing with *Generonimos* often has a large element of randomness, as players grab nearby interesting cards that they might not have otherwise considered, and attempt to work them into their designs. We thus see an echo of Johnstone's improvisors interweaving the tilt into a scene, but in a generative design.

Dissociative thinking can also be described as a feeling of internal conflict, of containing multiple actors. The synthesis and mediation of their conflicting demands creates creative ideas. Psychologist and writer Jerome Bruner cites "internal drama" played out by "dramatic personalities, producers of surprise" as a condition of creativity: "...it is the working out if conflict and coalition within the set of identities that compose the person that one finds the source of many of the richest and most surprising combinations"[40].

Creativity guru Edward De Bono suggests a 'six-hat' method: create mental characters arguing for their values of "managing", "information", "emotions", "discernment", "optimistic response", and "creativity". By pitting these voices against each other, new ideas will become synthesized by their juxtaposition. Similar games and activities can be found in Thinkertoys, a compendium of such

[1]More on the *Generominos* in chapter 6.

techniques by Michael Michalko [271], several which use randomness, and the book suggests the "suggestions in a hat" technique from Impro to "Select Thinkertoy at random. Write the Thinkertoy titles on slips of paper and put them in a container. When you have a challenge, shake the box and randomly retrieve the name of the toy to use."

Jesse Schell created a similar method of divergence and reassociation for video games with his "Book of Lenses"[346] a set of provocations for leading a game designer to consider new and conflicting voices about a design. Like many of original Oblique Strategies cards, these cards are intended to get the designer to reconsider the game from a perspective outside their own mind (like a player or a marketer).



Figure 5.3: Images from the "Game Design: A Book of Lenses" Android app. It has a Tarot-like "divination" mode where four cards are randomly drawn and juxtaposed together.

In this section, we have seen many ways that theories of creativity were incorporated into non-digital methods and tools (with some later spawning digital offspring). In the next section, I will examine "Creativity Support Tools", a

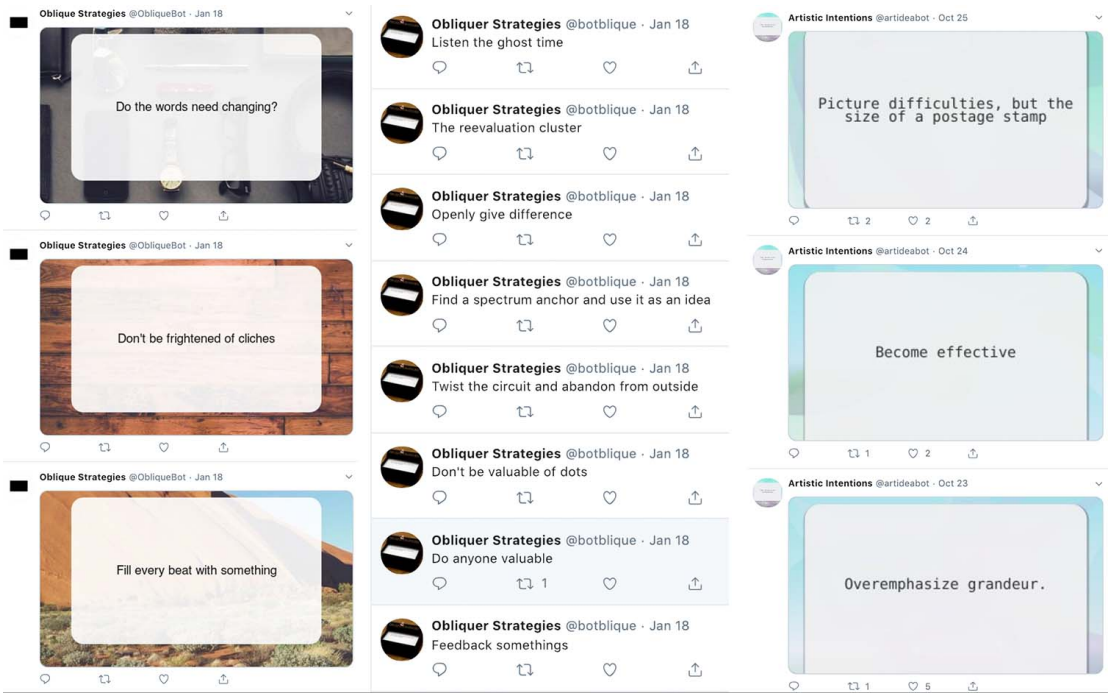


Figure 5.4: @sambreed's @ObliqueBot posts cards from the Eno/Schmidt deck, @norareed's @botblique and my own @artideabot create generative riffs on the Eno cards, doubling down on the divergent properties of the original deck

framework designed for the considerations of digital tools.

5.4 "Creativity Support Tools"

These many different techniques for enabling creativity demonstrate that an experience of creativity can be, to some extent, externally assisted. The flow state can be created with a balanced feed of challenges. Playfulness and improvisation and divergent thinking can be induced by following a set of instructions or playing mental games. Sternberg derides these techniques (specifically De Bono's) as unscientific:

> "Equally damaging to the scientific study of creativity, in our view, has been the takeover of the field, in the popular mind, by those who follow what might be referred to as a pragmatic approach. Those taking this approach have been concerned primarily with developing creativity,

> secondarily with understanding it, but almost not at all with testing the validity of their ideas about it"[394]

Developing creativity is a real goal — for this dissertation, and for business, society, and education. This dissertation is evaluated (however non-empirically) in an evaluation of its use to practitioners (see chapter 16). Experientially, if not scientifically, these activities are found to be useful for creative types (anecdotally, I've heard many professional game designers rave about Jesse Schell's lenses). Thus, if creativity is a fluid state that we can induce or prevent, even Sternberg, in "Creativity as a Habit", agrees that we can build systems, social or digital, that can influence the creative state:

> "Like any habit, creativity can either be encouraged or discouraged. The main things that promote the habit are (a) opportunities to engage in it, (b) encouragement when people avail themselves of these opportunities, and (c) rewards when people respond to such encouragement and think and behave creatively. One needs all three. Take away the opportunities, encouragement, or rewards, and one will take away the creativity. In this respect, creativity is no different from any other habit, good or bad." [393]

Many of the examples so far in this chapter share a framework of creativity based on divergent thinking, as encoded in mental exercises. However, creativity support tools are equally well suited to forms other than paper instructions, like being encoded in software. They can also explore and operationalize many other frameworks of creativity as well. As this idea is practiced and encouraged in a number of different fields, it has adopted names like "information technology in creative practice" or "creativity support tools"[366], a term popularized by Ben Shneiderman (also the inventor of the "direct manipulation" term we will see in chapter 8), who further developed the idea in many papers[368], a book[367], and a "Creativity Support Tools" NSF workshop[369]. *Beyond Productivity*[107], a

National Research Council report on creativity support through software, heralds this as an excitingly futurist intervention:

> "This report argues that, at the beginning of the 21st century, information technology (IT) is forming a powerful alliance with creative practices in the arts and design to establish the exciting new domain of information technology and creative practices—ITCP."[107]

Ben Shneiderman, however, sees the merging of creativity and technology as instead a continuation of the past:

> "A critic might say that creativity is inherently human and no computer could or should be brought into the process. But technology has always been part of the creative process, whether in Leonardo's paint and canvas or Pasteur's microscopes and beakers. Supportive technologies can become the potter's wheel and the mandolin of creativity–opening new media of expression and enabling compelling performances."[367]

Both "Beyond Productivity" and the report of the NSF Workshop on Creativity Support Tools cite the need for greater understanding of the conditions for creativity: "It is possible to identify and establish the conditions necessary for creativity, and conversely, we risk stifling creativity if we get those conditions wrong."[107], and Creativity Support Tools provides a more grounded, application-specific framework. This framework suggests methods to facilitate social creativity (similar to Csikszentmihalyi's domain evolutions), such as creating online collaborative spaces so that more diverse perspectives can contribute and clash and collaborate, and "exploiting conceptual collisions and breakdowns to stimulate imagination and invention", an externalized version of Jerome Bruner's internal creative drama.

Schneiderman suggests that these tools actually need to consider **three different intersecting theories of creativity** (structuralist, inspirationalist, and situationalist) when designing tools:

- "Structuralists believe people can be creative if they follow an orderly method, typically described with several stages, such as preparation, incubation, illumination, and verification."

- "Inspirationalists argue that breaking away from familiar structures elicits creative solutions."

- "Situationalists recognize that creative work is social. They seek to understand the motivation of creative people, their family history, and their personal relationships with challenging teachers, empathic peers, or helpful mentors"[368]

Each of these perspectives requires a different set of features and interventions to support creativity:

> "Structuralist thinking encourages systematic tools that include progress indicators with reminders of what is still needed. The inspirationalist view supports development of image libraries, thesauri, sketching interfaces, and concept-mapping tools. Situationalists broaden the designer's view to include email and collaboration tools, as well as the e-science notebooks that guide users and coordinate groups through scientific processes over weeks, months, and years."[368]

The Creativity Support Tool framework has evolved over its many separate publications. In "Design Principles for Tools to Support Creative Thinking"[314], an individual sub-report from the "Creativity Support Tool" workshop, Resnick et al suggest twelve concrete design principles for building such tools:

- **"Support Exploration":** not only should the user be able to find alternative solutions in the space, they should be able understand what the tool itself is capable of, discovering the domain space of the tool

- **"Low Threshold, High Ceiling, and Wide Walls":** a tool with a low threshold can be learned very quickly, or at least, be understood enough

to use the tool, but not necessarily mastered. A high ceiling means that, once mastered, the tool can be powerful enough to create very sophisticated solutions. A tool with wide walls has a large and interesting possibility space

- **"Support Many Paths and Many Styles":** recognize that both hard and soft approaches have merits, and so don't force users to work in a mode they aren't comfortable with
- **"Support Collaboration":** celebrate the strengths of collaborators, and search may be understood as a way to collaborate with past users and thinkers.
- **"Support Open Interchange ":** Many tools may be employed in a single task, consider data portability
- **"Make It As Simple As Possible - and Maybe Even Simpler":** reduced features do not always mean reduced utility. Users will find more unexpected uses for a tool with few features than designers would predict.
- **"Choose Black Boxes Carefully":** sometimes abstractions can be useful, but often they can hide necessary information
- **"Invent Things That You Would Want To Use Yourself":** if the designer doesn't take pleasure in a system, who would?
- **"Balance user suggestions, with observation and participatory processes":** with simplification and black boxes, a user could still ask for greater complexity without understanding the drawbacks of cognitive clutter
- **"Iterate, Iterate - Then Iterate Again":** no-one knows what users will do. A designer can make predictions, but only user testing can prove them out

- **"Design for Designers":** be aware that you cannot anticipate your users goals, make a system extensible enough to surprise you

- **"Evaluation of tools":** usability and usefulness are not the same thing, so be wary that "tools that are not effective and efficient will probably hinder creativity, but it isn't clear that the reverse will hold"

These design principles are clearly applicable to casual creators, as well as general creativity tools. Exploration of the possibility space is a core value of casual creators, but these principles suggest exploration of the tool as a significant process as well. Collaboration and data portability are important for professional creators and scientists, but as creativity tools like *Twine*[s236] and *Minecraft*[s136] show, users value being able to host and share files independent of a central server — a portability which also encourages sharing communities and the construction of third-party add-ons to the tool.

Not all creativity support tools are casual creators (for example, *Maya* and *Overleaf*). Not all casual creators are creativity support tools (which still focus on *productive* creativity).

As the last chapter ended with a list of the many sites of creativity, this chapter can end with a reminder of the range of those sites (from personal to perceptual to social/economic). Support tools, including casual creators, *can* support creativity, but we must remember all the sites and consider how our software does or does not provide support in them.

Chapter 6

Existing Frameworks of Generativity

> "Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent." - Ada Lovelace[247]

6.1 The case for frameworks

The digital games industry is now several decades old, and throughout its history, practitioners have documented their thoughts on this art form. A common concern for games creators and researchers is providing frameworks with which to understand games: what they are, how they are played, what their component parts are.

"[A] game is a closed formal system that subjectively represents a subset of reality" defines game designer Chris Crawford in 1984.[109] He proposes taxonomies of games, and in doing so, defines his preferred lenses (platform-based) to create grouping of games (like computer games and arcade games). Once done, he lists

the essential elements of game design: "We shall return to the subject later, to create a taxonomy of computer games, and later still to draw on specific examples of games to make points about their nature. We must now address the question which motivated our initial reconnaissance: what are the fundamental elements common to these games?" The rest of the book answers this question, with a directory of elements, such as conflict and its many subclasses (physical, verbal, political, economic, direct and indirect). Once the framework is defined, he has a language that he can turn back to analyze these elements in games. This ontology, once defined, can also be used to compare the game concepts to the corresponding concepts in other disciplines, like comics and cartoons and evolutionary psychology.

Other works provided frameworks of player experience in games[25], best-practice design patterns for games[225], and an architecture linking human experience to raw system mechanics[202].

Which of these ontologies captured the "ground truth" of games? Each one is myopic in its own way, false in its own way, and truthful in its own way. Taken together (along with all the other proposed game frameworks in the recent explosion of game studies) they provide illumination on every discipline and purpose of game design.

A curious researcher or designer can google "game studies framework" and get a hundred different lenses on the field. Whatever discipline is looking at games, they still call them "games". Games, for whatever reason, developed very clear barriers early in their historical development. An average person can look at most commercial digital software and immediately tell from the marketing, interface, and graphics, whether or not this is a "game"[1]. Perhaps due to marketing pressure,

[1]Games academics like to debate whether something is a game or not based on increasingly precise distinctions, but this bug does not seem to have affected marketers or app stores.

systems on the borders of other disciplines tended to move either closer to, or further from games. The result is a relatively isolated island within the larger ocean of general "software" or "apps". Is our ability to study games (in the range that we now see them) related to our ability to define and isolate "gameness" from other software? Does the existence of a framework enable us to build a field?

I cannot argue whether early frameworks helped define digital games as a clear category, or whether they influenced early game designers to hew closer to the orthodoxy of what games were first defined as by writers like Crawford. Did the existence of game frameworks help create the eventual well-recognized field of game studies, or did both come into existence because games are, in some elemental way, definable?

Questions like these are far outside the scope of my expertise and this dissertation. I would argue, though, that casual creators are just as definable as games (indeed that's the founding premise of this dissertation), but have not developed as a coordinated discipline like games. **An absence of a good framework is likely not the cause of this situation. But I believe that it could be of use in ameliorating it.**

This dissertation intends to provide a tool belt of terminology, concepts, and frameworks. In the following two sections, I will catalog a few existing frameworks of generativity and interactivity. I will also propose my own framework of generativity (the generative framework of generativity, or GFG) and framework for **interactivity with a generative system**, with the goofy-yet-catchy term of the "grokloop".

Casual creators always include *some* form of assistance or automation. Sometimes this is a set of rules (like rules to weave a tartan pattern[s239]) or a physical form that somehow encodes those rules (like a Spirograph[s214]), though often it is

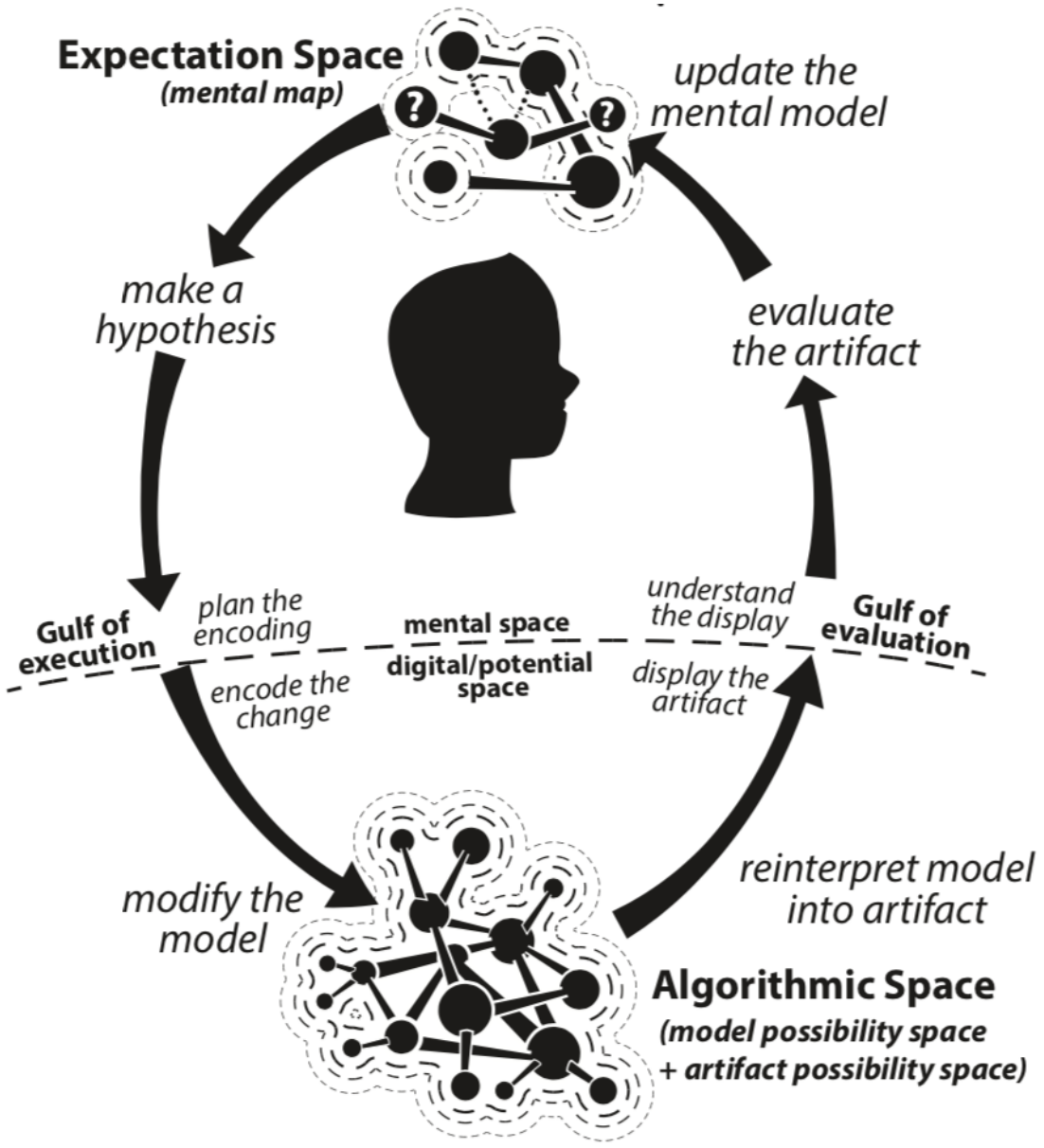


Figure 6.1: The Grokloop, which will become much clearer after the next two chapters.

a digital form, like the rules and tuning values of a Lindenmayer system running on a computer. This back-end assistance is required to transform a *small action or choice* from the user into something *larger* or *more complex* without increased effort or training on the part of the user. This *emergence* is part of what makes

casual creators feel effortless yet powerful, and why generative methods seem so inextricably linked to casual creators.

Several frameworks exist to describe how generativity (or procedural content) can be understood, or how it can be used in games and art. In this chapter, I present a framework that considers generativity as a pipeline of successive data transformations, with each transformation either generating, transforming, or pruning away information. This framework has been iterated through repeated engagement and education interactions with the game development and generative art communities. In its most recent refinement, it has been physically instantiated into a deck of cards, which can be used to analyze existing examples of generativity or design new generative systems.

6.2 Generativity across domains

Generativity has been a source of wonder, surprise, creativity and meaning for millenia, since we cast knucklebones and charred sticks and made up stories about the results. A lack of user control in these systems was often not a flaw, but a virtue, as the randomness allowed the will of the gods (or the hand of fortune, or the spirit of the dead) to intrude and make its intentions known.[166]

Some of these generative systems were fortune-telling or story-telling devices, like the Tarot[s226], I Ching[s95], Ouija boards[s157], M.A.S.H. boards and *Mad Libs*[s128]. Others were employed as components of competitive games, like drawing a hand of blackjack cards, or rolling Yahtzee dice, or became the settings for competitive games, like *Betrayal at the House on the Hill*[s12] or *Settler of Catan*[s200]. Other forms of generativity turn small and mechanistic repeated movements into complex patterns, like cable-knitting patterns, tartan patterns, pottery wheel techniques, or the strange-attractor musical patterns of analog synthesizers[376]. Each

pattern of the crafting canon must have surprised the first artisan to attempt it, as they saw the new pattern emerging from their novel algorithm.

Generativity *always* requires the ceding of control in order to receive, in exchange, the powers of an alien (or algorithmic) logic. By understanding this trade-off, we can find common ground in *all* generative methods, across many divergent domains and fields of study.

Generativity touches many fields, so unsurprisingly many fields have written their own frameworks for generativity. From my background and social circles, I'm most familiar with those from digital games. It was initially difficult to find frameworks in other fields, not because they don't exist, but because each field has its own terms for it, or no terms at all. Architects refer to "parametric" architecture. Musicians to algorithmic or generative music (or "patch-based"), so to do artists refer to generative art. Game designers talk of "procedural generation", or "procedural content generation". It was the exposure to these other fields that led me to start using "generative" or "generative methods" rather than "procedural content generation", to reduce the terminological isolation of generativity for games[86].

6.3 Terminology for "generativity" across fields

Fields have different terms for generativity. Some (like literature) have terms that encompass only a few kinds of generativity. Others, like mathematical art, contain some generative work but also other kinds of production.

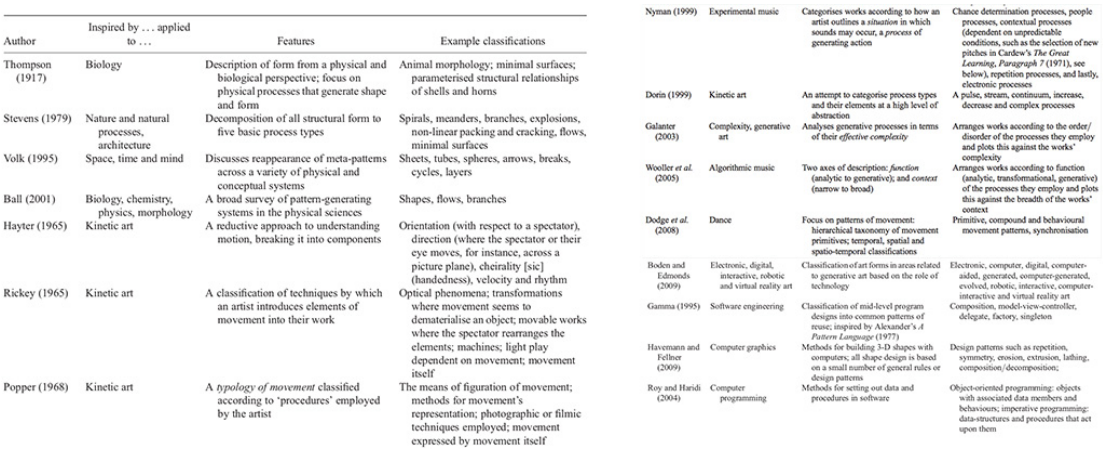
- Art: "generative art", "algorithmic art", "mathematical art" (older term)
- Design: "generative design", "parametric design"
- Architecture: "parametric architecture"
- Music: "algorithmic music", "generative music", "patch-based"

- Games: “procedural content generation”, “PCG”, “procedural content”
- Literature(rare):“combinatorial”, “constrained”

Fields of craft, like knitting, quilting, marquetry, glass-working, or weaving are decentralized and lack the other fields’ more formal ways of recording implicit knowledge and building shared vocabulary. Millefiore glass-working and Yosegi-zaiku woodworking have very similar forms of generativity (bundle, slice and layer thin strips of material). If a glass-worker in 19th century Venice wrote a treatise on the emergent qualities of millefiore, and a Yosegi-zaiku woodworker had the same insight about their patterns of bundled and sliced wood, they probably would not have found each other’s writing. Due to the historical lack of academic work around craft (combined with a lack of process-focused analysis), there are very few second-hand or academic readings of emergence in crafts. There are no prior frameworks of generativity in crafting to quote in this chapter. My framework provides a way to understand at least some forms of emergent crafting, but there is little to compare it to.

In contrast, many of the fields with established academic presence *have* written frameworks, each one sharing many analogous insights with the others.

In a paper I’ll be discussing in more depth later[123], Dorin, et al, catalog various generative frameworks across fields. While I won’t discuss all of these frameworks from their list in this dissertation, I will go over the ones that influence or relate to my framework.



Author	Inspired by … applied to …	Features	Example classifications
Thompson (1917)	Biology	Description of form from a physical and biological perspective; focus on physical processes that generate shape and form	Animal morphology; minimal surfaces; parameterised structural relationships of shells and horns
Stevens (1979)	Nature and natural processes, architecture	Decomposition of all structural form to five basic process types	Spirals, meanders, branches, explosions, non-linear packing and cracking, flows, minimal surfaces
Volk (1995)	Space, time and mind	Discusses reappearance of meta-patterns across a variety of physical and conceptual systems	Sheets, tubes, spheres, arrows, breaks, cycles, layers
Ball (2001)	Biology, chemistry, physics, morphology	A broad survey of pattern-generating systems in the physical sciences	Shapes, flows, branches
Hayter (1965)	Kinetic art	A reductive approach to understanding motion, breaking it into components	Orientation (with respect to a spectator), direction (where the spectator or their eye moves, for instance, across a picture plane), cheirality [sic] (handedness), velocity and rhythm
Rickey (1965)	Kinetic art	A classification of techniques by which an artist introduces elements of movement into their work	Optical phenomena; transformations where movement seems to dematerialise an object; movable works where the spectator rearranges the elements; machines; light play dependent on movement; movement itself
Popper (1968)	Kinetic art	A *typology of movement* classified according to 'procedures' employed by the artist	The means of figuration of movement; methods for movement's representation; photographic or filmic techniques employed; movement expressed by movement itself
Nyman (1999)	Experimental music	Categorises works according to how an artist outlines a *situation* in which sounds may occur, a *process* of generating action	Chance determination processes, people processes, contextual processes (dependent on unpredictable conditions, such as the selection of new pitches in Cardew's *The Great Learning, Paragraph 7* (1971), see below), repetition processes, and lastly, electronic processes
Dorin (1999)	Kinetic art	An attempt to categorise process types and their elements at a high level of abstraction	A pulse, stream, continuum, increase, decrease and complex processes
Galanter (2003)	Complexity, generative art	Analyses generative processes in terms of their *effective complexity*	Arranges works according to the order/disorder of the processes they employ and plots this against the works' complexity
Wooller *et al.* (2005)	Algorithmic music	Two axes of description: *function* (analytic to generative); and *context* (narrow to broad)	Arranges works according to function (analytic, transformational, generative) of the processes they employ and plots this against the breadth of the works' context
Dodge *et al.* (2008)	Dance	Focus on patterns of movement: hierarchical taxonomy of movement primitives; temporal, spatial and spatio-temporal classifications	Primitive, compound and behavioural movement patterns, synchronisation
Boden and Edmonds (2009)	Electronic, digital, interactive, robotic and virtual reality art	Classification of art forms in areas related to generative art based on the role of technology	Electronic, computer, digital, computer-aided, generated, computer-generated, evolved, robotic, interactive, computer-interactive and virtual reality art
Gamma (1995)	Software engineering	Classification of mid-level program designs into common patterns of reuse; inspired by Alexander's *A Pattern Language* (1977)	Composition, model-view-controller, delegate, factory, singleton
Havemann and Fellner (2009)	Computer graphics	Methods for building 3-D shapes with computers; all shape design is based on a small number of general rules or design patterns	Design patterns such as repetition, symmetry, erosion, extrusion, lathing, composition/decomposition;
Roy and Haridi (2004)	Computer programming	Methods for setting out data and procedures in software	Object-oriented programming: objects with associated data members and behaviours; imperative programming: data-structures and procedures that act upon them

Figure 6.2: The table of notable frameworks of generative art from Dorin et al. Don't worry about reading them individually, just note how *many* there are.

6.3.1 Generative art frameworks

"What is Generative Art", Margaret Boden and Ernest Edmonds

Creativity researcher Margaret Boden and long-time computer artist Ernest Edmonds wrote this 2009 article[35] to answer the question of what "generative art" is and how it relates to other forms of digital, interactive art. To map the space, they create a loose taxonomy of 11 overlapping categories.

- Ele-art (electronic): "any artwork whose production involves electrical engineering and/or electronic technology"
- C-art (computer): "art in whose productive process computers are involved"
- D-art (digital): "uses digital electronic technology of some sort"
- CA-art (computer-assisted): "the computer is used as an aid (in principle, non-essential) in the art-making proces"
- G-art (generative): "the artwork is generated, at least in part, by some process that is not under the artist's direct control."

- CG-art: "the artwork results from some computer program being left to run by itself, with minimal or zero interference from a human being."
- Evo-art: "evolved by processes of random variation and selective reproduction that affect the art-generating program itself."
- R-art: "the construction of robots for artistic purposes, where robots are physical machines capable of autonomous movement and/or communication"
- I-art: "the form/content of the artwork is significantly affected by the behaviour of the audience"
- CI-art: "the form/content of some CG-artwork is significantly affected by the behaviour of the audience."
- VR-art: "the observer is immersed in a computer-generated virtual world, experiencing it and responding to it as if it were real"

Many of these categories overlap — for example, almost all digital art is computer art. Most computer art is also electronic and digital art (except for the rare examples made with analog or non-electric computers, like Desmond Paul Henry's drawing machines[288] or the Phillips Hydraulic Computer[2]). The taxonomy becomes more productive when they start teasing apart what it means to make art with computers as *an incidental tool* (like a digital artist using *Photoshop* instead of paint) and using computers as an active part of the process. We hear echoes of Heidegger's "transparent tool" metaphors (see more in chapter 8) in which the computer is a channel for the human artists intentions.

[2] `https://www.youtube.com/watch?v=rVOhYROKeu4`

> "Indeed, the phrases "computer art" and "generative art" are often regarded as synonyms. Notice, however, that in our terminology not all C-art is CG-art. CA-art is not, because the computer is there used as a tool subject to the artist's hands-on control (and is of no more philosophical interest than a paintbrush or a chisel). That's why we called this paper "What is Generative Art?", rather than the seemingly synonymous "What is Computer Art?""

A "G" or "CG-art" system is built by the hand of an artist, but intentionally cedes control over the production to some external process, whether natural, social, or the actions of an algorithm.

> "In G-art, (df.) the artwork is generated, at least in part, by some process that is not under the artist's direct control. This is a very broad definition. It does not specify the minimal size of the "part". It does not lay down just what sort of generative process is in question. It does not say what counts as being outside the artist's direct control. And it is silent on the extent (if any) to which the processes concerned may have been deliberately moulded by the artist before 'losing' direct control. In short, our definition of G-art is largely intuitive. In general, it picks out cases of art-making in which personal control is deliberately diminished, or even wholly relinquished, and relatively impersonal processes take over. ... Those impersonal processes vary greatly. They may be physical, psychological, sociocultural, biological, or abstract (formal). And if abstract, they may or may not be implemented in a computer."

They give many examples of these "impersonal processes": Bryan Johnson's The Unfortunates, a 1969 novel to be reassembled by rolling dice, Haydn's dice music, wave-based and insect-based artworks, as well as human-but-non-intentional forces like the surrealists automatic writing. They recognize, for G-art, that these external forces may be mediated or "moulded", and that the human artist may still maintain control in some ways. Oddly, when they move to CG art, they claim that control must be fully given over to the computer, with no human meddling, or, conceding the impossibility of that, with human intervention minimized as

much as possible, leaving mixed initiative and more collaborative partnerships between humans and their CG processes without a place in the hierarchy.

I-Art, and CI-Art do allow for the interactions between audience and the artwork. They create a broad definition of "behavior", encompassing traditional forms of interactions, but also the "interaction" of being present, breathing and emitting heat, in a space, and even the possible interactions of a non-human environment with the piece (though this is better described as G-art, they argue).

Like many of their distinctions, I-Art and CI-Art raise issues of control, and even authorship. Interactive and generative systems are not static artifacts, but a space of artifacts. Can the artist be the "author" of the artifacts in a possibility space that they had never predicted?

> "[I]n sufficiently complex cases, it is not even clear that they can recognize the general potential of their own work. With regard to CI-art, then, perhaps we should speak not of the "artwork" but of the "art system"–where this comprises the artist, the program, the technological installation (and its observable results), and the behaviour of the human audience? (And perhaps, if the concept of the "artwork" falls, then that of the "artist/author" falls too?)"

This argument digresses into a bizarre and fascinating analysis of what rule-style programming versus constraint-satisfaction programming means to an artist.:

> "One could argue that the art-community's objection to a step-by-step algorithm is more a matter of taste than anything else. For even when a programmer has written explicit step-by-step code, he or she does not necessarily–or even usually–know the outcome. " "However, computer artists–and computer scientists, too–know from their own experience that when writing algorithmic code there is a 'feel' of fully controlling the computer. This 'feel' does not carry over to systems where they are specifying sets of rules, or constraints. In other words, rule-driven systems appear to have a greater degree of autonomy, relative to the conscious decisions of the human artist. That phenomenological fact is significant because autonomy is a concept that's closely connected with art-making (see Section V). This explains why computer artists are more comfortable in speaking of "generative art" where the system

> is rule-driven, not algorithmic–and why they usually avoid step-by-step programming."

This paper, perhaps because it is co-written with an artist of generative works, takes a deeper look at the aesthetic and authorial considerations of the actual algorithms than I have ever seen in any other work. How much evolution should take place in each step of an Evo-Art piece? If evolution is too rapid, control is lost: "radical mutations" may be interesting to watch, but they erode the artist's sense of personal style. Even the choice of fitness functions for an evolutionary algorithm has implications for ownership:

> Does Evo-art leave more room, or less, for human authorship than AARON does? That is, does the artist's choice of fitness function suffice to give him/her the authorial credit for whatever artwork emerges after many generations of random (i.e. undirected) change? Is the credit greater, or less, if instead of relying on a programmed fitness function the artist does the selecting 'by hand'?

This article identifies many useful properties of these types of generative and interactive art: authorial control, external control, possibility spaces and how to understand them, whether or not interaction is intentional, and more. But I find this framework frustrating. Despite their clear interest in these issues, the taxonomy they propose does not address them at all, instead focusing on surface-level implementation details, like whether the art is a robot or in virtual reality. Tellingly, only 8 years after the article, there are many other forms of art that I would add to the list, like augmented reality, neural-network/deep-learning art, and conversational AIs. They did not anticipate these forms, and there's neither a place in their taxonomy to put them, nor relevant ways to use this taxonomy in understanding or building them:

> As in mathematics or computer science, the fact that the problem space (of making the artwork) is bounded by clear constraints does

not necessarily lead to any method or process for designing or making art.

This paper represents one of the few frameworks that attempts to address the artist's experience and feelings of ownership of the work. The taxonomy hints at the importance of that property, and they way that control can be withheld or ceded, between the artist, system or viewer. Yet many of the distinctions in the taxonomy divide the work up by its components (robots, evolutionary algorithms, VR), without addressing how those components could work with each other, and what axes of control/freedom/ownership can be explored *within* each category.

Dorin - "A Framework for Understanding Generative Art"

The closest framework for my needs is the one describe in Dorin et al's "A Framework for understanding Generative Art"[123].

The authors of this work seek a framework that can describe interaction, complexity, and process in all generative artworks. They examine and discard many past frameworks. Works by D'Arcy Wentworth Thompson and Philip Ball [408][20] express natural process and its emergent properties, but artists make use of artificial processes, which may not map to natural ones. Like me, Dorin et al object that many of Boden and Edmond's categories erase the distinctions of the process in favor of classification by output platform, like robots and VR.

They propose a framework that looks at each generative artwork as a construction of four types of components: entities, processes, environmental interaction, and sensory outcomes. Entities are persistent state-carrying units with individual or shared attributes — like cells in Conway's Game of Life, or agents in a simulation. These entities are acted on by processes, actions with initialization conditions and termination conditions. Interactors, the creator, or other environmental forces can set conditions or parameters or entity attributes: these environmental

interactions are “flows of information between the generative processes [...] and their operating environment.” The entities and processes are captured in static form (“snapshots”, “endpoints” or “accretions”) or viewed continuously or “live”. All entities and outcomes may be visible (“flat” system) or translated, a translation with may be structurally similar (“natural mapping”) or arbitrary (for which they use the example of cellular automata mapped to musical notes, lights, or knitting).

I appreciate this framework’s focus on process, and recognition that identical processes can be viewed though different “sensory outcomes” like the examples of mapping CA’s to a wide range of output platforms: “All of these examples share similar processes, yet their physical manifestations and aesthetic experiences are significantly different, making mapping central to the artistic contribution of each system. “ Despite not being necessarily focused on interaction, the framework acknowledges different kinds of interaction, and can “characterise these interactions in terms of their frequency (how often they occur), range (the range of possible interactions, or amount of information conveyed) and significance (the impact of the information acquired from the interaction on the generative system).”

This framework’s “environmental interaction” and “sensory outcomes” are shorthand for too many subcomponents which have very different properties and effects on user interaction. The physical forms of the output, how the internal representation of the system is translated to those forms, the experience of the viewer, and the viewer’s interpretation of the system are all rolled into “sensory outcomes”. Likewise, it is unclear in “environmental interaction” whether this is one-way input, or also encompasses action-reaction interaction, as the world “interaction” lacks directionality (“All generative systems operate within a wider environment from which they may draw information or input upon which to act.

The environmental interaction component of our framework describes flows of information between the generative processes [..] and their operating environment. “) Does this interaction act upon the system (“configuring the states of cells on a cellular automata grid or generating an initial population of agents in a simulation”, “Incoming information might also set or change parameter values during execution”)? Is it then an “interaction” or a “process”?

I borrow heavily from Dorin’s framework for my generative framework, so here are a few of the modifications I am making:

- separate the “inside” and “outside” of a system, so that I can better compare the mechanical implementation of process with the modeling of that process that occurs in a viewer/interactor’s mind
- remove “entities”, which, until they become part of the output, are only blackboards on which processes store information, and so are better kept under the concept of process, if they are needed at all
- express “input” and “output” as initiators and endpoints of streams of data, respectively, clarifying that any chain of processes must begin with an input and end with an output.
- use this more mechanistic representation to express the mechanical affordances and emergent properties of a system, **before** trying to express how a user will perceive and be affected by the system

In this chapter and the next, I explain two frameworks, of Generativity and Interactivity. Dorin’s framework conflates the two, but as seen in the next chapter, there is a big difference between *the generator* and *how people interact with, experience, and explore it.*

6.3.2 Generation-in-games frameworks

There is no shortage of frameworks available for describing generative systems used to create game content. These categorize generative systems by the type of content they produce, by the type of algorithm used to filter content, or their use in gameplay or content construction.

Hendrikx et al - "Procedural content generation for games: A survey"

The framework by Hendrikx, et al[190], classifies what can be generated by a generator: sound, textures, vegetation, buildings, behavior, fire-water-stone-and-clouds, outdoor maps, indoor maps, bodies of water, ecosystems, road networks, urban environments, entity behavior, puzzles, storyboards, the story, levels, system design, world design, leaderboards, and news. They also define five categories of tools: pseudo-random number generators, generative grammars, image filtering, spatial algorithms (tiling, fractals, Voronoi diagrams), modeling of complex systems (cellular automata), and artificial intelligence (genetic algorithms, ANNs, constraint satisfaction), and provide examples of pairings of method and content, such as L-systems for trees. This style of framework provides a well-structured way to catalog the PCG applications of past games, and offer recommendations for the best approaches, such as using Perlin noise to generate cloud textures.

However, the possibility space of games is ever-expanding, and many game designs use content in novel ways (such as object design in *Katamari Damacy*, or the layout of trees and structures in The Witness). A Lindenmayer system, as a context-free grammar, has no concept of the outline of the tree that it generates, or its branches' locations relative to other objects, and so could not address the gameplay needs of a tree for Witness-like spatial puzzles. It is easy to think of many kinds of content not described by this ontology (social customs, pottery,

poetry, language, memes, recipes, crochet patterns) which might be of great importance to some future game. Each one of them might be part of a gameplay, a story, or the background of a game. While this framework classifies *common* uses, it cannot predict or help design *novel* uses. Additionally, though they list the broad category of method, often there are vast gaps between a method and a working algorithm, and it is easy to create a description with specification gaps: as exemplified by the statement that "[b]uilding floor plans can be procedurally generated using PRNG techniques". How to translate a number into a floor plan is a difficult puzzle, one which is not explained by this framework.

Togelius et al - "Search-based Procedural Content Generation"

In the framework of "Search-based Procedural Content Generation: A Taxonomy and Survey" by Togelius et al[411], the process after generation is separately viewed through three lenses: what kind of content is generated, how it is represented, and how the content can be evaluated. This framework intentionally does not address the methods used in creation, specifically how the representation of a piece of content (its genotype) can be translated into the finished artifact (the phenotype). Instead, they focus on the heuristic (how desirable or apt for its purpose), and how that can be used to iteratively re-run the generator to optimize results. They, like Hendrikx et al, list several kinds of content (trees, terrain, racetracks), but they decline to add storytelling to their taxonomy, as it has not yet had a search-based generative implementation. Though, while it necessarily catalogs only existing systems (and ones the authors are familiar with), in this case, unlike the Hendrikx classification, the steps to implement a search-based algorithm would be useful even for unknown applications.

Smith - "Understanding procedural content generation"

In Gillian Smith's "Understanding procedural content generation: a design-centric analysis of the role of PCG in games"[382], Smith divides procedural content generation into five categories:

- generation as optimization
- generation as constraint satisfaction
- generation with grammars
- generation as content selection
- generation as constructive process

The last category, constructive process, refers to generators that "build content in an ad-hoc manner by piecing together customized building blocks", but uses Rogue level-generation (a series of sequential operations on a grid of empty-or-solid cells) as an example, so it is unclear if the building blocks refer to steps in an algorithm, or actual pieces of content, in which case it may be more closely related to a grammar. The framework further breaks the parts of a generator into experiential chunks, templates, components, and subcomponents, human-authored sections of varying sizes of granularity and emergence. Unlike the other frameworks, this one specifically addresses how the player affects the generator: no-influence, setting parameters, selecting from generated content, or directly manipulating the artifact itself.

6.3.3 Frameworks of natural history

Thompson - "On Growth and Form", and Ball - "The Self-Made Tapestry"

D'Arcy Wentworth Thompson's massive 1917 tome of biological and natural algorithms[408] catalogs the range of generative forces at play in the natural kingdoms (bean patterns, beehives, horns, soap bubbles). Each natural phenomenon belongs to a set of similar phenomena (e.g: "the equiangular spiral"). Thompson speculates about it not as a single function, but as a product of both the general algorithm and a set of tuning values, wondering if "diverse and dissimilar fishes can be referred as a whole to identical functions of very different coordinate systems". For example, after discussing the forces that create spiral antlers in antelope, he observes the rhino's horn as the same general category, but operating in the unique conditions of being in the *center* of the animal, not as a matched set:

> "Its longitudinal growth proceeds with a maximum velocity anteriorly, and a minimum posteriorly; and the ratio of these velocities being constant, the horn curves into the form of a logarithmic spiral in the manner that we have already studied. The spiral is of small angle, but in the longer-horned species, such as the great white rhinoceros (Ceratorhinus), the spiral curvature is distinctly recognised. As the horn occupies a median position on the head — a position, that is to say, of symmetry in respect to the field of force on either side — there is no tendency towards a lateral twist, and the horn accordingly develops as a plane logarithmic spiral. When two median horns coexist, the hinder one is much the smaller of the two: which is as much as to say that the force, or rate, of growth diminishes as we pass backwards, just as it does within the limits of the single horn. And accordingly, while both horns have essentially the same shape, the spiral curvature is less manifest in the second one, by the mere reason of its shortness."

Not only can he see the relationship between the rhino's horn and an antelope's, but he can see the relationship between two horns as reflections of the **same behavior occurring within the horn itself.**

He recognizes that his natural framework "is but a portion of that wider Science of Form which deals with the forms assumed by matter under all aspects and conditions, and, in a still wider sense, with forms which are theoretically imaginable." Operating well before computers, he probably would not have predicted the mathematically generative art we have today, but generative artists have had notable success taking analytical and descriptive frameworks (Lindenmayer systems, grammars) and running them backwards to generate "natural" forms. I also take inspiration from Thompson's concept of "transformations", simple pressures placed on generative systems to transform one form to another. While one could imagine infinitely complex transformations, it is more productive to catalog **simple** transformations that we can understand and compare:

> "We might suppose that by the combined action of appropriate forces any material form could be transformed into any other: just as out of a "shapeless" mass of clay the potter or the sculptor models his artistic product; or just as we attribute to Nature herself the power to effect the gradual and successive transformation of the simple germ into the complex organism. But we need not let these considerations deter us from our method of comparison of related forms. We shall strictly limit ourselves to cases where the transformation necessary to effect a comparison shall be of a simple kind."

Philip Ball's "The Self-Made Tapestry" draws very direct inspiration from "Growth and Form":

> "I wish to show in particular that pattern and organized complexity of form need not arise from something as complicated as life, but can be created by simple physical laws."

Like Thompson, he looks at biological and geological phenomena, with more examples that Thompson had not had the advantage of, like reaction-diffusion, which was only unveiled in Alan Turing 1952 paper "The Chemical Basis of Morphogenesis". He more explicitly defines his examples into algorithmic categories:

"Patterns", "Bubbles", "Waves", "Bodies", "Branches", "Breakdowns", "Fluids", "Grains", "Communities", as well as outlining a few broad principles across the algorithms: "Symmetry breaking", "Non-equilibrium", "Competing forces", "Instabilities, thresholds and bifurcations", and "Increasing complexity with increasing driving". Each category reaches across scale and field of study; Voronoi diagrams describe both microscopic cells, soap bubbles, and galaxy clusters.

One reason that I appreciate Ball and Thompson's work is that it encourages analysis that breaks across the boundaries of traditional disciplines. We have astronomers and microbiologists, but there are no degree programs in "Voronoiology".

In the next section, I propose a framework to model generative production across disciplines. This method-first approach will describe generativity as a *pipeline of generative methods*. I also provide an ontology of generative methods, identifying each type of method as requiring a particular form of input and producing a particular form of output. I have found examples of generative casual creators in music, crafting, games, experimental literature, and advertising. I hope that by making an "Self-Made Tapestry"-style taxonomy of generative methods, it will be as easy to compare across our disciplines as it is to use Voronoi diagrams to study the relationship between cells and stars, and that within similar algorithms, I can see relationships as clearly as the relationship between a pair of rhino horns. Yet with the influence of Dorin's framework, this framework also pays attention to "flows of information", the data being passed through this pipeline, and how it transforms. As always, this dissertation is about casual creators, which are fundamentally *interactive* systems, so this framework also maintains a concern of *where* people can interact with the system.

Chapter 7

A generative framework of generativity

What is generativity? To the procedural generation community, it often means the method of constructing content for use in a game. Game-creators seek to generate content to fulfill many conflicting constraints: content that is novel yet not game-breakingly novel, content that may either fill out the background of a game, or radically change the gameplay. There are established tools that are of use in this pursuit. Many of us in this community are expert practitioners with Perlin noise, L-Trees, grammars, tile-based placements, or genetic algorithms. The particular constraints of game content may be unique to games, but these generative methods, and their issues and affordances, are shared with many fields[1].

Generative music, computational creativity, parametric architecture, and many other fields all use these same methods. And many successful generative systems are hybrids of multiple kinds of generative methods, chained together. Games like *Spore*[s215], *No Man's Sky*[s149], *Dwarf Fortress*[s58], and *SimCity*[s204] each have many

[1]Some of this chapter is adapted from my paper "A Generative Framework for Generativity", 2017.

intersecting pipelines of generativity, where the output of one system becomes the input of another. Some subsystems contain cellular automata (Dwarf Fortress and SimCity). Some use inverse kinematics (*Spore* and *No Man's Sky*), some simulate particles and agents in a field of forces (*Spore* terrain generation, *SimCity* transportation). Some games use Perlin noise to create terrain (*Dwarf Fortress* and *No Man's Sky*). And in both games and interactive art, humans interactors may be *part* of this pipeline!

This section describes a way of thinking about generativity as a **pipeline of data transformations**, constructed from methods that take one form of data, and return data that is annotated, transformed, compressed, or expanded. Although there are many frameworks of generativity (in games and elsewhere), this framework attempts to provide a system to design and describe generative pipelines in a way that captures how generativity is able to take input (random numbers, mouse input) and transform it into something new and surprising.

Shifting the focus from game-content (and how it is used) to the methods of generativity and how they apply across fields has allowed me to work in conversational agents, bot-making, interactive art, and generative textiles, often using the same tools developed for games (tracery.io). While this focus on generative methods has been very productive for me, I believe that there is also useful work in formalizing this approach, so that others, in games, art, generative text and other fields, may also use these methods. To this end, the GFG framework is the result of a year of community engagement: presenting this idea as a popular blog post [73], a standing-room-only talk at GDC 2017 [78] and a Creative Commons zine [75]. In the latest iteration, this framework has been developed into a deck of cards, which were tested as a tool to help students understand the design of generative systems (see the section at the end of this chapter).

This framework includes both "additive" and "subtractive" methods. Adam Smith[378] divides PCG techniques into two broad categories — additive and subtractive techniques, after the terms used in sculpture for building up a form, and carving away unnecessary parts. Additive methods *add* to a possibility space, and subtractive methods *remove* possibility. Togelius et al's[411] framework use slightly different terms, referring to "constructive" (PCG systems which construct content but does not evaluate heuristics) and "search-based" (removing individual substandard or incorrect solutions, but also finding the optimal solutions and discarding the rest) methods. In their original use by these authors, these terms refer to solutions that represent the output of an entire pipeline of generativity, but they are just as easily found as components in larger pipelines (i.e., "search based" or "subtractive" methods to find places to put cities in a procedural terrain generator). Various methods like these increase information (or variety or complexity), or they decrease it, or they transform the data in some way that does not clearly grow or shrink the data. So I do not use this terminology as a classification principle in the GFG, though it is useful in talking about the *size* or *range* of the resulting generative space, and to identify strategies to deal with a generative space of the wrong size, quality or expressive range.

One goal of the GFG framework is to separate these full systems into their component pieces, so I will use the term "generative methods" to describe a transformation that adds, subtracts, expands, or transforms information. Each generative method has input and output data types that defines what kind of information they can operate on, and what kind of information they return. These parameters define the *suitability* of particular categories and methods to different situations. For example, Markov chains and generative grammars are often confused by novice text-generation creators, understandably so since they both

output generative text. However one requires a corpus of existing data as input (Markov chain), and one requires hand-authored rules (grammar), so they will be useful in different situations. One could not pipe the contents of a twitter feed directly into a grammar-generator, like one could with a Markov chain, without also adding a generative method to first transform raw text to grammar rules.

7.1 Generative methods (components of a generative pipeline)

"Generative methods" as a term began with a request for me to give a guest lecture on "PCG" in Evan Merz's UCSC 2013 undergrad class on "Music and AI". As I tried to fill out a slide starting with "PCG is...", I realized that I couldn't draw a line around PCG and non-PCG. As an exercise, I made a deck of cards that had possible PCG or non-PCG systems on them: "Spore Creature Creator", "Rogue-like dungeon generation", "Perlin noise", "Sine waves", etc, and asked my labmates to sort them into piles. What we discovered (and Joe Osborn and I wrote into a paper[89]) was that "PCG" as a term referred to the *use* of generative systems (levels and content-generation), but that these systems were built out of many components with generative properties (sine waves and perlin noise).

As I further developed the idea, though projects like Generominos and creating the GFG framework, it was clear that these atomic components combine and recombine in a variety of generative pipelines. I call these atoms of generativity "generative methods". They can be tiny components that hardly stand on their own as "generativity" (if-statements, sine functions, random numbers) or complex clearly-generative algorithms like metaballs and Voronoi diagrams. But they almost all fall into a limited number of categories, and each category has central

properties that define how its methods can be used in a pipeline.

7.1.1 Content selection

Though not often considered "generativity" in itself, content selection is often a component in these pipelines of generativity. If an 20-sided dice is rolled, that action selects from 20 possibilities, as does drawing from a deck of 20 cards. A content selection method has some logic of selection, whether randomness, or randomness with changing probabilities (like cards), weighted randomness, or more advanced non-randomness based logic, such as sorting content based on some heuristic. Content selection may even represent a request for external data, such as asking for the last tweet with a particular hashtag, or the most popular newspaper headline. Content selection can be used *as a filter*, and as such, multiple content selections can be chained together, such as getting all images from a particular user's tweets, using machine learning to detect which ones contain cats, then sorting by "like" count, and selecting the top three, plus one random one.

Requires: Some collection of content or data, or a source of that content

May require: A heuristic for selecting content (can return true/false for inclusion/exclusion, or a ranking)

Returns: A single piece of content, or array of content

Potential for interaction: Humans can power many different heuristics, including aesthetic, personal, or strategic preference for one piece of content over another. Humans are famously bad random number generators, *and yet* we feel more emotionally connected to dice we've thrown ourselves, or tarot cards we've drawn, than to content selected *just as randomly* by a digital assistant

Casual creator (or interactive) examples: "Which Disney Princess are You" quizzes (select 1 of 4 options in several questions, then that is calculated into a

selected princess from the 8 available ones). Recipe dice[s185]. Apps where you have different choices for each "slot", such as "Barbie Magical Fashion"[s10] or "Create Your Own Unicorn"[s40].

7.1.2 Tiles

Perhaps the most "classic" form of generativity is tile-based generativity, in which available slots are filled with some selection of content. A common historical example of this is a tarot spread Figure 7.1 in which from 78 major and minor arcana cards, ten are drawn and placed in a "Celtic cross" pattern, and the choice of each card in each position may be interpreted into a personal meaning. Another historical example is the *Musikalisches Würfelspiel*[446], a tile-based music generator from the 1700s. Similarly, each cell in *Rogue* is a slot which can be empty, or full of a wall, monster, or object. Some tiles have constraints: a *Settlers of Catan*[s200] board must use all the tiles, but ocean and harbor tiles may only be placed in specific locations.

Often tiles are recursive: one tile may contain a template to fill with more tiles (i.e., equipping a socketable weapon in *Diablo II*[s48]) but this may be better modeled as a grammar.

Requires: A set of possible tiles, a template of slots or sockets

May require: Selection logic (see "content selection" above), constraints (which tiles can fill which sockets, orientation or proximity constraints), heuristics for discarding or "re-rolling" tile placement

Returns:the selected tile (or none) for each socket

Potential for interaction: Letting users select which tiles to slot into what sockets. Users select which tiles are available for placement. Users set some higher-level heuristic. Users set the layout of the tile sockets.

Casual creator (or interactive) examples: Tile layout in *Betrayal at the House on the Hill*[s12], which is mostly controlled by randomly drawing a room tile to fill a particular grid space, but the player can choose how to orient the tile. *Civilization V*[s34] let players set a variety of controls and heuristics (like "archipelago" or "continents", granularity, temperature, etc) when generating maps, granting players control even when the generation was autonomous.[2]. *Parable of the Polygons*[s162] and *Emoji Simulator*[s207] both let the user fill tiles (before the tiles layout is used as a cellular automata). *WolframTones*[s243] and *Google Song Maker*[s88] both use the approach of dividing a musical staff into a grid of beat squares. In *Google Song Maker*, the user may edit these directly, but in WolframTones they can only be controlled by heuristics.

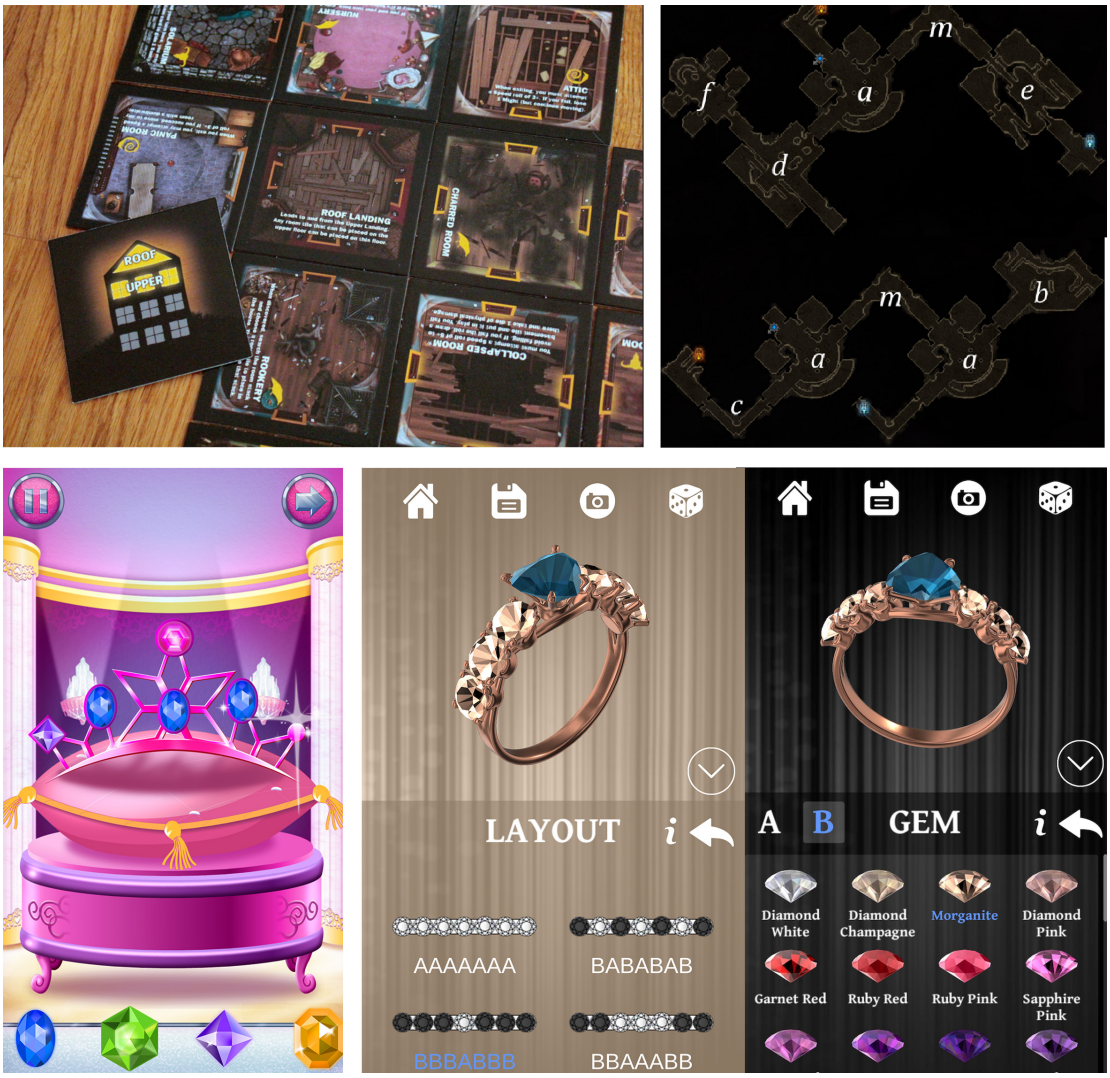


Figure 7.1: Tarot spreads (from Flickr user Atell Rohlandt). The tarot reader selects a spread type (content selection), then fills the spaces with cards (tiles), which are randomly selected from a deck (content selection).

7.1.3 Grammars

Grammars recursively replace non-terminal templates with either more non-terminals, or with terminal symbols. Sometimes these are purely context free, but real world examples often have some amount of context-sensitive features

[2] `https://gaming.stackexchange.com/questions/19389/civilization-v-map-generator`



.

Figure 7.2: (bottom)In *Ring Designer*[s190], the user selects a layout and band style (content selection). For each gem type, they select the size, style, and color of the gem (content selection), but these choices are used to fill in sockets in the ring's band in a fixed pattern (tiles, grammar). *Barbie Magic Fashion*[s10] uses a simplified version of the same mechanic decorate tiaras. (top) Constrained tile-placement in *Diablo II* and *Betrayal at the House on the Hill*`https://diablo.gamepedia.com/Map_Generation`, Flick user *meoplesmagazine*

(preventing repetition, for example). Commonly, grammars require a selection from multiple possible expansion rules, which can involve a content-selection sub-

method. *Tracery*[82][3] is an example of a system designed to handle text-generation with user-authored (mostly context-free) grammars. Other systems, such as Joris Dormans' *Zelda*-style level generators[124] operate on graphs by replacing particular kinds of node-edge patterns with more complicated (or gameplay-meaningful) subgraphs.

Tracery returns not only a tree of recursive choices, but also a flattened depth-first traversal of the finished text of each node. Grammars themselves *do not return finished content*, they still require some additional methods to compress that tree into the flattened content. A grammar may create, SVG text, for example, but that requires a browser to parse those parametric geometry instructions to draw an image on the screen.

Requires: A set of replacement rules (grammars)

May require: Non-context free rules that govern preferences or constraints (if so, some form of a constraint solver is required)

Returns: A tree of recursive choices, or the flattened version of the result of those choices

Potential for interaction: Any time content selection happens, whether in the selection of tiles in the previous section, or during the selection of grammar rules (for replacement rules with multiple matching option), users could affect the heuristic or replace the random selection mechanic. Users could provide expansion rules, or tuning values of those rules (for Lindenmayer system).

Casual creator (or interactive) examples: In *Mad Libs*[s128], users provide expansion rules for an extremely simple grammar. *Eliza*[s64] also has a template system of reminiscent of *Mad Libs*, where the computer therapist can detect particular kinds of phrases to harvest words from, then can reuse them in dialog with the user (an interestingly *indirect* form of *Mad Libs*). We don't yet see many sys-

[3] `tracery.io`

tems where users playfully create new rules (except in Tracery) but can imagine systems where there is some simple or playful way of providing examples that are used to construct new rules, such as systems that let users input raw text to create grammars or Markov chains. There is a potential for interaction in *choosing which parts of the grammar are expanded*, if the grammar is partially applied. Orteil's *Nested*[s147] could expand infinitely, but relies on the player to choose which components to expand, unpacking and repacking them until they find one that generates what they are looking for.

7.1.4 Parametric

A parametric generative method is one which converts some set of numerical values (a vector) to content (even if the content is another numerical value). This conversion may be as simple as turning a value into the sine of that value, or turning two values into the Perlin (or Worley) noise value at that point. But many unusual and custom functions can be created. The "superformula" [155] is an equation that can describe a striking range of 3- and 2- dimensional shapes using 6 parameters, and was considered for use in *No Man's Sky*[264][s149].

Lindenmayer systems[242] are very a common generator type (almost a cliche in tree generators); any taxonomy cannot neglect to include them. Interestingly, an L-system is a hybrid: a computational grammar, with parametric values defining the parameters of each grammar replacement. This can be seen in a recent platformer game, *How to Be a Tree*, where the player manipulates the parameters of a Lindenmayer tree to make it crawl and jump in a physics simulation[4].

Parametric methods are, in fact, often less of a category of generative method, and more like an interface which they may implement. In chapter 11, we discuss

[4] `https://twitter.com/aZaphos/status/1092209422430916608`

some properties of certain parametric methods which make them particularly expressive for generativity and interaction. In "Example systems", several simple parametric generators are dissected to reveal that a parametric generator may be part of an L-system or a particle system. However, all forms of parametric generator *take an array or set of numerical data as input*, regardless of what they output.

The translation from the vector to the output content is a critical part of parametric methods. Parametric translations that are discontinuous or low-locality (small changes in input produce large changes or discontinuous jumps in output) are common and easy to make accidentally. Commonly programming practices like if-statements, thresholds, selectors (like the dice-roll/loot tables in *Dungeons and Dragons*[s57]), or any integer math may cause discontinuity. While it is seductively easy to have one parametric value determine the *type* of content or *number* of some feature (i.e. if the value is less than .5 make a flower, otherwise make a tree), this can limit the ways that parametric methods can be used. A fully continuous method can translate changes in input, changing smoothly over time, to smooth changes in output content, which can create attractive animations or smooth transitions from one output to another. Methods with high locality can also be used in search-methods, including evolutionary algorithms.

Requires: An array or set of numerical data (often normalized)

Returns: Some form of content constructed from that data

Potential for interaction: As we will investigate in chapter 11, parametric methods have many possible interaction paradigms. At their heart, they are a simple array of numbers, so these numbers can come from a variety of input devices, like mice, EEG headsets, pressure sensors, music frequency analysis, or hand/-face/skeleton tracking data. A set of sliders (one per dimension of the vector) are

the most common parametric input, but 2D sliders (or even 3D positions) can be used to set multiple values at a time. Many forms of data can be transformed into vectors (they are a common output data type in generative methods), so interaction can produce data that, through other generative methods, produces a vector. For example, the user could produce words from some source (ie, speaking, typing, or selecting a text-based content source), which could then be translated with Word2Vec into a series of vectors and used to make generative flowers based on those words.

Casual creator (or interactive) examples: Any casual creator with a color-picker (usually a 2D slider for hue/value and a separate saturation slider, for a 3D parametric value). My own *Funky Ikebana*[s76], as discussed later in chapter 11, *Palettsy*[5], and *BoxCar2D*[6] use vector-based data with user or heuristic-selected genetic algorithms for control. *Petalz*[s167], *Galactic Arms Race*[s81], and *PicBreeder*[s169], all have some form of parametric representation, controlling the blending of some combinational tree of functions (compositional pattern-producing neural networks, or CPNN), evolved according to variety of human-directed heuristic interactions.

.

7.1.5 Geometric extrapolations

Geometric extrapolation a class of functions, that, unlike parametric methods which take numbers, take some other form of geometric information and process it into a different kind of geometric information.

One familiar form of this is a Voronoi diagram[138]. This computation takes a set of points, and returns the region graph (a set of edges and nodes) that

[5] `https://sarahgould.itch.io/palettsy`
[6] `http://boxcar2d.com/derpBikeDesigner.html`

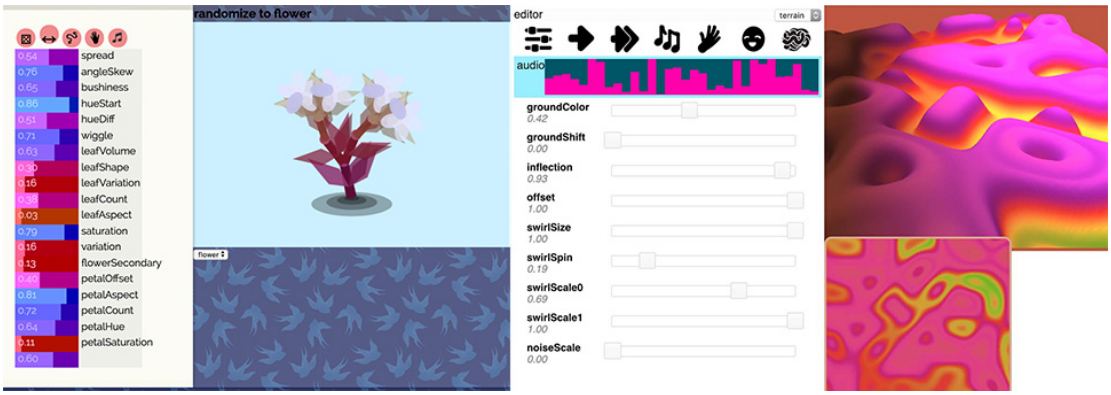


Figure 7.3: Two revisions of an array-of-floats tutorial and platform I provided for users to play with or modify. This is part of a long-running[72] set of tutorials and code I have developed to help developers better understand parametric methods, which is one of the most approachable, powerful, and easy-to-get-wrong generative methods categories. (`http://www.galaxykate.com/apps/aof/`, `https://github.com/galaxykate/slidertoy`)

represents the region of space closest to each point. A Delaunay triangulation, which can be constructed from a Voronoi diagram, also takes a set of points, but returns a triangulation of those points. Many other specialty triangulation algorithms exist for various needs and dimensionalities.

Some methods take 3-dimensional points and turn them into triangle meshes, often in a generative and emergent way. Metaballs[363] represent an implicit surface at some radius away from a set of points, and can be calculated with the "Marching Cubes" algorithm to construct a triangle mesh, which was used in Spore[s216]. Likewise, a region of arbitrary voxel data can be turned into a triangle mesh with voxel polygonization algorithms, a key step in *No Man's Sky*'s generation of renderable terrain[264][s149].

There are even some simple, often overlooked generative methods by which curves are turned into more complex curves, or geometry. Offsetting a Bezier curve is one such method. This seems too simple to be worth noting, but being able to remap geometry along a curve is the foundation of many digital painting

tools. Painting tools work as either deformed geometry or shapes along a curve, or distributions of images or geometry along a curve [381]. This is visible in Adobe Illustrator brush strokes, in Figure 7.5, and is also used in the physically impossible generative 3D paintbrushes of Google's *Tilt Brush*[s235] VR program.

There are too many different forms of geometric extrapolation to list all of them (this category is a bit of a catch-all for a variety of individual methods). But here are the specifications for two:

Voronoi diagram

Requires: A set of points, in 2 or 3 dimensions

Returns: A node-edge graph of the regions around them. Can also trivially return the Delaunay paths between points.

Potential for interaction: Many body-tracking inputs return a node-edge graph or array of vectors. Voronoi diagrams are

Casual creator (or interactive) examples: *Idle Hands*[s97] is my quickly-created interpretation of hand-tracking data from the Leap Motion into attractively-colored triangles. It experienced mild success at art-festivals, with a strong feeling of direct manipulation, due to the simplicity of the clear mapping between hand position, input points, and output graph (which also made it very simple to code). Stjärna is a Swedish paper-lantern generator that I made, also using Voronoi diagram on source points to create regions that tiled a lantern side, then cutting a virtual hole in each region to create a perforated pattern. In this case, the points are *not* from body tracking, but can be dragged around to change the resulting graph, and the shadow pattern created by the lamp. The points can *also* be generated by a curve, in a geometric extrapolation (curve to points) that is then fed into *another* geometric extrapolation.

Curve extrapolation

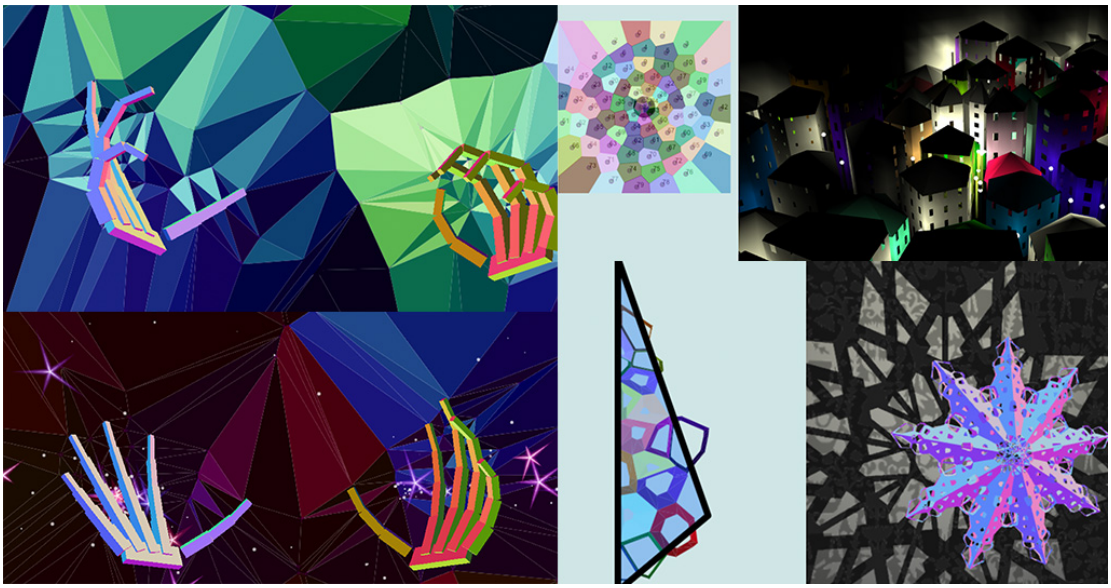


Figure 7.4: My projects, *Idle Hands*[s97] and *Stjärna*[s220], which both use Voronoi diagrams. *Stjärna* uses them to create generative paper lanterns *and* a small Swedish town to install them in.

Requires: A curve, whether extrapolated from individual points or defined more sparsely by Bezier points, and a brush or pattern or profile to extrude along that curve

Returns: Geometry, vector-graphics, or pixel data resulting from that extrapolation

Potential for interaction: Curves are a common output of gesture tracking over time, whether from points drawn by a mouse, a finger on a touchscreen, or a hands position tracked in 3D space. While it is also possible to interact with this by editing the brush or pattern used on the curve after the curve is created, this is more common in "serious" tools like Illustrator. Casual creators usually select the brush pattern first, then create the curve.

Casual creator (or interactive) examples: *I/O Brush*[s104][337] was a prototype of a creativity system at MIT's Media Lab. Users held a paintbrush with a camera in the tip and 2D tracking capacilities. They could "pick up" colors and

animations from the world, then "paint" those onto a screen along the curve they drew with the brush. In *Tiltbrush*[s235], users can draw in 3D with positionally-tracked hand controllers. They can choose a variety of brush styles (content selection) and colors/sizes (parametric), which are then extruded as 3d geometry behind the brush's path. In my own *Jazz Composition*[s105], a user with a Leap-motion or mouse draws curves on the screen. Each curve has a style randomly selected for it, usually with multiple offset curves and interpolated points. The curves "come to life" (as particle simulations) with rhythmic wiggles.



Figure 7.5: Brushes I made in *Adobe Illustrator*, and their deformation/application along a curved path

7.1.6 Distribution

A sufficiently sparse tile-based system (very few tiles to place, many possible sockets) can often be treated like a distribution. There are many forms of logic that can control a distribution, such as suitability, but may also include aesthetic logic such as:

- greebling: a term originating in Star Wars production history [78], meaning that the uneven distribution of object over a surface makes the surface look

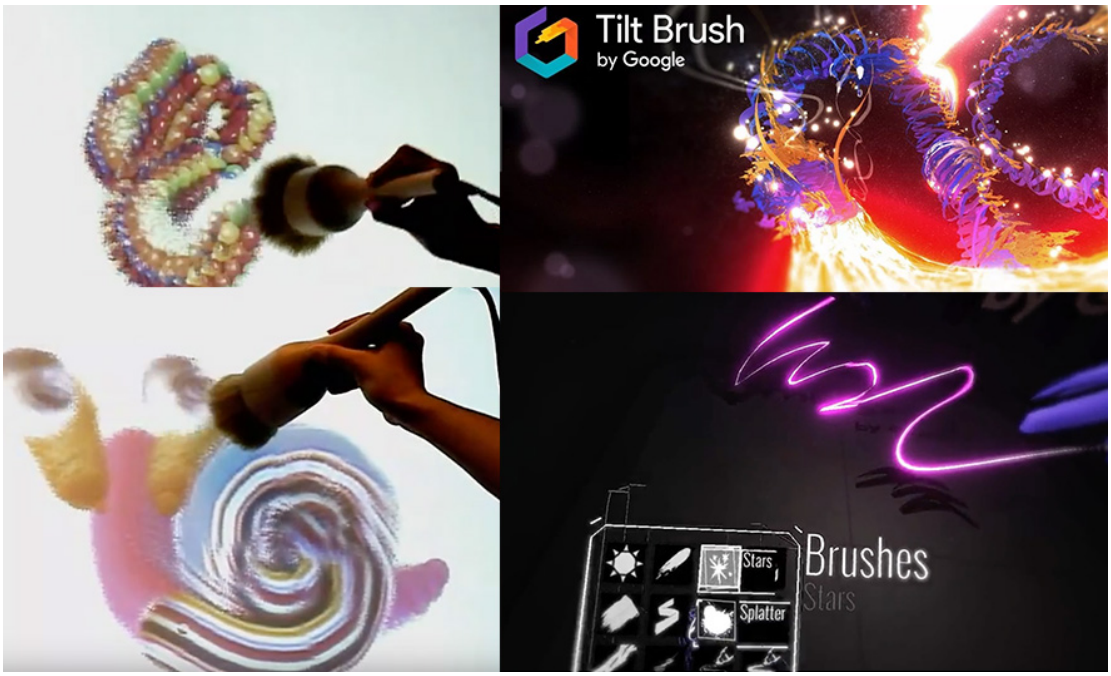


Figure 7.6: *I/O Brush* (left) and Google's *Tiltbrush* (right). Both tools use extrapolation along a curve, with different types of generative/emergent pattern. In IOBrush's case, the pattern is sampled with a webcam (content selection). For *TiltBrush*, the patterns are textures or geometry provided by the system, but are animated or decorated with particles and glow effects

"real"

- footing: the tendency for the intersection between two naturally-occurring objects to be different than non-intersection spaces, such as grass growing higher against a fence post, moss accumulating the the cracks of boulders, trash getting caught against a park bench, or sand pooling where a river bank meets the river.

- barnacling: a term-of-art on Spore used to describe the aesthetic value that a monumental object is often best framed by smaller simpler versions of itself, or by some contrasting object.

These aesthetic choices may suggest that distribution occurs only in landscapes, but is useful as well in generative art, music, decoration or fabric design.

Requires: Content to place

Often requires: Constraints on the space, information about the content/space that controls the placement

Returns: A list of the selected content and its placement

Potential for interaction: Like the content-selection or tile-based methods, this can have heuristic or direct control by users. This algorithm works well with heatmaps (which can control density or other placement choices) so can be used with aggregate data (like Google maps travel data)

Casual creator (or interactive) examples: Though not a casual creator, Pokemon GO[s176] uses Google's pedestrian traffic data to control placement of collectable Pokemon and activity centers, as well as using geographic data to control the type of Pokemon (water Pokemon near bodies of water, e.g). Several casual creators use *direct* control over placement, such as the placement of decals on clothing in *DIY Fashion Star*[s52] or placing parts on creatures in the *Spore Creature Creator*[s216].



Figure 7.7: Decals in *DIY Fashion Star* can be selected from the user's photos or pre-created assets(content selection), then placed, scaled, and rotated on the clothing

7.1.7 "Agent" simulation

Several different kinds of generative methods fall under the general category of "agent" simulations (or "a-life"). I'm not particularly fond of the term; as we will see in several examples, the difference between a smart "creature" and a mindless physical particle is a spectrum, and plenty of complex behavior does not feel sentient or biological. However, each of these different systems (regardless of its relation to the "a-life" term) has a slightly different set of properties, and can be used in different ways.

Cellular automata:

Cellular automata are grid-based cellular systems where each cell represents an agent with rules for updating based on the properties of its local neighbors. The origin of the concept is in *Conway's Game of Life*[s78][147] but it is also used in *Dwarf Fortress*[s58], *Minecraft*[s136], and in the pollution and water models of *SimCity*[436]. It's also the basis of a platform for running emergent simulations to make arguments, *Simulating The World (In Emoji)*[s207] by Nicky Case[51], and an interactive essay, *Parable of the Polygons* (with Vi Hart)[53][s162], based on a 1971 experiment about segregation[347].

Requires: A grid of cells, starting values for the cells, and rules for the cells

Returns: The next state (or a streaming sequence of next states)

Potential for interaction: Initial placement seems to be the most common interaction type, though this may be because many interactive cellular automata are just interactive versions of *Conway's Game of Life*[s78], which has very limited flexibility in tuning (changing any of the values of the rules to other integers breaks the game[7]). "Parable of the Polygons" and "Simulating The World (In Emoji)" also encourage the player to tune or create rules.

[7] There are interesting experiments in making stochastic non-integer Game of Life variants

Casual creator (or interactive) examples: *The Powder Game*[s178] is a decade-long project of making a cellular automata with many emergent properties. It has resulted in many user-created levels with unusual and creative configurations of its elemental materials (sand, laser, vine, powder, magma, water, etc), each of which obeys simple sets of rules.[94] A very similar method powers the mobile game *The Sandbox:Craft, Play, Share*[s194].

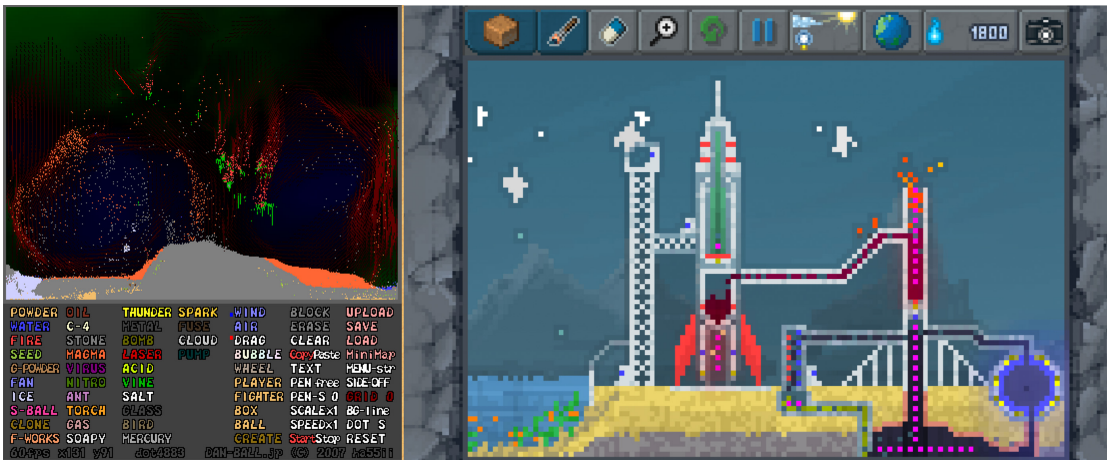


Figure 7.8: *The Powder Game*, with assorted user-made levels that simulate volcanoes, engines, and electrical systems (`https://dan-ball.jp/en/javagame/dust/`), and *The Sandbox:Craft, Play, Share* commercialized version of the same gameplay on Android

Particles:

When agents are liberated from the grid and can move freely through space, they become particles. A particle has physical properties like velocity and acceleration, and may also have other properties that define how they respond to forces. Almost every particle system has some forces being applied to the particles, such as gravity or drag, but often those forces represent non-physical forces, such as flock cohesion and avoidance in the Boids algorithm [316]. Applying complex forces to particles can generate interesting patterns that respond to external geometry. Particles were used to texture the planets and creatures in Spore: as

particles moved across the geometry, they acted as paintbrushes, painting into the texture map, using distribution of stamps or deformable brush geometry extruded along their paths [81].

Requires: Starting positions for the particles, any forces or colliders that can interact with the particles

Returns: Subsequent positions for particles, possibly also collision data

Potential for interaction: One form of interactive particle system is a pinball machine. The ball is affected by forces not controlled by the user (magnets and gravity), as well as forces that are (tilt and flippers). The ball's behavior is emergently determined by its interaction with these forces, some of which are directly or indirectly in control of the player. So for this category, we can consider any of the many ways that people could drive forces, and use that for interaction.

Casual creator (or interactive) examples: Collision behavior in particles can be controlled with any hard shape. Text Rain[s232] is an early body-controlled digital artwork where users' shadows interact with particle-based "rain" made of letters. The falling words break apart and slide across the edge of hands and shoulders, creating surprising yet controllable behavior. Force-based particle behavior, compared to collision behavior, is softer and less direct. In this form of behavior, the user's body acts as a magnet or gravity well (or other fantastical physical force) for particles. The particles don't need to intersect or collide with the body, but they follow it in a swooping orbit. This works well for dance or other gesture where you want to emphasize the smoothness of movement. I use this technique frequently, in *PaintThrush*[s160], and *Monster Mirror*[s138][87], *Teuhana*[85]. The *LuminAI* project also uses it to decorate their virtual dancer[s127][244].

Graph-based simulations:

While cellular automata are confined to a grid, and particles can move freely

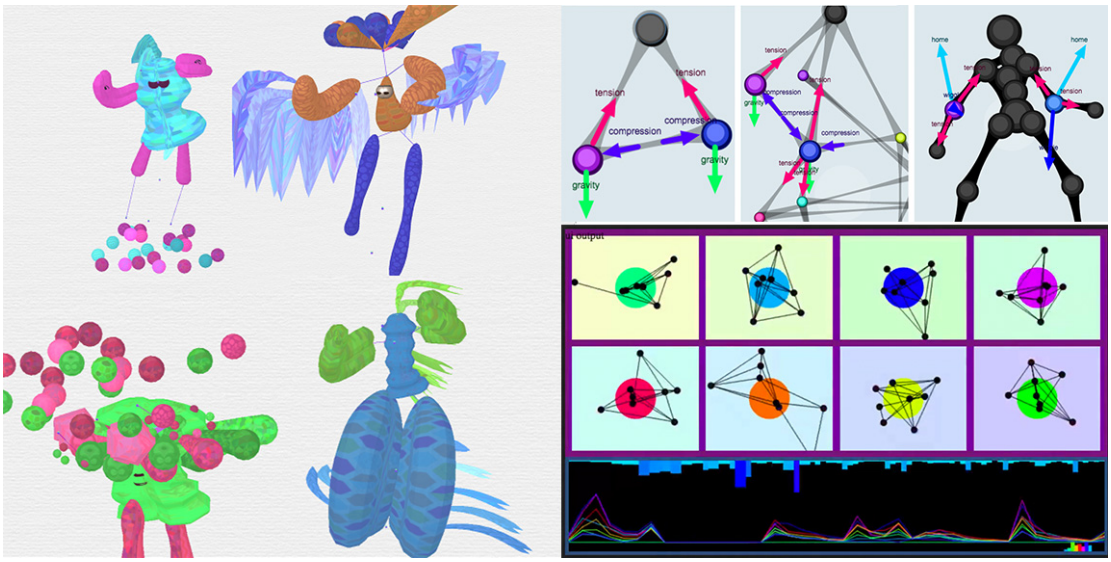


Figure 7.9: I often use particle physics with spring-forces in my prototypes, rather than inverse kinematics, for softer and more expressive generative animation. *Monster Mirror* (left) pulls particle-based costumes around according to motion-tracked skeleton data. *Teuhana* (right) tugs at the limbs of generative particle-based dolls around in time to music to create virtual dances.

though space, some agents are confined to paths on a directed graph, or even simulated as continuous flow through that graph. The difference between flow and agent in a directed graph is ambiguous: *SimCity*'s transport system implemented both with the same system[436]: humans, cars, water, and electricity (as well as invisible game data like "request a fire truck" or "jobs available") were all represented as units or flows moving along streets, sewers, sidewalks, or electrical lines[8]. *Loopy*[49][9]Figure 7.10, by Nicky Case, is a platform for designing emergent simulations based on direct graph flows, such as measuring the pleasure/productivity feedback loop of spending time with friends or work. Case also makes other exlorable explanations (and casual creators) with graphs, like the Hebbian-learning neuron simulator *Neurotic Neurons*[52][10] and the crowd-psychology sand-

[8]Yes, we had bugs where pedestrians would route along power lines
[9]`http://ncase.me/loopy/`
[10]`http://ncase.me/crowds/`

box *The Wisdom and/or Madness of Crowds*[50][11]. Similarly, some game systems can be modeled as a directed graph of resource movement, as Joris Dormans has built in *Machinations*[125][12]. These graphs can model many types of resource-management games, and predict their emergent properties. Sometimes multiple simulations in each of these styles will interact with each other. In *SimCity*, the distribution of new houses along a graph of streets is driven by the values of cellular automata maps, each of which has its own rules for how to update (groundwater pollution diffuses downhill). Those houses create Sims, which route themselves as particles or along the streets, as the houses siphon energy off an energy graph, and write into the pollution map.

Requires: A directed graph with update rules, or forces and distribution(particles), or cell update rules (cellular automata)

Returns: Particle trails, or the new states of the graph or grid-cells

Potential for interaction: These graphs work very similarly to cellular automata. Users may set starting values, or connections and rules.

Casual creator (or interactive) examples: In both Loopy and Machinations, users are supposed to model systems by creating connections through which resources can flow. Click-drag is used to connect two nodes, and there is a way to edit and deleted connections, as well as to edit their resource-flow properties. These systems are fun to watch, as their values slosh around and accumulate.

7.1.8 Machine-learned statistical models

So far, all the rules discussed for various systems (grammars, graph simulations) have been hand-authored. But there are systems which *generate* rules, and those, too, can be part of the generative pipeline. Markov chains and grammars

[11] `http://ncase.me/neurons/`
[12] `https://machinations.io/`

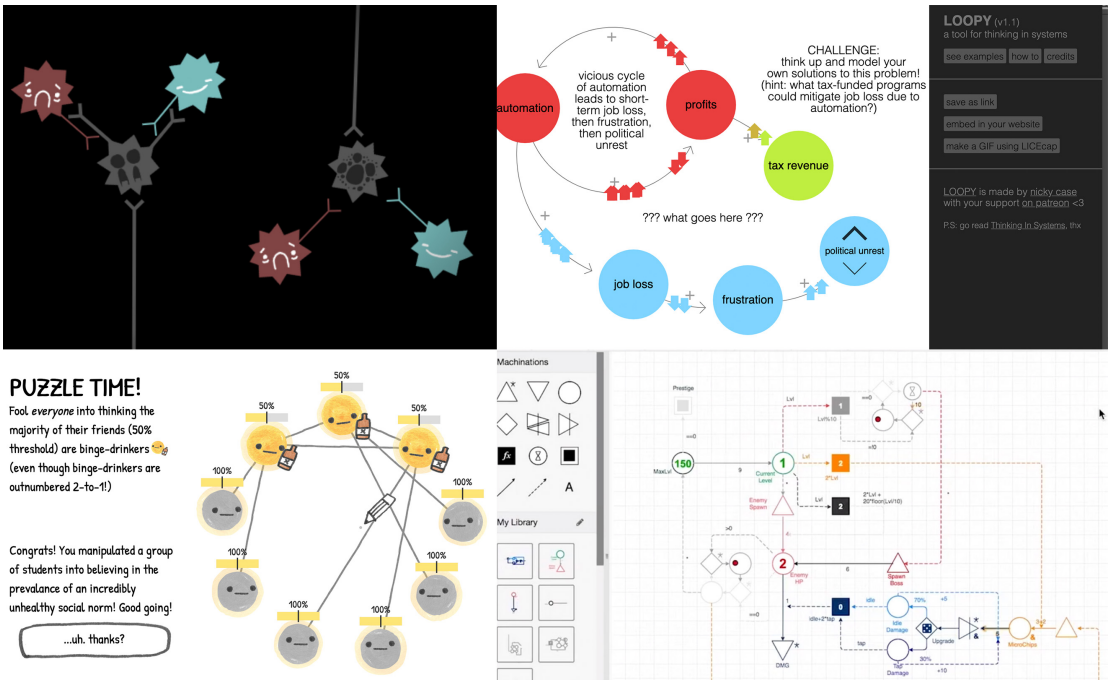


Figure 7.10: *Loopy*, *The Wisdom and/or Madness of Crowds*, *Neurotic Neurons* and *Machinations*, all of which use graph-based simulations, expressing wildly different phenomena.

can create similar-looking output, but a Markov chain (at least, a machine-learned model) requires an input of content to learn its statistical rules. *Deep Forger*[13][55] is a bot that has trained on classical artworks until it has generated a flexible statistical model of what colors go where in a Picasso. When given an image from a user on Twitter, the model can be used to iteratively optimize that image towards its ideal of Picasso-ness, and returned to the user (although I say "the model", all the data transfer to and from Twitter is done manually by the human Alex Champandard).

The PCG community is still just getting started using deep-learning to generate content, and are often unsure of where it will fit in a workflow. However, with this framework, there is a clear input, and a clear output, and surprisingly,

[13] `https://twitter.com/deepforger?lang=en`

"machine learning" is actually *two* components, which we can use separately.

Model construction:

Requires: lots of content

Returns: a model representing a set of statistical patterns in that content

Potential for interaction: it is very rare for a machine-learning algorithm to have tuning values exposed or dynamically modifiable, in a casual environment, so interactivity is usually limited to providing different training sets

Casual creator (or interactive) examples: Fortunately, users seem to get *very* creative with providing training sets. *Teachable Machine*[14s228] is a Google Creative Lab experiment to show how users how deep learning algorithms "train" on data. Users provide some webcam data, with one of three colors selected, to train the network that, for example, showing a banana on screen, or sticking out your tongue, should classify as "purple" and activate one of several actions (operating as a content selection mechanic). Deep neural nets are well suited to this, as they can handle a variety of user-provided content streams, but approaches like Wave Function Collapse [215] and the reverse Tracery grammar-generator *Cheap Markovs, Traced Quick!*[121], can take well-formed human artifacts and turn them into statistical recipes as well.

Transformation via model:

Requires:a trained model, deep-learning activation-optimization may also require a starting piece of content to optimize

Returns:a statistically-viable walk though the Markov chain, an activation-optimized version of the source content, a classification using that algorithm, etc.

Potential for interaction: Like with model construction, usually interaction is limited to providing different input data, or selecting a different model

Casual creator (or interactive) examples: Google, with its collection

[14]`https://experiments.withgoogle.com/teachable-machine`

of many, many well-trained deep-learning models, has been releasing a variety of casual creator (or at least, playful) apps to let people interpret their input through the models. *Move Mirror*[15s140] takes webcam data, turns it into a skeleton/graph, and matches that with the closest photo to that pose (content selection!), creating a collaged version of your webcam footage. The *Giorgio Cam*[16s85] uses a model to detect known objects in a photo, then slots those nouns (and reliability rating) into a templated grammar of bad rhymes. The *Quickdraw* project[17s182] challenged users to draw particular nouns, with the computer attempting to guess what they were drawing, a la Pictionary. This drawing data was released publicly, so other users quickly trained their own networks on the human drawings, resulting in apps like Monica Dinculescu's *Magic Sketchpad* [18], which attempts to take user's provided curve data, and elaborate on it according to past sketch data (a very AI-powered version of curve extrapolation).

7.2 Consumers of generative pipelines

Generative methods take in an input, and transform it into some output, and can be combined into a generative pipeline. What happens if their input *is* a generative method? Each of these **meta-methods** takes a complete generative pipeline (some input to some artifact or content) **as one of its input parameters**. Thus we can see generative pipelines built out of generative pipelines.

[15] `https://experiments.withgoogle.com/move-mirror`
[16] `https://experiments.withgoogle.com/giorgio-cam`
[17] `https://experiments.withgoogle.com/quick-draw`
[18] `https://magic-sketchpad.glitch.me/`

7.2.1 Navigation Methods

Each of the methods above creates some possibility space. For purely parametric methods, that space can be modeled as an N-dimensional cube, in which each point (a vector of N values) is a valid example of that space. Depending on how those N values are used to generate a piece of content, the possibility space may possess (or lack) several useful properties. The function that turns a vector into an artifact may possess "locality": two points near each other generate similar artifacts. If the functions of the parametric generator is continuous, then that space is continuous: the two similar artifacts will become more similar as the two points move towards each other with no discontinuities where two very-close vectors create dissimilar artifacts.

Such a space enables many useful techniques, such as creating a series of points interpolating between two points, which, viewed as frames in an animation, will show one artifact smoothly morphing into another. Similarly, an N-dimensional curve, drawn meandering through the space, with artifacts sampled along the curve, can create an animated tour of the space. Distance in the space can represent aesthetic distance between artifacts, so sampling artifacts at points evenly distributed through the space will give a set of representative yet different artifacts.

7.2.2 Search-based techniques

There are many approaches to navigating a space that has locality and continuity. By adding another dimension to the vector, a **heuristic value**, we can model that each vector in the space has a quality dimension, computed somehow from the artifact generated from that vector. A hill-climbing algorithm can take a given vector (representing some artifact), and sample the space nearby, moving

from lower quality artifact to a higher quality one, repeating the process until it has achieved a local maximum. This method *can* work with *discontinuous* space, but needs *some* amount of locality to work. "Breeding offspring" is only an optimization method if the offspring have similar properties to the parents, and are not a complete phenotypic reboot.

Starting from many different starting spots can increase the chance of finding more maxima (not just the local one). Other approaches such as modeling inertia (particle swarm optimization) or interpolations between two points ("crossover" in genetic algorithms) can improve speed or performance in some situations. If the space is truly continuous (rare but possible), gradient descent can be used to predict the fastest route to improvement.

Unfortunately, in practical generativity, very few spaces are purely parametric, and many spaces are not continuous or not very local. Often to model a possibility space, there are conditionals, selections (choose one of several options), or integer values, all of which introduce discontinuity into the possibility space, even if the space is still modelable by parametric values. For example, many parts of a Spore creature have parametric values: each "rigblock" part on a creature had between 1 and 3 values for shape adjustment, as well as rotation, scale, and placement along a limb. However, any number of rigblocks could be placed on each limb segment, each limb segment could branch into many more limb segments, and a body could have arbitrarily many limb segments. A Spore creature could be modeled by an array of scale-and-position vertebra, plus tree of limb joints (with scale and position), each containing an array of rigblocks (selected from a 60+ rigblock set), each of which itself is modeled as a 6-dimensional vector of settings. Thus, while a rigblock could animate through its possible configurations (in a nicely continuous possibility space), the space of creatures is non-continuous and

not modellable spatially.[19] Interpolating between two creatures, as Petalz[322] does with flowers, would not be advisable.

However, even in systems that are not fully *mathematically* continuous or local, these techniques *can* still work, just not as smoothly or well. In *Funky Ikebana*[20], the flowers have a 26-dimensional parametric "genotype", which generates Lindenmayer system flower in a *mostly* continuous function, but the number of leaves and petals are integers. Thus, when the flowers animate from one form to another, there is some popping as leaves and petals appear, but the animation feels mostly continuous, and users still can navigate the space continuously. In systems that must be modeled as trees of choices, such as the classic genetic programming example of building an expression tree to model a function, there is no possibility to create a continuous and local space. And yet, with a large enough tree, the effect of choices at the leaves are smaller, and it begins to behave *more* like a continuous and local system.

7.2.3 Constraint solvers

For certain classes of well-defined problems, there exist computational solvers. The solvers take in problem specifications in their particular format, and compute (in reasonable time) solutions to those problems. Two examples of this pattern are **IK-solvers** and **Answer-set programming**.

Inverse kinematics (IK) is a method of describing and solving the positioning of intermediate joints of an inflexible skeleton when an end point is moved to some absolute location: for example, determining where your elbow would be comfortably positioned if you touch your nose. A non-cyclical graph represents the constraints of a moving body (whether human, animal, alien or robot). Each joint

[19] We did prototype this on *Spore*. The interpolated creatures were a horrorshow.
[20] `http://www.galaxykate.com/apps/ltrees/`

in the chain has a range of motion, optionally with preferred ranges in addition to absolute ranges (there are positions for your elbow which are mechanically possible yet uncomfortable and unlikely). A variety of optimization techniques and approximations converts that information, plus the target location for the end points, into the optimal positions for the intermediate joints.

IK solvers are common in games for any kind of procedural animation, but perhaps reached the zenith of expressiveness in the development of *Spore*, where they were the final part of a complex chain of generativity, through which animator-authored animations were de-coupled from the sample creature they were animated on, to become generalized sets of "goals", which became IK-specification input for creature skeletons with arbitrarily many-jointed, twisted, and branching morphologies:

"The resulting pose goals preserve the overall motion and stylistic details of the authored animation. Stylized locomotion is synthesized for the player-created leg morphology and layered onto the goals. The goals are fed into an inverse kinematics (IK) solver tuned to handle conflicting objectives while attaining natural solution poses."[188]

A less-commonly-seen solver is Answer Set Programming (ASP), a recent descendent of Prolog-style logic programming. In ASP, sets of constraints and possible options are grounded into a boolean satisfiability problem, which is then used to generate sets of final values for which all the constraints are satisfied. This method allows an author to describe a generative world as a *space of possible choices that can be made*, while also describing the rules for how different choices impact each other (for example, a tile in a Roguelike dungeon cannot contain both a wall and an object). This technique has been used in a small set of indie games[378], as it can handle complex gameplay and design constraints, without

having to code them implicitly in the generative pipeline. *Storyteller*[s22121], is a rare example of a game/casual creator with constraint solving, which it uses to calculate possible story results for different groupings of tiny cartoon characters in a comic panel.[22]

7.3 Example generative systems

Certainly, this list of generative method categories is not exhaustive. There are likely uncategorized methods, and it is to be hoped that many will someday be invented that we have not anticipated and need whole new categories. But even with this set, it becomes possible to analyze a few example systems as generative pipelines.

7.3.1 StippleGen

StippleGen [294] is an example of a generative pipeline for non-game content that is used to create real-world objects (which will be explored further in chapter 23). It was created by Evil Mad Science Labs for their Eggbot, a CNC-controlled plotter and axle that can spin an egg to draw on it with a Sharpie marker, as a way to convert user-provided images into a format that could be sent to the plotter's hardware, This content had constraints: it needed to take an image, and return a series of dots that could be drawn with a single CNC-controlled marker (on an egg), and still be recognizable as the original image. This is the pipeline they created

- The user uploads an image

[21] `https://www.kongregate.com/games/danielben/storyteller`
[22] It is still in development years later. Custom constraint-solvers are a nasty loss of time.

- The image is converted a greyscale image
- The greyscale image is used to control a random distribution of many dots (more, larger dots for dark areas). This distribution is not as aesthetic as desired: random distributions create a lot of overlap
- A voronoi diagram is created using all the randomly-distributed points
- The system uses simulated-easing (a particle-based force simulation) to re-distribute the points more evenly
- The new point locations are used to guide the CNC-controlled marker to draw on an egg

7.3.2 No Man's Sky and Spore

This framework can be used to understand simple generators, but what about complex ones? As one of the main contributors to the Spore terrain generation system, here is the pipeline which created the Spore terrain:

- The system selects a set of particle types (based on scripts and pairing guides I wrote) which it emits at various points on the spherical planet (**content selection**).
- When they collide with the planet, the particles are simulated according to a variety of forces (**particles**).
- As the particles move, they form paths along which a brush-texture is either extruded/deformed (**curve extrapolation**) or dotted at points along the path or randomly over the planet (**distribution**).

- Greyscale images of these brushes are stamped along the path and added or max-valued into the terrain height map, itself just a giant array of height values (**parametric**)

- An artist-authored color ramp is selected for the planet (**content selection**) and the height map is interpreted into a terrain color map and 3D mesh based on height values (**parametric**)

- Creatures, rocks, and flora are distributed according to slope, altitude and some basic distribution rules (**distribution**).

Without going into complete detail, here is one large part of the *No Mans Sky* terrain generation pipeline [264]:

- The ground is modeled as a voxel field, a thin voxel layer around an empty sphere (**geometric extrapolation**)

- Various generative methods create terrain interest, such as using thresholded Worley and Perlin noise to add and remove voxels, which creates canyons, caves, tunnels and "Perlin worms" (**geometric extrapolation** based on **parametric** Perlin data)

- Turbulence is applied, a rough physical simulation of particle movement in a force field (**geometric extrapolation** based on **particle simulation**)

- The voxels are polygonized

- The polygons are deformed to match the sphere's surface (even more **geometric extrapolation**)

- Various parametric objects are generated and distributed on the surface of the landscape (**distribution**)

Like many shipped systems, both terrain generation pipelines are a collage of many different layered techniques. They both take a spherical planet and create landscapes on it, but differ in their entire approach until the distribution stage. We can also examine the pipelines to identify potential issues. One major issue with Spore was that the particle system had no way to be certain that it was covering the whole planet. Perlin noise deformation, on the other hand, is a parametric function run across *all the pixels* of the planet, ensuring total coverage.

We also see the complexity of these pipelines, and their lack of allegiance to a single technique, as many academic projects do. In the words of the engineer, Innes McKendrick, "Nothing by itself is really interesting enough, the key is having this toolkit of loads different techniques that you can use in different areas until you get something that looks like the goal you're seeking."

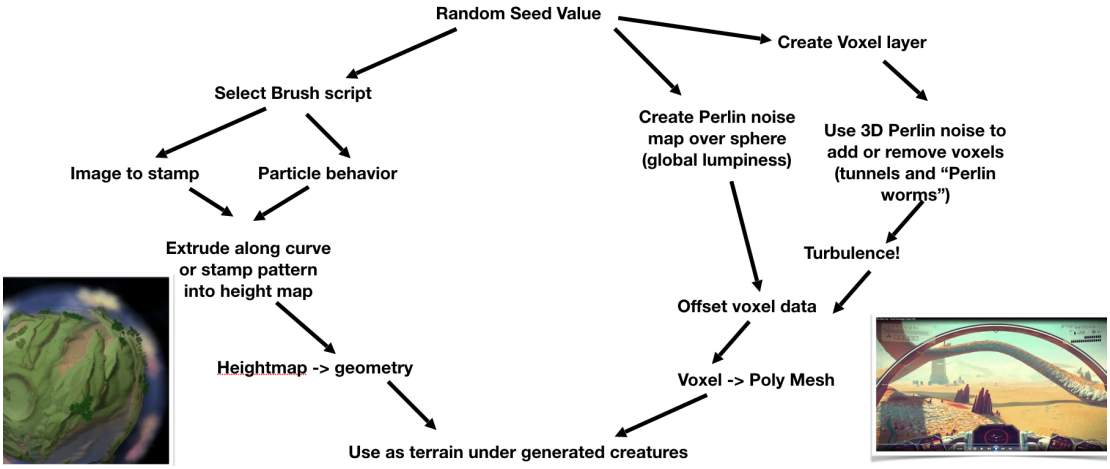


Figure 7.11: Comparing the pipelines of the *Spore* and *No Man's Sky* terrain generators.

7.4 Generominos: turning a framework into a physical toy

Generominos[88][23] are a playable version of the Generative Framework for Generativity. Just as Lego pieces can be combined into many different models, the Generominos provide players with a set of cards representing individual generative methods, as well as potential inputs, outputs, and modifiers (which recontextualize the inputs or output).

It began as a cut-and-play set of cards for Michael Cook's PROCJAM collaborative newsletter *Seeds*[104][24], and was included in my chapter on designing art toys in Tanya Short and Tarn Adams' PCG anthology[76]. Eddie Melcer helped me evaluate them in a classroom setting (his *Alternative Controllers* game design class that I was TA'ing for). I then Kickstarted them[25] for funding to print them, mailed out several hundred physical decks, and released an online editor[26][27].

So what *are* the Generominos?

Generominos, like dominos, have two sides, only in this case, there is an **input** side and an **output** side. Each side has some number of sockets. In Figure 7.12, the cards for a Voronoi diagram has one input (2D points) and two outputs (the points-and-edges of the region, and the shapes of the regions). There are over 180 cards and 16 data types (numbers, sound waves, images, text, 3D meshes, etc). The goal of the exercise is to build correctly-connected **generative pipelines** out of the **generative method**, that begin with some input and end with an output (or more than one of each).

[23] Some text here is included from that paper.

[24] `http://www.procjam.com/seeds/issues/cut-and-play.pdf`

[25] `https://www.kickstarter.com/projects/1921765752/generominos-ideation-cards-for-interactive-art`

[26] `http://www.galaxykate.com/generominos/editor-dev/`

[27] The name "generomino" was suggested by Florian Schmidt.

Two cards can be socketed together if the input from one matches the output of another, as in Figure 7.12. In that example, the stock market produces a numerical value, which can be used to control the amount of propane being fed to a fire. One can imagine this artwork in a corporate lobby, as the stockmarket crashes and the fire dims or jets upward.

Most of the cards are "transformations" of one data type to another (generative methods). There are also other kinds of cards than can be added to the pipeline or used to inspire new ideas.

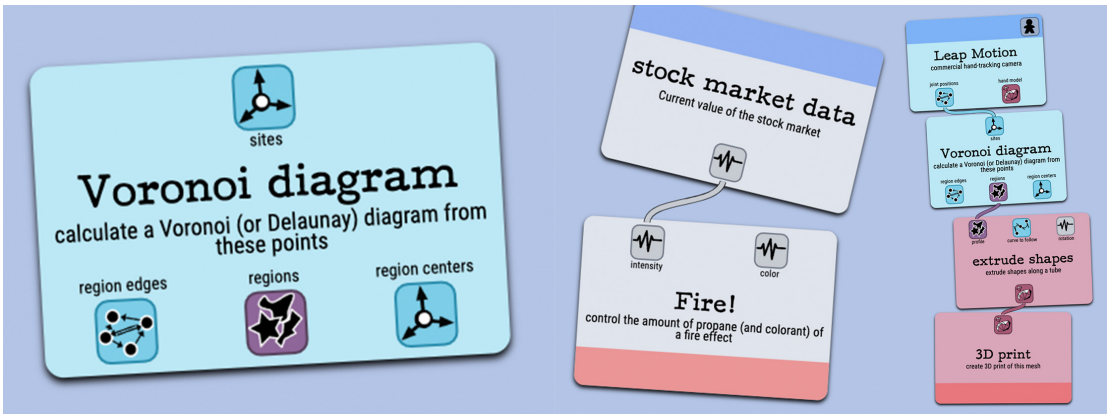


Figure 7.12: The basic shape of a Generominos card and pipeline.

Input and output cards

Input cards have a blue stripe along the top (to indicate that one can't extend further in that direction), a set of output data types, and a "use" type (content, agent, or sensor) than determines how that input can potentially be modified (agents can be modified with "used by a non-human animal", while content can be "from a historically significant source"). Inputs are things like mice and touch pads, but also toy guitars, webcams, EEG headsets, motion tracking, or Twitter feeds.

Output cards have a red stripe along the bottom, suggesting that they are the

end of a pipeline. These are things like "display on a monitor", "project on a screen", "render to stereographic VR", "light an LED", "spin a motor", or "3D print".

Input mods and output mods

From analyzing many interactive art pieces with this framework, we noticed that a substantial number have unusual ways of using sensors and agents for input. In some pieces, the sensor is used by a non-typical user, like a family, a couple, or a group of strangers. In others, the "player" may be a non-human animal, or a non-biological natural force. We often assume that the interactor is the viewer, but a modifier can reminds us that they *might not be the same person*, or even in the same place. Similarly, a common kind of output can be made interesting or uncommon through its context–e.g., in a public space, on someone's body, or being very very large. To better capture this novelty in existing designs and allow for more generative as well as creative expressivity, we constructed input/output modifier cards that can designate unusual modifications to the agents, sensors, or context of an interactive generative system.

Scenario cards

Often users are unsure how to start designing with the cards, so we provide a number of scenario cards (based on real-world grants) that players can earn by designing an outdoor installation game, a game for cats, an art toy for a senior center, etc. Each card also has a listing of the design constraints that are important for that challenge. One issue with the Generominos is that they do not model the affect created by the pipeline or the context where it will be used. Will this pipeline feel "generative" or "creative"? Would these inputs and outputs

be suitable for a particular use case, such as outdoors, or for use by the elderly or children? These are important questions for system designers to consider, so the scenario cards encourage users to consider context and design constraints in addition to the implementation constraints embedded in the regular Generominos cards.

Evaluation

Generominos is a system-style deck of cards representing a real world set of constraints. We wanted to have cards that clearly represented the actual constraints of a pipeline theory of generativity, but still allow for some flexibility when players can see how two data types would interact, even if there's not a card for it. Therefore, players are provided with additional blank cards and encouraged to write their own, or instead match similar-colored icons which are mostly compatible (like particles and vectors, or waveforms and values. Likewise, to avoid making hundred of cards, some cards can be used in many different ways, representing a class of algorithms rather than a single algorithm.

We tested this to see if it would improve the creativity of our students project in an alternative controllers class. We received positive feedback from the students Figure 7.13, but also were greatly impressed by the quality of ideas and their creativity (the fire/stock market idea used as an example was created by a student group in the evaluation).[28]

The Generominos were an intervention to see if I could get people to adopt the GFG as a language to think about generativity as a pipeline, rather than as some of the frameworks in the previous chapter suggested, discrete categories like "robot art" or "evolutionary art". The nature of the cards (small and slippery

[28] After letting them use Processing with their Arduinos, they had all begun to converge on a limited "Mario but with a weird button" design space.

and tangible) makes them easy to shuffle and rearrange. The variety of cards in the deck guarantees that a card inspired by gallery art ends up next to a card inspired by a Burning Man installation, and a card taken from a Siggraph demo or an academic presentation.

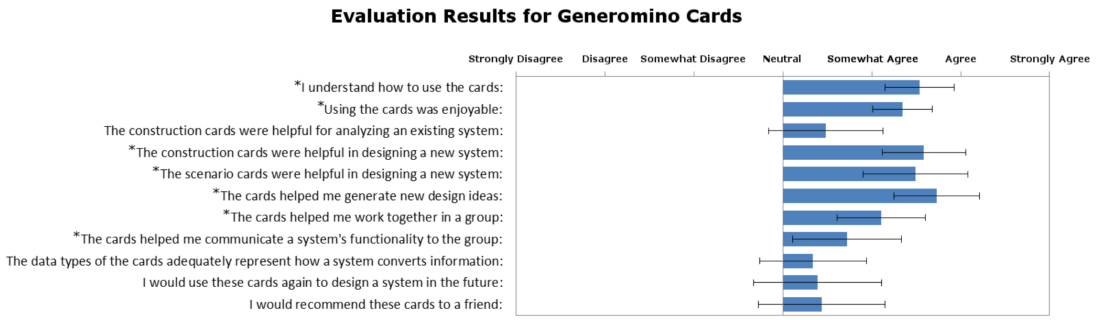


Figure 7.13: Positive student feedback on the Generominos, from Eddie Melcer's evaluation.



Figure 7.14: The Generominos went out into the world and immediately were put to use in classrooms and design studios.

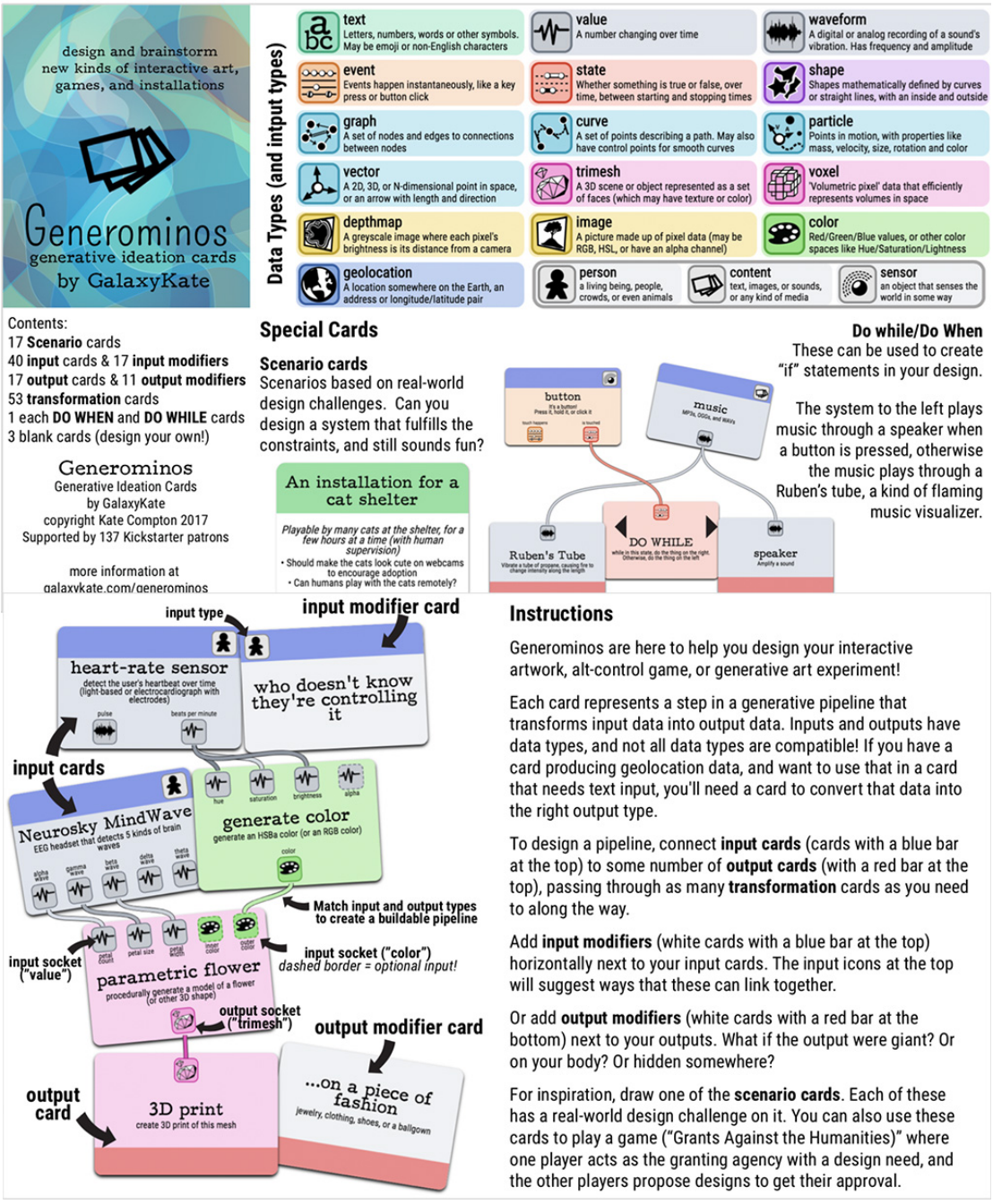


Figure 7.15: Instruction pamphlet for the Generominos.

Chapter 8

Frameworks of Interactivity

How do we interact with systems? Through direct manipulation, reflection-in-action.

> "Only a positive attitude of generosity, a friendly dialogue with your work from the beginning to the end of its development, will produce pleasant variety" - Eva Zeisel

> "Don't think. Thinking is the enemy of creativity. It's self-conscious, and anything self-conscious is lousy. You can't try to do things. You simply must do things." - Ray Bradbury

The goal of this chapter is to assemble a set of tools with which I can analyze casual creators, as well as other systems as **interactive** generative systems, by examining how we **understand systems through using systems**. To that goal, I summarize several frameworks for interactive systems, from philosophy to user-interface design and art. When presented with a new interactive system, how do we begin to use it? Some systems are purposefully opaque, like *The Gostak*[s89] or *BECOME A GREAT ARTIST IN JUST TEN SECONDS*[s90], and their pleasure is derived from exploring and slowly mastering an alien system. Other systems that may be more familiar, like the *Spore Creature Creator*[s216] and *Sketchup*[s210] are intended to be transparent, easy to understand and use. Through these existing

frameworks, and by creating one additional one of my own (the "grokloop"), it should be possible, for any casual creator, to answer two questions:

- "How do learn to use this system?"
- "How do I learn the possibility space of this system?"

Other frameworks have been designed to understand interaction, both with interactive digital system, and in the real world. For the most part, the frameworks for digital systems have been developed to deal with purposeful, productive software systems where the goal is clarity and efficiency of use. These frameworks will help us answer the first question (the second will wait until the second half of the chapter).

8.1 Direct manipulation

In 1982, Ben Shneiderman proposed a term "Direct Manipulation"[364], which he would later expand[365] to describe a category of software in which the user "directly" manipulates the objects of their interest. Much like I am doing in this work, he sought to capture with a term a set of design practices that *already existed* (in cars, games, text editors, and file management software) but whose innovations had not been collected in a way that they could be examined or applied in new situations. Also similar to this dissertation, he defined this category by the set of *effects* it had on users:

- "mastery of the system
- competence in the performance of their task
- ease in learning the system originally and in assimilating advanced features
- confidence in their capacity to retain mastery over time

- enjoyment in using the system
- eagerness to show it off to novices, and
- desire to explore more powerful aspects of the system"

A nice set of effects for software to have, even today, and probably more strikingly compared to the alternatives of 1982! How does a piece of software achieve these effects? To understand it, Shneiderman compares *syntactic knowledge*, the lexicon of possibly arcane commands needed to "speak" to the computer, with *semantic knowledge*, the concepts with which a user understands and strategizes about their task. Task-fulfillment requires the user translating their semantic plan to a syntactic instantiation. A closer mapping of semantic to syntactic will then reduce the cognitive load on the operator, and they will experience the effects listed above.

He proposes a class of interfaces that can "display a representation of the object of interest and permit rapid, incremental, reversible operations through physical actions". Interfaces like these would minimize the semantic/syntactic distance as "each command produces a comprehensible action in the problem domain that is immediately visible". The syntactic input is already cognitively related to the semantic intention, and additionally, the immediate response reinforces the link, even in a novice's mind.

Shneiderman lists a few concrete design principles that create "an integrated portrait of direct manipulation":

- "Continuous representation of the object of interest
- Physical actions (movement and selection by mouse, joystick, touch screen, etc.) or labeled button presses instead of complex syntax
- Rapid, incremental reversible operations whose impact on the object of interest is immediately visible
- Layered or spiral approach to learning that permits usage with minimal knowledge. Novices can learn a modest and useful set of

> commands, which they can exercise till they become an "expert" at level 1 of the system. After obtaining reinforcing feedback from successful operation, users can gracefully expand their knowledge of features and gain fluency."

Implementing a system with these features will result in a positive and powerful user experience. He suggests many user outcomes that will result from this, of which the last three will be the most important for this dissertation:

- Users can immediately see if their actions are furthering their goals, and if not, they can simply change the direction of their activity.
- Users experience less anxiety because the system is comprehensible and because actions are so easily reversible.
- Users gain confidence and mastery because they initiate an action, feel in control, and can predict system responses.

He mentions some outcomes that deal with faster task completion and not needing error messages[1]. On the whole, though, this groundbreaking article deals *very little* with measurements of task completion and the quality of the output. Instead, the argument is made that if the user is confident and continuously learning and can make and test predictions about the system behavior, then the system is successful, and improved outcome quality will follow from that. This focus makes the framework useful this dissertation: in the case of autotelic systems like video games and casual creators, the correct reward for system interaction is *system mastery itself.*

[1] "The annoyance and delay of debugging the format commands is eliminated because the errors are immediately apparent."

8.1.1 Going Deeper in Direct Manipulation

Shneiderman's article persuasively explains the need for direct manipulation interfaces, and suggests a framework (semantic/syntactic) to understand what "directness" may be. Three years later, Edwin L. Hutchins, James D. Hollan, Donald A. Norman[204] take a much deeper exploration into what "directness" means, and discover new components into which it can be further decomposed. Their stated intention is to find the underlying explanations for Shneiderman's list of features and goals:

> There are systems with attractive features, and claims for the benefits of systems that give the user a certain sort of feeling, and even lists of properties that seem to be shared by systems that provide that feeling, but no account of how particular properties might produce the feeling of directness. The purpose of this article is to examine the underlying basis for direct manipulation systems. On the one hand, what is it that provides the feeling of directness? Why do direct manipulation systems feel so natural? What is so compelling about the notion? On the other hand, why can using such systems sometimes seem so tedious?

In doing so, they build a much deeper and more developed ontology from Shneiderman's original work. They identify the semantic/syntactic divide between intention and input (as originally noted by Shneiderman) as one major phenomena, the **Gulf of Execution**. They also identify the relationship between input and output "vocabulary" as a second phenomena of equal importance to the semantic/syntactic divide, and name it the **Gulf of Evaluation**. Each gulf is composed of two distances, the **semantic distance** and the **articulatory distance**, the distance from the intention to the expression in the system's vocabulary, and the distance from that expression to the "physical" instantiation, like an input gesture or the visualization of output.

In addition to these distances, there is a *quality* to good direct manipulation of **engagement**, the feeling that the user is directly manipulating a **model-world**. The model-world metaphor is contrasted with the **conversational** metaphor, in which the user engages the interface in a "conversation" about the model which the model itself is not directly touchable.

In short, **directness** is a "feeling" and **distance** is "factors which underlie the generation of the feeling of directness". We can manipulate the factors of distance in our system, but only through those factors can we affect the feeling of directness and engagement.

8.1.2 The Gulf of Execution and Evaluation

They identify the primary concern of direct manipulation systems to be two gulfs of understanding: the gulf of execution, as the user attempts to execute some intention with the system, and the gulf of evaluation, in which the user must understand the outcome of their action or the system's response. Together, a thorough understanding of the two gulfs will "present a good conceptual model of the system that is readily perceived, interpreted, and evaluated". We can anticipate achieving direct manipulation (the end goal, after all) when a system minimizes the semantic and articulatory distances of the gulfs and ensures a "simple and straightforward" translation loop, from the user's intentions to the system's model, and from the system's model to the user's understanding.

The article repeatedly refers to one particular symptom (or perhaps it is a cause?) of achieving short, successfully-bridged gulfs of execution and evaluation. It is the feeling of closeness (directness) to the object of interest: not closeness to the interface, but the "object" that is being manipulated, like a spreadsheet or program or calendar or chessboard. This object doesn't technically exist, except

in the perceptions of the user and the abstracted data of the system, but they describe the phenomena in very tactile terms: "we should be manipulating directly the game world, touching and controlling the objects in that world".

As they argue: "[t]he goal is to permit the user to act as if the representation is the thing itself." It is worth asking: how *would* the player act as if it is "the thing itself"? We have certain expectations of objects in the real world, which have leaked into our digital expectations as well. If I write on a calendar, I have expectations of the writing appearing immediately, of the writing being permanent until erased, of the ability to move between pages of the calendar, and the concept of "pages" altogether. One very important issue seems to be reaction speed. Any tactile action I take in the real world has some form of instantaneous feedback. Any delay of this effect increases my perceptual distance from the object: "immediacy" simultaneously defines *speed*, *distance* and the cognitive effect they create.

Presence and proximity may then be phenomena of *time*. If controlling an RC car and controlling the Mars Rover are similar problems, ignoring the differences in actual UI complexity, their difference is not in distance, but in the 10 minute delay that the distance to Mars creates. That is, an RC car controlled with a 10 minute delay would have more in common with controlling the Mars Rover, regardless of distance. Rapid feedback seems to be a key issue in shortening the gulfs and making the object of interest feel close at hand, and "present":

> One factor that seems especially relevant to maintaining this illusion is the form and speed of feedback. Rapid feedback in terms of changes in the behavior of objects not only allows for the modification of actions even as they are being executed, but also supports the feeling of acting directly on the objects themselves

Semantic Distance

Semantic distance can be evaluated with two questions, according to Hutchins et al:

> "Is it possible to say what one wants to say in this language?" "Can the things of interest be said concisely?"

There exist many systems that are immensely powerful, in that you can express a vast range of intentions with them. They point out that the classical Turing machine (a read-write head on a strip of marked paper) succeeds at the first question, as any computable intention can be expressed as a Turing machine. The problem is that the expression may be (and almost alway is) immense and unreadable! Thus the Turing machine has poor semantic distance.

Like human languages, each system has its own mapping of which concepts are easy to express or difficult to express (both CSS and Super Mario Maker have been demonstrated to have Turing completeness, eventually). A languages like Python, with its many layers of abstraction and built-in list manipulations, can express anything the Turing machine can, but much more concisely and with better cognitive clarity. The closer something is to a system's core abstractions (Python: lists of things and lambdas, JavaScript: hashtables and closures), the more concisely it can be expressed.

In a digital system, the semantic distance will alway extend from the mind of the user all the way down to the flipflops controlling the flow of electrons in a circuit. But invisibly, most of this distance is bridged for us by operating systems and higher-level languages and GUI libraries. The designer of a direct manipulation system will extend the bridge further with abstractions for whatever their system's focus is.

With each extension, intentions outside the focus become more difficult to express, while intentions inside the focus become clearer. Complex actions become simpler, and more powerful, as single higher-level commands can bundle together many co-occurring sub-actions. These actions may also become *safer*: if the higher-level action avoids any errors that might have occurred by hand-programming it, then each time the user invokes it, they skip over those potential errors.

Thus a system without higher-level commands wearies the user with a high cognitive load (encoding intention and worrying about error). A system that *does* have these higher-level commands has the inverse effect. Or in the words of Hutchins et al, "the feeling of directness results from the commitment of fewer cognitive resources" and "the need to commit additional cognitive resources in the use of an interface leads to the feeling of indirectness".

Not just any higher-level abstractions will suffice, however. The bundling of actions must match how the user would describe task: "Semantic directness requires matching the level of description required by the interface language to the level at which the person thinks of the task."

This is further complicated, because *how* a user would describe a task depends on their familiarity with a task, and also their past training on tools to accomplish the task. Ask a writer and a computer programmer to describe how to make a piece of writing software, and the writer may speak of choosing fonts and maintaining an outline, and the programmer will speak of input handlers and text buffers and undo stacks. Each one's abstraction is *correct*, but each is informed by the tools used to solve problems in the past. The tools we use define how we conceptualize a problem. If a system's "directness" is based on how well the abstraction fits cognitive descriptions of the task, a system can be successful by training a user

to *think* in terms of its own abstraction. Thus the interface attains directness by *reshaping the user's cognition.*

Hutchins et al claim that, with this training loop, often non-direct interfaces may feel direct, if a user has used them long enough (talk with any use of the vi editor for a strong example of this):

"To skilled users, the interface feels direct because the invocation of mediating structure has been automated. They have learned how to transform frequently arising intentions into action specifications. The result is a feeling of directness as compelling as that which results from semantic directness."

Further evidence of the plastic relationship between cognition and abstraction is provided by the mods and macros that very skilled users build *on top* of the system's built-in abstractions:

"Sometimes users have a conception of a task and of a system that is broader and more powerful than that provided by an interface. The structures they build to make contact with the interface go beyond it. This is how we characterize virtuoso performances in which the user may "misuse" limited interface tools to satisfy intentions that even the system designer never anticipated."

The Spore Creature Creator[s216] has a very direct interface, but while simplifying some actions, other possibilities were lost: for example symmetry made better looking creatures, so actions performed while editing creatures would always be mirrored on both sides. But the system had many virtuosic users, and they laboriously exploited bugs in the system until they were able to achieve asymmetry. *Minecraft*[s136] is also a very direct system, but often users want to think at a higher-level than the interface (moving one block at a time) and so a vibrant market in macros for "common" tasks (like building electronic circuits) developed.

Thus, while semantic distance *is* a property of the system, it is *also* negotiated with the user: "..some of the production of the feeling of directness is due to adaptation by the user, so that the designer can neither completely control the process, nor take full credit for the feeling of directness that may be experienced by the user."

Articulatory Distance

In addition to the semantic distance, the user of a system must also bridge the *articulatory* distance. Shneiderman had declared the gulf to be semantic/syntactic; here the gulf is semantic/articulatory, so "articulatory" seems to be closely related to syntactic expression, though why they chose a separate term is unclear. Articulatory distance, here, is the space between the *meaning* and the *form*, by which they mean the actual instantiation, in mouse clicks, string input, gesture data of machine-processable input. Output also has an articulatory distance: graphics, audio, actuators, or text by which the machine creates human-processable output. In each case, in Hutchins, the articulatory distance is the distance both the system and the user must generate "language" that the other can understand:

> "Frequently the output interface language does not share vocabulary with the input interface language. Two forms of interface language-two dialects, if you will - must exist to span the gulfs between user and device: the input interface language and the output interface language."[204]

This has traditionally been the case with screen-based interfaces. The user's input language is physical: a button press, a gesture, a mouse click. The screen can only speak back in a visual output language, lighting up a set of pixels with the correct color such that they are perceived as shapes, and those shapes are perceived as objects in the world model. Some unusual interfaces speak the same

language for both output and input, such as posing a robotic arm to "teach" it to perform actions [137]. With devices like the Google Home and Alexa, devices that only receive input and respond via voice (input via the human's voice and output with their own synthesized voice), we may finally see mass-market interfaces with matching input-output languages.

From their vantage point in 1985, the authors were able to predict technologies with better "articulatory similarities" than they currently had available, and suggest that eventually we could develop an interface we could call "onomatopoetic", "an interface that permits specification of an action by mimicking it".

> "This potential has not been exploited, in part because of economic constraints. The restriction to simple keyboard input limits the form and structure of the input languages and the restriction to simple, alphanumeric terminals with small, low-resolution screens, limits the form and structure of the output languages."

Issues with direct manipulation

Like Shneiderman, they recognize that some tasks are not suited to direct manipulation: sometimes a tactile metaphor is not faster than being able to script higher-level specifications, and this recognition is still true today. We still have command-line terminals in operating systems, though most users will only even use GUIs. Alternately, though there are many direct-manipulation-influenced visual programming language (like Scratch), most users will only use the same syntactically complex, tactilely distant style of programming that was the default in 1985.

In addition to issues of convenience or speed, they identify another issue with direct manipulation interfaces, an issue that is at the core of this dissertation. The semantic and articulatory distances are best bridged by considering the *very specific* semantic and articulatory problems for the particular domain, not by solving

for a general case. Semantic distance solutions must support the user's understanding of the problem (as discussed above). The articulatory solutions must reflect the same. But that power requires a sacrifice of generality and flexibility:

"The result of matching a language to the task domain brings both good news and bad news. The good news is that tasks are easier to specify. Even if considerable planning is still required to express a task in a high-level language, the amount of planning and translation that can be avoided by the user and passed off to the machine can be enormous. The bad news is that the language has lost generality. Tasks that do not easily decompose into the terms of the language may be difficult or impossible to represent. In the extreme case, what can be done is easy to do, but outside that specialized domain, nothing can be done."

And eventually, this system may train the user to stop considering certain possibilities altogether:

"While moving the interface closer to the users' intentions may make it difficult to realize some intentions, changing the users' conception of the domain may prevent some intentions from arising at all." Direct manipulation systems encourage some forms of action, both in the interface and the user's mental model of the problem. But in doing so, they necessarily suppress other forms of action.

Hutchins, Hollan and Norman have here created a strong framework that builds on Shneiderman's original insights. The concept of "distance" (separate from "directness"), the recognition of the execution and evaluation gulfs, and distinguishing between the articulatory and semantic components of each, give a terminology that will be useful in the rest of this dissertation.

At the heart of this framework, and at the center of many other similar frameworks Figure 8.1, is a *loop*.

This loop consists of a user:

- interacting with a system, causing the system to perform some action and produce output
- interpreting or understanding the output
- revising the understanding of the system
- interacting again with the system, using this new knowledge as guidance

Hutchins, Hollan and Norman identify such a loop behind the "direct manipulation" of Shneiderman's interactive software interfaces. But this loop is far from native to software, it exists in *any* system: digital, social, or physical. One instance of such a loop concept is **reflection-in-action**, as laid out in Donald Schon's book on workplace expertise "The Reflective Practitioner".[351]

8.2 Reflection-in-Action

How do professionals understand their practice? Schon contrasts his framework for the "reflective" practitioner with the previous understanding of expertise as "technical rationality". Technical rationality is defined by the professional standards and practices derived from rational theory and institutionalized empirical knowledge. Organizing practice through these principles leads to "instrumental practice", as the practice becomes a process of choosing the correct technical "instrument" to achieve an end goal. But this process requires a formalistic understanding not only of the solutions, but what the problems are at all! Instead, he suggests that in most situations, "we behave according to rules and procedures that we cannot usually describe and of which we are often unaware".

The looping process of the reflective practitioner is as follows:

> "The practitioner allows himself to experience surprise, puzzlement, or confusion in a situation which he finds uncertain or unique. He reflects

> on the phenomena before him, and on the prior understandings which have been implicit in his behaviour. He carries out an experiment which serves to generate both a new understanding of the phenomena and the change in the situation. When someone reflects in action, he becomes a researcher in the practice context. He is not dependent on the categories of established theory and technique, but constructs a new theory of the unique case"

Reading into this description of the loop, several steps seem to be glossed over (how they determine the content of the experiment, for example), so I would describe the full loop here:

- examination of the phenomena
- planning an experiment (requiring an actionable hypothesis)
- executing the experiment
- observing the results
- gaining new understanding, modifying or generating new theories

Schon describes this process as a continuous experiment, albeit a flexible and playful and often improvisational one: "Exploratory experiment is the probing, playful activity by which we get a feel for things. It succeeds when it leads to the discovery of something there"

The "discovery" is some information or feedback which creates a better understanding of the phenomena being examined. For this reason, I think that the formation of the hypothesis may be important than Schon's writing suggests. What do we *expect* will happen when we use an interface? Do we form this expectation explicitly, or, more likely, is it a blend of conscious and subconscious awareness? How formal is the hypothesis that precedes the interaction? It must be at least well-formed enough for us to recognize when it fails. The "surprise" of

a failed hypothesis is often the signal that we need to build a different model than we currently have, and the surprising outcome brings the knowledge necessary to revise the model, thus continuing the reflection-in-action cycle:

"Much reflection-in-action hinges on the experience of surprise. When intuitive, spontaneous performance yields nothing more than the results expected for it, then we tend not to think about it. But when intuitive performance leads to surprises, pleasing and promising or unwanted, we may respond by reflecting-in-action"

8.2.1 Models of learning: Single and double loop learning

Chris Argyris[15] built further on Schon's concept of continuous revision. Most systems have a primary action loop: we perform an action, examine the results, and perform the next actions. These action loops exist for interactions with software, as well as in the corporate actions that Argyris is concerned with. Argyris calls this **single loop learning.** We learn to modify our actions in response to the outcomes.

When the user begins to examine the system itself, and discover new features or constraints in it, that becomes **double loop learning**. The user continues to perform actions on the system and to respond to its output. But in this case, a *second* loop occurs as the user updates their mental model of the system, which then changes how they respond to the output. He suggests that in business, this second loop is performed over longer time spans than the action loop. Is this also true in software interfaces? The evolution of the user's mental model of the system and the user's continued interaction with the system *might* occur simultaneously. Does the user needed to be *guided* to update their model, *reminded* to push at the systems perceived boundaries? That is, can we build a system where

continuous double loop learning is the default *norm* and not the heroic rebellion against structure that Argyris paints it as (in the slow-feedback corporate world at least). This may be the case with direct manipulation systems: with small gulfs of execution and evaluation and a short action loop, the object of interest will be present and directly manipulable, and thus learning can occur continuously.

8.3 The site of interaction

If the single loop of learning could be described as *using the system to effect change* and the double loop might be described as *examining the affordances of the system and critiquing it as a tool*, then several new frameworks become pertinent. Sometimes when we use a tool, it is an extension of our hand, and our attention is on the work we are performing, and the tool becomes invisible. But to examine the tool, to understand its limitations, to guess at new uses, we have to look *at* the tool, to make the invisible instrument visible. Paul Dourish uses the Heideggarian concepts of "present-at-hand" (the visible tool, to be examined) and "ready-to-hand" (the invisible tool, a transparent conduit of intention) to explain these phenomena. If the goal of this dissertation is to examine the tools of creativity, do we look at the work, the tool, or the hand holding it? To quote the title of Dourish's book, we must know "where the action is".

Schon is interested in the origins of a practitioner's expert knowledge, especially when it is stored as implicit, non-verbalizable "know-how". For this, he turns to Michael Polanyi's The Tacit Dimension[310], a work about how knowledge can exist outside of explicit definitions, famous for the line "I shall reconsider human knowledge by starting from the fact that we can know more than we can tell".

Polanyi is also interested in where the site of interaction is, and pertinent to the goals of this dissertation, how we use tools to explore and understand spaces:

> "Anyone using a probe for the first time will feel its impact against his fingers and palm. But as we learn to use a probe, or to use a stick for feeling our way, our awareness of its impact on our hand is transformed into a sense of its point touching the objects we are exploring."

We understand the tool (and its investigations in the world) through the pressure on our hands, its resistance, but the tool itself is a transportation to the actual site of investigation "some distance from the original feeling". The physical resistance we feel in our hands is a translation of what the tool tells us about the space its is exploring, and so feeling physical resistance *is* feeling resistance in the distant space: "We are attending to the meaning of its impact on our hands in terms of its effect on the things to which we are applying it."

With a doctor's probe, the resistance of tissue travels from the site of the tissue through the probe and up into the tissue of our hands, where we feel the analogous resistance in our hands and can interpret it. For a software tool, the translation can be more metaphorical, like the resistance of vertex in *Adobe Illustrator* to leaving a comfortable snapping point when dragged with a mouse. Here the mouse and screen together form our probe, as the mouse itself cannot express haptic resistance as the doctor's metal probe can. It was strangely difficult for me to write this sentence, because my mind *remembers* the *feeling* of resistance from the last time I used Illustrator. Even the incomplete probe of mouse-and-screen had, in my mind, vanished to the point where I could only remember the resistance of the vertex.

When a tool is used to know an untouchable space, the space becomes touchable through the directness of the tool. But the tool becomes part of our body:

> "Whenever we use certain things for attending *from* then to other things, in the way in which we always use our own body, these things change their appearance. They appear to us now in terms of the entities to which we are attending *from* them, just as we feel our own body in terms of the things outside to which we are attending

> *from* our body. **In this sense we can say that when we make a thing function as the proximal term of tacit knowing, we incorporate it in our body–or extend our body to include it–so that we come to dwell in it.**" [310] (bold emphasis mine)

This may explain why so many of these works that I have cited above use physicalized hand metaphors and words to describe their phenomena. We "touch", "feel", "manipulate" the objects in our interface, because our hands extend themselves through the keyboard to reach the object *inside* the system. Perhaps this is why time delay is so disconcerting in such systems. If my virtual hands no longer respond to my intentions, or delay in returning sensations to me, they have been severed, and are no longer working hands.

8.4 The Grokloop

The frameworks above all concern themselves with *interaction with a system that cannot be directly touched*. The distance separating our intentions from our "touch" may be as short as length of the neuron chain from our mind to our hands (Polanyi), or extend from our plans to the sphere of eventual professional operations (Schon) or make multiple leaps from our mind to our hands on a keyboard to a virtual digital model (Hutchins, Hollan and Norman, and Shneiderman). Each framework has a different answer for the question "How do we investigate what we cannot touch?" The answer for each also contains a feedback loop, sometimes illustrated(Figure 8.1). If we add up all the steps in each loops it looks something like:

- make a hypothesis
- act

- examine the result
- update one's goals or mental model

Hutchins, Hollan and Norman are working with digital systems, and so they add substeps:

- transform the hypothesis to a planned action (gulf of execution - semantic distance)
- enact a planned action (gulf of execution - articulatory distance)
- read the result/output (gulf of evaluation - articulatory distance)
- interpret and understand the result (gulf of evaluation - semantic distance)

Is this enough to address the interaction issues for casual creators? Casual creators have the interesting property that they have a generative possibility space which the user is exploring (and having fun while doing so). These users aren't seeking to gain expertise with the system, they are using the system to get to know a possibility space. Generativity also adds another level of indirectness to Hutchins, Hollan and Norman's steps. We also need a concept of how user changes a model, which *then* changes into an artifact. The user is actually learning *two* possibility spaces: the model space (what the limits of the system/UI are), and the artifact space (what kinds of artifacts are represented within those limits). So I need to add the steps:

- modify the model
- reinterpret the model into an artifact

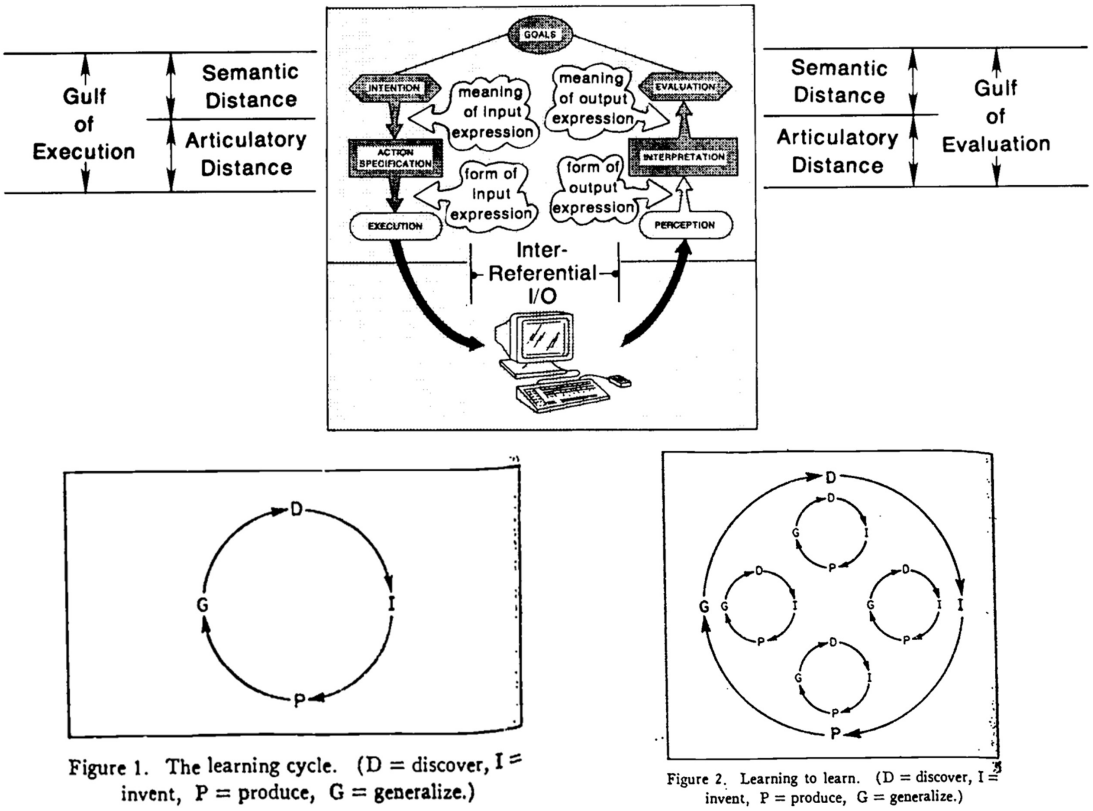


Figure 8.1: Loops from Argyris[15] and Hutchins[204], showing the same loops of iterative learning.

I also want to make it clear that this stage of thinking is *grounded* in the artifact itself, and that this is a conversation between the user and the generative space, in which the user is probing the space through their actions and the probe is returning artifacts.

One other niggling problem I have is that we don't have a term for all of these learn-by-looping diagrams. I found myself wanting a way to say "the speed of learning depends on how short the loop is", which is very true, but there are many loops in games and game-like objects, and I wanted something that specifically referred to the process of iterative learning, while being short and catchy enough to drop into conversation. As someone who does contracting for companies developing casual creators, I needed a punchy way to tell them what was

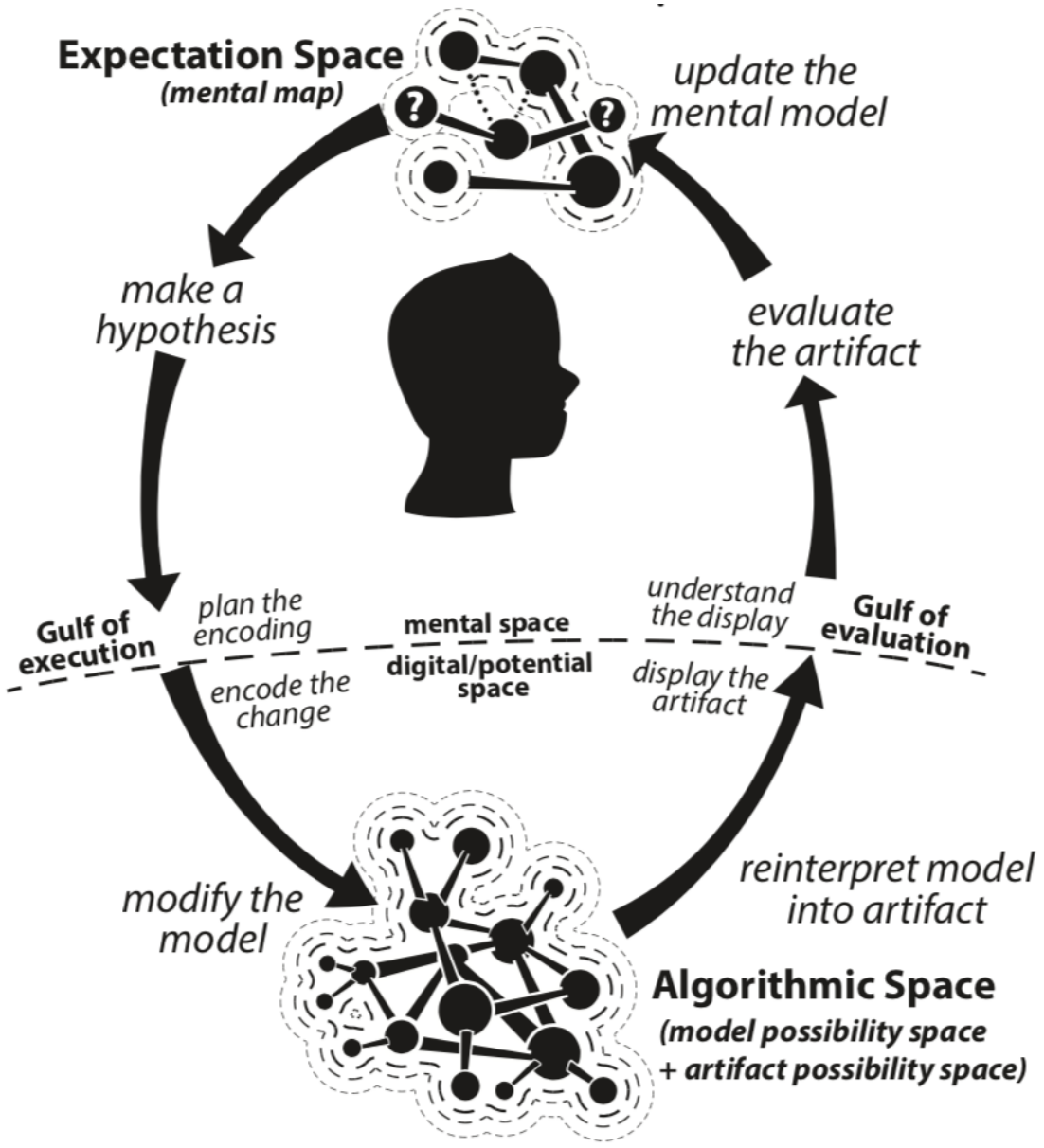


Figure 8.2: The "grokloop", a process of exploring a generative space through interactions with an artifact

wrong with the whole loop.[2] "Reflection-in-action" is a philosophy, and "double-" and "single-" loop learning is very general. Hutchins, Hollan and Norman's loop is the closest to what I need, but they neglected to name it. So I came up with

[2]"Gulf of execution" and "gulf of evaluation" have also so far been helpful in the field, when trying to point out problems to clients.

"grokloop"[3], a slightly cheesy term which seems to have stuck. Fortunately, it fulfills the punchiness requirement,[4] and from the cursory evaluation of the zine in chapter 26, it has been cheerfully adopted by the target audience of interactive/generative system designers.

The grokloop framework rolls up the ideas of direct manipulation (instant feedback helps us learn), Hutchins, Hollan and Norman's expansion of it (the barriers to direct manipulation are the two gulfs), Argyris and Schon's reflective practices (we learn by going around a loop), and Polanyi's assertion that much of this learning happens at the intersection of person and tool. The grokloop will make many appearances in this dissertation. It makes possible the exploration of the different casual creator possibility spaces in the next few chapters, and shows us where the problems are that we need to solve for *difficult* casual creator domains like music and programming.

[3]From Heinlein's term 'grok', now popularized in internet lingo, which means to drink in understanding to the point that the known information becomes inseparable from oneself.

[4]"Your grokloop is too slow" is a short and almost always true piece of feedback for students and clients.

Chapter 9

Possibility Spaces

> "The designer must understand that form does not follow function nor does form follow a production process. For every use and for every production process there are innumerable equally attractive solutions."
> - Eva Zeisel[448]

Now we have a framework to understand the construction of generative pipelines (the GFG) and a framework for understanding how the user is in conversation with that pipeline (the grokloop). We have seen in the grokloop diagram that the generative pipeline creates some *space* of possible artifacts, and the user has some mental model of what that space is. Next we need to look how the algorithmic properties of the pipeline define its possibility space, and in what ways the user can navigate that space, moving from artifact to artifact, browsing artifacts, or even seeing a "map" of the space. The next six chapters (chapter 10, chapter 11, chapter 12, chapter 13, chapter 14, and chapter 15) will explore different kinds of possibility spaces. Casual creators can make many different kinds of artifacts, from many creative domains, but these possibility spaces (and their affordances) comprise distinct *subgenres* of casual creators that share properties with each other. This chapter will address ways of *talking* about possibility space, including language we can use to describe them.

9.1 What are possibility spaces?

Often when we speak of generative algorithms, we talk about **possibility spaces**. Any generative pipeline that can be authored has a mathematical space of artifacts which it is possible to generate with that system[1]. Any artifact *outside* that space is unreachable by that generator. Simple pipelines may have a very narrow possibility space of only a few artifacts, or a very shallow one of many nearly-identical ones. Even very complex systems still have a limited range. A professional 3D modeling tool may only be able to model voxels, or triangle-based meshes but not fluids or cloth or materials of varying density, while a casual creator 3D tool might only generate extruded tubes, or radially symmetrical objects. Lego blocks can be used to build "anything", as long as the objects are made from Lego blocks.

The risk created by bad artifacts in the possibility space, and thus the system's choice of possibility space, depends heavily on the system's intended use and user experience. The possibility space of a text editor like Microsoft Word encompasses *all ASCII texts*: good, bad, or meaningless. This is important because Microsoft and its users want to know that there is no great textual work that can't be written in it (assuming an ASCII-compatible language). The risk of writing bad or meaningless text is left as a burden on the user, which is a symptom of a decidedly non-casual creator system. There are also few tools to help the user navigate this possibility space or find good artifacts in it, aside from the occasional nudge from the spell-checker. Casual creators, as autotelic software, may instead opt to not require that any *particular* good artifact exist in the possibility space, as long as *some* good artifacts do (and few bad or broken ones do).

Casual creators are a subcategory of interactive generative system. Interactive

[1]For example, a random number generator can generate any floating point number of some precision between 0 and 1.

generative system have two properties.

- a possibility space of artifacts, defined by the generative pipeline
- a way to navigate and understand that possibility space

When those systems privilege safety, power, and flow (at the expense of total control) they are also casual creators[2]. The **possibility space** of a casual creator is one of its most defining characteristics. Even the way in which we interact with a casual creator is *defined* by what kind of possibility space the algorithm can generate. Conversely, the *actual* possibility space of a system is defined by what parts of the space the interaction allows us to access.

We can see an example of this in the Windows 3.1/Windows 95 game *Freecell*[s75][218]. In a real-world game of *Freecell*, there should be a truly incomprehensible number of possible layouts, 8 x $10\hat{6}7$ unique "deals". If we think about a *Freecell* deal as a procedurally-generated level (as it is!), then the possibility space should be giant. *However*, in Microsoft's digital version, the way these deals are accessed is with a seed value from 0-32000 (plus "-1" and "-2") entered at the beginning of the game. If we look at the deck shuffle algorithm as a generative pipeline, it has an informational choke-point at the very beginning, thus limiting the *total* possibility space to 32,002 deals[3]. This smaller space allows other interesting player behavior (see chapter 19).

Thus, one of the primary ways we can begin cataloging a casual creator is **by its possibility space**. This chapter will begin by outlining some of the core concepts of possibility spaces. The next chapter will then showing some examples of *very simple* casual creator possibility spaces. The remaining chapters of this section will examine what happens with *particular* kinds of possibility spaces.

[2] A system like Maya is an interactive generative system, but **not** a casual creator.

[3] . . . as long as each seed is only used in a single deal, as *Windows Freecell* does, not a sequence of deals

9.2 Translating from model to artifact

Let us examine the difference between a **model** and an **artifact**[4]. In evolutionary algorithms, these are referred to as the "genotype" and the "phenotype", but I have a hard time keeping those terms straight, and I find that they needlessly confuse newcomers as well as come with the sometimes-unhelpful baggage of real world biological expectations, so I will be using "model" and "artifact" in this dissertation.

A model is a static representation of an artifact, as it is modeled in the system's most specific components, at a level where the outcome is maximally specified with the minimum amount of data. This is often a *digital data structure*, but can also be physical. A "model" for a Spirograph[s214] picture would be something like:

- 42-tooth fixed, 24-tooth rotating, 4 turns, color red, pen in the 2nd hole
- 48-tooth fixed, 24-tooth rotating, 4 turns, color orange, pen in the 5th hole
- 52-tooth fixed, 24-tooth rotating, 24 turns, color yellow, pen in the 5th hole
- 63-tooth fixed, 24-tooth rotating, 24 turns, color blue, pen in the 5th hole

This results in the following picture:

If you had this instruction list, you could reproduce the Spirograph drawing. Each instruction has a number of fixed teeth, a number of rotating teeth, a color, a hole, and a number of turns. If you know that the Spirograph set has only 10 colors, the color could be represented by a number from 1 to 10. Likewise, if the set only has 24 kinds of gears, we can represent those as an integer 1-24 (because floating-point gears don't make sense here). This limits our model to **only good solutions in this space**.

[4]As we use these terms in this dissertation. Both terms invoke a terrible variety of meanings across many fields

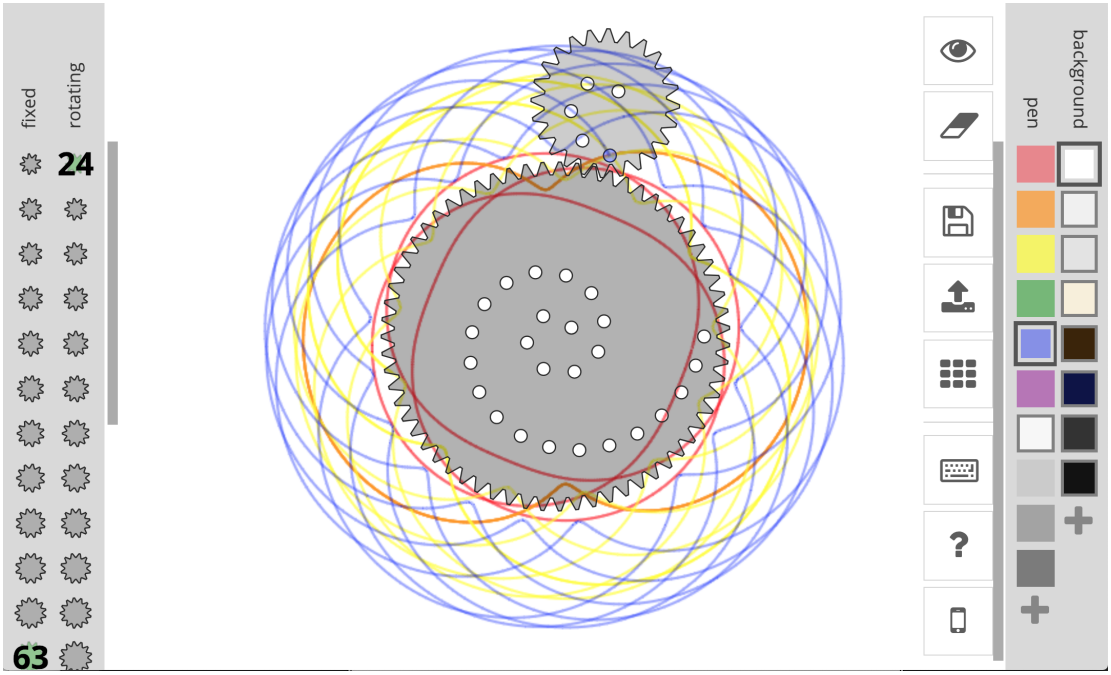


Figure 9.1: A spirograph drawing resulting from the previous instructions, drawn with the Inspirograph online simulator[142][s101]

We *could* have a system that represents color as a 3-dimensional vector, which would increase the possibility space of drawings, but if that color isn't selectable in our UI (or available in our physical kit) then that part of the space **is not accessible**. If we used a floating point, or unbounded integer to represent the number of teeth, we allow models with 2, -24, 240, or 24.5 teeth, which are all invalid.

We *could* also compress the model into a single number, from 1-6, and the instructions for a value of "3" would be to use the 3rd smallest gear, with the 3rd smallest rotating gear, with the 3rd pen, in the 3rd hole, for 3 turns. This is a much smaller model, which describes no invalid possibility space, but useful possibility space is *cut out* by this model.

These are two models that are likely to be *inappropriate models* of this space[5].

[5]It is *possible* to design a valid experience around an otherwise inadvisable model. But an ill-suited model of a domain (over-broad, over-narrow) will make designing a good system much

One is overly broad, and contains bad or invalid parts of the space. One is too narrow, and removes useful parts of the space. There is no *correct* model, only ones that suit what the users is doing in the space and how much the system wishes to trade **safety** or **ease** for **control**. For example, we might want to restrict the number of turns to only those that finish a complete cycle so that the loops are smooth and continuous. Or we might want to give the user that choice, so that they could use that as an artistic tool (to make multicolored partial segments, for example). Even the single-number representation could be useful, if we want the user to control the Spirograph with a single dice-roll. If the experience is interesting enough, the user may never notice the lost possibility space.

For any given system, there is no single model or single way of interacting with that model, there are only ways that provide different balances of safety, ease, control, or power. This decision can be called "directness", as Togelius et al describe different possible balances for dungeon-level generation:

> "1) directly as a grid where mutation works on the contents (e.g. wall, free space, door, monster) of each cell, 2) more indirectly as a list of the positions, orientations and lengths of walls [...], 3) even more indirectly as a repository of different reusable patterns of walls and free space, and a list of how they are distributed (with various transforms such as rotation and scaling) across the grid, 4) very indirectly as a list of desirable properties (number of rooms, doors, monsters, length of paths and branching factor), or 5) most indirectly as a random number seed." [411]

There can also be many **layers of models** for a single system. We may wish to save artifacts as single seed for optimum storage for thousands of artifacts. On the other hand, a generator that needs to reason over or modify an artifact needs a model that is more direct than a seed, but also more abstract than the pixels-and-triangles rendered model that we present to the user.

more difficult.

We can see this in systems like *Spore*[s215], where a planet's terrain-generation code was controlled by a seed value and a selector for one of the several dozen terrain scripts. This model was progressively "unpacked" into subsequent models through multiple pipelines, leading to a model stack that looked like this:

- seed value and script selector
- script: multiple brush scripts and their order (each brush is a set of Swarm particle instructions)
- simulated particles with positions, velocities, and forces (ie, drag) over time
- ribbon geometry and brush shape/size/rotation/height/bumpmap data
- height map (a cube map to be wrapped around a sphere)
- finished artifact: 3D-triangulated deformed sphere mesh

9.3 Describing a possibility space

Famously, the No Man's Sky marketing team advertised "18 quintillion planets"[90], a numerical size for the possibility space that came from the number of seeds available to the generator, which was itself just a function of the number of bits that they stored (64). Are these actually all *different* planets? If they had doubled the seed size to 128 bits, would they have gotten 18-quintillion-times the number of planets? This fallacy of generative content is *extremely common*, and it is not rare to find papers or projects touting their combinatorial possibilities as a single number, a mathematical solution to a multiplication problem. However **the number of numerically distinct artifacts in the space is the lowest-quality measure of a generative space**.

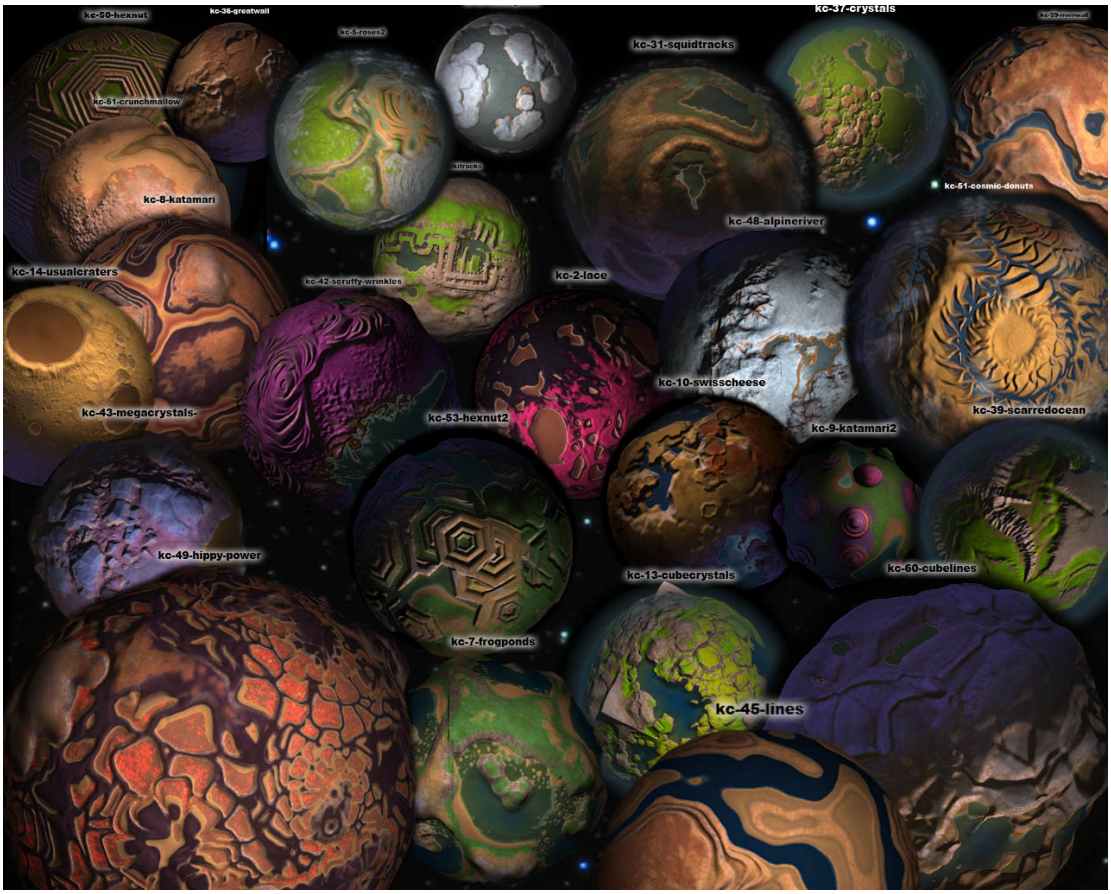


Figure 9.2: Some of the Spore terrain scripts I wrote. Each script was a set of "brushes" written in the Maxis in-house particle system language Swarm. These would rain particles down on the planet, move them according to some set of choreographed forces, and leave a different kind of stamp (riverbed, lake, alpine mountain, round dome, e.g.) multiplied by different height values. Because of the stochastic nature of these brushes, some bad behavior was common, so I pre-ran dozens of planets for each script to save a catalog of good seeds that could be used in-game.

I coined the phrase **10,000 bowls of oatmeal** to describe this in my introductory post to generativity[73]. As an example, imagine that we make a bowl-of-oatmeal generator, which uses whatever complex pipeline we might build for dynamic flake-generation and oat-placement, to create bowls of plain oatmeal. If we consider that each oat's placement is a variable, we have a possibility space of millions or trillions of possible mathematically-unique unique bowls. And yet,

each bowl will be experienced as *the same bowl*.

This phrase so accurately describes the issue with the No Man's Sky planet generation that it was included or used as a headline in multiple articles about the game.[187][54][254]

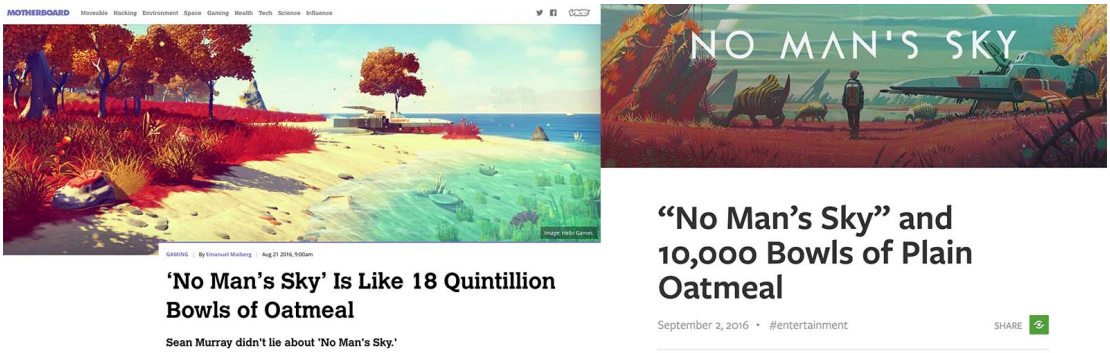


Figure 9.3: "Bowls of Oatmeal" as a headline.

The "big number" is both seductive and a dead-end to us as system designers (or marketers)[102]. Fortunately, we have other options to discuss and analyze possibility spaces.

Property analysis

Any given artifact has a set of higher-level properties. This is distinct from a "model", which is the low-level instructions for creating the artifact, or the "artifact", which is the object as it is experienced by the user (or its least-abstract rendered form). The experiential properties of this object are properties which may be derived *from* that artifact, but are not explicit in its representation.

Each of these examples has a very different set of properties. Some of them are directly calculable from the artifact, like center of mass of the chair, or volume of the teapot. Others can be approximated with some simulation or calculation, like choke-points or fairness in the strategy map, or speed limitations of the racing track. Others, though, are not calculable, or may even vary person to person,

Example system	Model	Artifact	Properties
2D platform game level	array of edges or grid of tile-pieces, location of pick-ups and threats	2D plane of sprites	Difficulty, precision/timing needed, momentum (Sonic-style) or routing (Mario-style) puzzles, multiple routes, pick-ups places on-route or in difficult to access side routes
strategy game level	grid of terrain heights and an array of resource nodes (vector position, type, amount of resource), starting positions	3D terrain mesh of the level, calculated navmesh, 3D objects for resource nodes	Fairness, uniform or asymmetric strategy optimality (resource wealth vs defendability of starting locations), chokepoints, open vs narrow field of combat
racing game level	Bezier points representing a curve, banking angles, material properties of the track vs grass	3D model of the track, with geometry physics properties	Top possible speed at each point, benefit of shortcuts through the grass, suitability for different car tunings, mix of straightaways or sharp curves (and the rhythm of alternating between them)
3D printed chair	list of connection points, edge-node graph to be used in the surface-tension surface generator	3D watertight mesh generated with surface tension algorithm, or the actual chair	Center of gravity, compression and torque strength, ability to stack, comfort of seat and backrest, weight, material use, printability
3D printed teapot	horizonatal radius and height of the body, wall thickness, spout curve, lid radius, selector for lid and spout style	3D watertight mesh, or actual teapot	Aesthetic quality, visual balance, thermal properties, hydrodynamic properties of the spout (splashing), ability to fit a tea strainer
evolved radio antenna	an array of bends, each with 2D angle and distance from the last bend	actual radio antenna, or simulatable model	Radio transmission of the antenna: "voltage standing wave ratio" at the transmit and receive frequencies, across a range of frequencies

Figure 9.4: Models, artifacts, and properties for three examples of typical game systems and three real or hypothetical generative design examples, like generative chairs[126] and antennas.[196]

such as aesthetic or play-style preferences. Any given system will have a set of properties, some of which are significant to the experience of the artifact and some of which are calculable to a floating point number.

When there are properties which are *both* significant and calculable, Gillian Smith has proposed a framework called “expressive range”[383] to map a possibility space according to those properties. Smith identifies two properties, leniency and linearity, for her procedurally-generated platformer levels. Platformer levels have many more significant properties, but these two can be directly calculated from the artifact with some accuracy. This ease is important, as the next step is generating 10,000 levels in order to visualize their frequency, a number that would be impractical for human-evaluated or even simulated heuristics. This expressive range histogram allows the system designer to understand the distribution of the current possibility space (as seen through those two properties), as well as compare

histograms from different tuning values of parameters of the generative pipeline.

Michael Cook's *Danesh*[s41][105] re-implements Smith's expressive range analysis inside of an interactive Unity tool for "less experienced users." In Danesh, users can slide parameters to see the results on a generator. They can also select two of their properties and run either an expressive range analysis (ERA), or random expressive range analysis (RERA) to find new tuning values. It also allows the user to optimize for different parameters. The authors also express hope that it will eventually be able to create meta-parameters that better encode desirable property-specific movements through the possibility spaces.

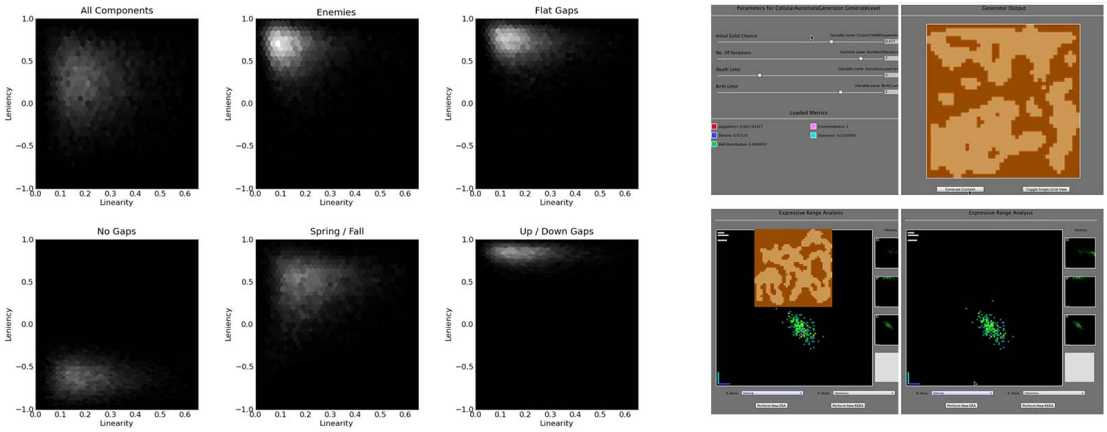


Figure 9.5: Expressive range(right)[383] and Danesh[105][103] both map properties of generated artifacts onto a 2 dimensional space. Danesh elevates the expressive range into a casual creator for generators by adding slider-tuning, autotuning and browsing content with rollovers.

Navigation-functionality analysis

The quality of a possibility space is important to a casual creator, but so is the **movement** through that possibility space.

Composer and computer scientist J. L. Alty[11] describes both listening to music, and composing music as a journey through a series of states in a possibility space in Figure 9.6. The listener remembers the past states of the music through

long term and short term memory (LTM and STM) to compare the current state to past states within memory (such as repeated motifs within the piece, or tropes in past musical experiences) and to predict possible next states in the music. The composer does something similar, but must choose the next state instead of waiting for it. Both listener and composer have a limited temporal **horizon** across which they can predict or remember states.

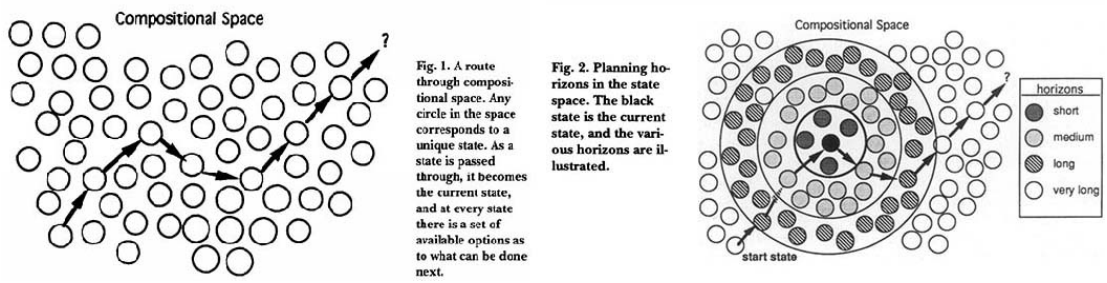


Figure 9.6: Movement through states in a musical work (in a field of potential next states), and the *horizon* of memory and prediction from a single state[11].

Alty recognizes that it is the *limitations* of the possibility space that make it possible to enjoy it or move reflectively in it. In a truly overwhelming space, where any notes could be followed by any other notes, we would have no way to understand our movement through the space. We need *some form of structure* in a possibility space for users to make sense of their movement through it. This structure can be intrinsic to the space itself, or it can be artificially-constructed through the casual creator's controls or feedback. What do I mean by "structure"? And how do we user structure to understand spaces?

To find out how people map *physical* spaces, Kevin Lynch[251] spent 5 years in the 1950s interviewing residents of Los Angeles, Jersey City, and Boston with questions about how to get to and from different landmarks in their city, or how they recognized pictures of locations in the city. From this, he derived five elements that made navigation (and spatial understanding) possible:

- **Paths:** "channels along which the observer customarily, potentially, or occasionally moves"
- **Edges:** non-path linear objects, which can be physically-limiting boundaries (a wall or river) or a seam between a "here" and a "there", like the line between sidewalk and grass.
- **Districts:** something you can be "inside of", with some "common identifying character"
- **Nodes:** "strategic" spots, or "concentrations" of importance, where many paths converge
- **Landmarks:** an orientation point that the observer does not enter, but looks upon (or sees from far away)

These landmarks have continued to be useful in the navigation of virtual spaces. Darken and Peterson applied Lynch's navigation insights to virtual worlds, first as hierarchical structure:

> "1. Divide the large-scale world into distinct small parts, preserving a sense of "place." 2. Organize the small parts under a simple organizational principle. 3. Provide frequent directional cues" [117]

Then later, they expressly endorse Lynch's paths, edges, districts, nodes, and landmarks as important in navigation, and something that should be elevated, as necessary, to the status of UI element in virtual maps[116]. I've never found confirmation that the designers of *World of Warcraft*[s244] knew about Lynch's categories or had read Darken and Peterson's influential argument that it should be applied to virtual space, however, *World of Warcraft* acts as a clear example of these principles in action. Players can travel along a variety of land, sea, and sky paths (or use them as navigational aides). There are clear edges where regions end

or begin, or where non-navigable terrain (ocean, cliff) occurs. Nodes are elevated to central gathering spaces and junctions between path types, and frequently are accented with landmarks. Other landmarks are placed at smaller nodes, while some are large enough to be seen from most of the paths. These large landmarks act as the "weenies" to pull the player forward to new regions (a term from the designers of Disneyland, another intentionally-constructed space[345]). Districts are particularly well-represented in *World of Warcraft*'s regions: the Teldrassil area is a forest colored in dark purples and blues, with tall redwood-like trees, while Durotar is a desertlike environment in reds and orange, with scrubby trees and large sandstone rocks. Even the creatures are shaded in hues appropriate to their region. An experienced player dropped into any region could immediately know where they are, even before they spot local landmarks (a property also true of Disneyland's regions).

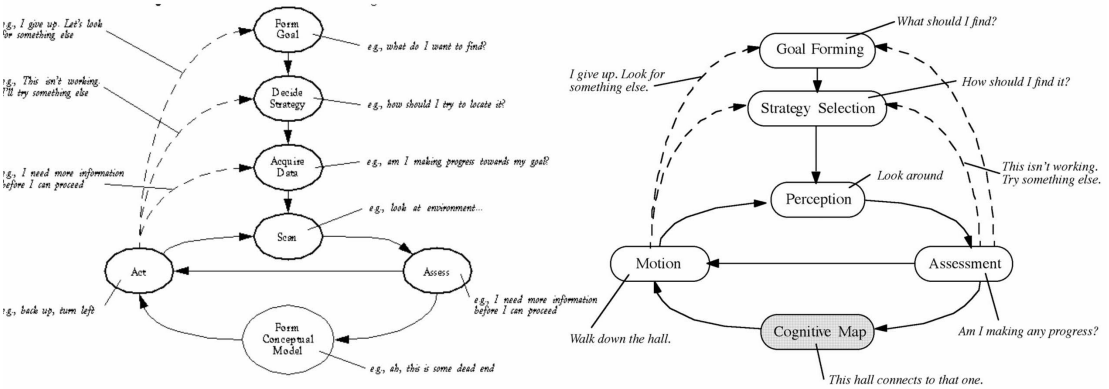


Figure 9.7: The loop of navigation from Darken's later work[116](right), adapted from Jul and Furnas[212]. The navigator continuously acts on hypotheses and updates their mental map with the results. Compare this to the grokloop and you will find a very similar structure. These loops-of-learning haunt every chapter in this dissertation.

These frameworks were created to represent physical human-navigable cities, before being successfully extended to virtual cities (still navigated by walking humanoid avatars). I believe that they may be extended to help users navigate

and build a mental model of *any* kind of virtual space, including the possibility space of a casual creator.

One technique is to *subdivide* the space. This technique divides up the potential possibility space into discrete districts, such that any point in the space belongs to one district, hopefully maintaining Lynch's definition of a "common identifying character" within the districts. In *Petalz*[s167], a *Facebook* flower-breeding game with generative flowers[323], this is done automatically with a self-organizing map. The flowers are clustered according to either their artifact image or "phenotype" (compressed to black and white, for some reason), or according to their model or "genotype". This resulted in districts that may or may not have a common identifying character: "While such genotypic clustering sometimes produces meaningful clusters (e.g. Species 1), often flowers are grouped together that look significantly different (e.g. Species 25, Species 69)." But even with imperfect clustering, once the space is divided up, the game can implement mechanics around this new structure, which provided meaning and goals to the players.

> "(W)ithout the objective of collecting a flower from all 80 SOM-determined species in Petalz, a user has little purpose beyond indulging their own aesthetic preferences when breeding flowers. Although the success of open world games demonstrates that not all players require explicit goals, many players do enjoy the concrete purpose they provide."[323]

The *Dryad*[s56] interactive tree-modeling project lets users tweak several sliders to create a large range of virtual trees, a pipeline in which parametric tuning is used to drive a Lindenmayer system[401]. Like *Petalz*, this space contains a vast number of possible artifacts, though in *Dryad*'s case, many of the trees are too odd and ill-shaped to be "good" artifacts. They solve this issue by tracking which trees the human users save, with the assumption that human interest is a good measure of quality. From that data, they perform clustering on the saved trees to identify a number of "landmark" trees. They only use this landmark

data to sample better trees from the remapped tree-space, but at least one other project extended the approach to also "preview possible geometric realizations in an empty region around the embedded points" in a simple interactive editor[16].

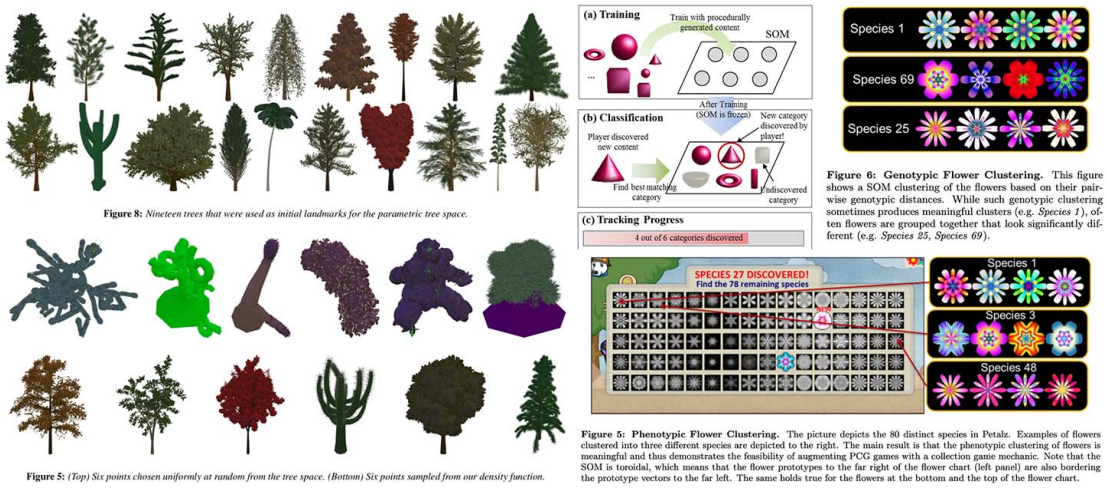


Figure 8: *Nineteen trees that were used as initial landmarks for the parametric tree space.*

Figure 5: *(Top) Six points chosen uniformly at random from the tree space. (Bottom) Six points sampled from our density function.*

Figure 5: Phenotypic Flower Clustering. The picture depicts the 80 distinct species in Petalz. Examples of flowers clustered into three different species are depicted to the right. The main result is that the phenotypic clustering of flowers is meaningful and thus demonstrates the feasibility of augmenting PCG games with a collection game mechanic. Note that the SOM is toroidal, which means that the flower prototypes to the far right of the flower chart (left panel) are also bordering the prototype vectors to the far left. The same holds true for the flowers at the bottom and the top of the flower chart.

Figure 6: Genotypic Flower Clustering. This figure shows a SOM clustering of the flowers based on their pairwise genotypic distances. While such genotypic clustering sometimes produces meaningful clusters (e.g. *Species 1*), often flowers are grouped together that look significantly different (e.g. *Species 25*, *Species 69*).

Figure 9.8: *Dryad*'s possibility space contains many aesthetically displeasing trees, but by identifying user-preferred landmarks (top left) they can identify "good" spaces (bottom left). *Petalz* divides its space in to computationally-distinct regions (top right), which are sometimes aesthetically related, to make a continuous/toroidal 2D-remapped possibility space of 80 collectible "species" (bottom right).

These few projects have not yet fully explored the ways that Lynch's framework can be used to annotate and explore the space. For example, if the space has landmarks, would a path between two landmarks be a *blending* function? Do edges represent limitations in the space, such as a limited size or color-range, and could that be visualized? Often users in these systems explore only in a limited horizon of local known-good solutions; could "edge" artifacts be used to show them the limits, in order for the users to feel more comfortable leaving a local maximum? Nodes, in Lynch's view, often function as social hubs where many users gather. In a possibility space, would nodes represent a part of the possibility space that has a lot of recorded and visualized creative activity, so that a user could "see" the crowd around that point? I explore some of these

techniques further in chapter 11, but they remain so far largely unexplored.

For many of these techniques to work, we need specific *kinds* of possibility spaces. One key property is **locality**, the principle that similar models will produce similar artifacts. This is necessary for automated navigation like hill-climbing or evolutionary algorithms; if we want to "breed" an artifact to be taller by producing "offspring" with slightly different models (or genotypes), we need to know that its offspring will also likely be of a similar height to it, with one perhaps being a bit taller. For some types of automated exploration (evolutionary algorithms, locality improves the speed of searching the space, except in rare cases[329]. For humans, as Alty noted, we have a limited horizon across which we can remember or predict patterns. In a possibility space with good locality, a user can tweak their model (moving from one point in the possibility space to a nearby one), evaluate the properties of the resulting artifact, and make a good hypothesis about what to expect by moving it further in that direction. If the pipeline is simple enough that linear changes in the model create linear changes in the artifact (say, a slider of petal-length for a flower generator), then that part of the possibility space becomes **extrapolatable**. The user can predict what's at the extremes of the slider without going there.

"Predictability" is not always the only correct design goal, though. In *Spore*[s215], the rigblocks (3D artist-modeled components like hands and eyes) had multiple sliders. These sliders *mostly* were extrapolatable, and had continuous locality, and controlled easily mapped values like the length of a muzzle or the size of a paw. However, the sliders *actually* controlled not a mathematical scalar, but a bone-rigged Maya animation, which the artists could use to "animate" the rigblock from one slider position to another. They were able to "hide" surprises in parts of the possibility space for the player to discover: when the muzzle was extended

to its limit, teeth would suddenly pop out, changing the character dramatically, and when the paws were extended, previously-unseen claws would emerge. A rare quality in a possibility space is **continuity**, where as two models get close to each other, their artifacts do as well, for any amount of closeness. This property allows the smooth blending of artifacts for animation, but is very difficult to achieve with many pipelines (anything with text is *extremely* discontinuous, for example, as are content selection, grammars, and tile-based methods).

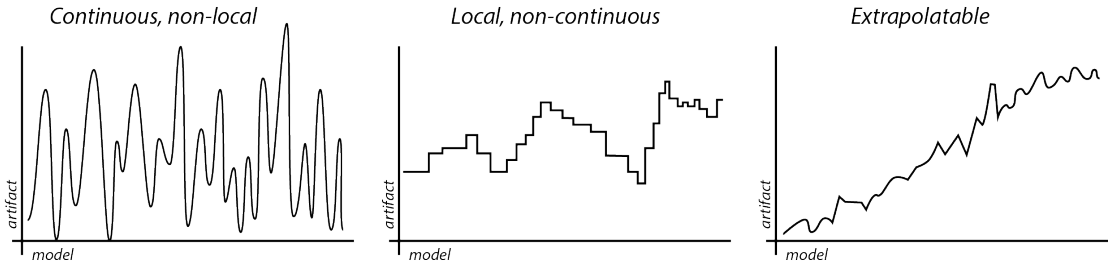


Figure 9.9: Not every local space is continuous, not every continuous space is local. Navigation qualities can differ over short and long scales, and can be different for different parameters of the model.

Once the user is comfortable and can navigate in a possibility space (a hard enough task!), the designers of the space may feel free to add mystery and surprise. As Lynch puts it:

> "It must be granted that there is some value in mystification, labyrinth or surprise in the environment. Many of us enjoy the House of Mirrors, and there is a certain charm in the crooked streets of Boston. This is so however, only under two conditions. First, there must be no danger of losing basic form or orientation, of never coming out. The surprise must occur in an over-all framework; the confusions must be small regions in a visible whole. Furthermore, the labyrinth or mystery must in itself have some form that can be explored and in time be apprehended. Complete chaos without hint of connection is never pleasurable."[251]

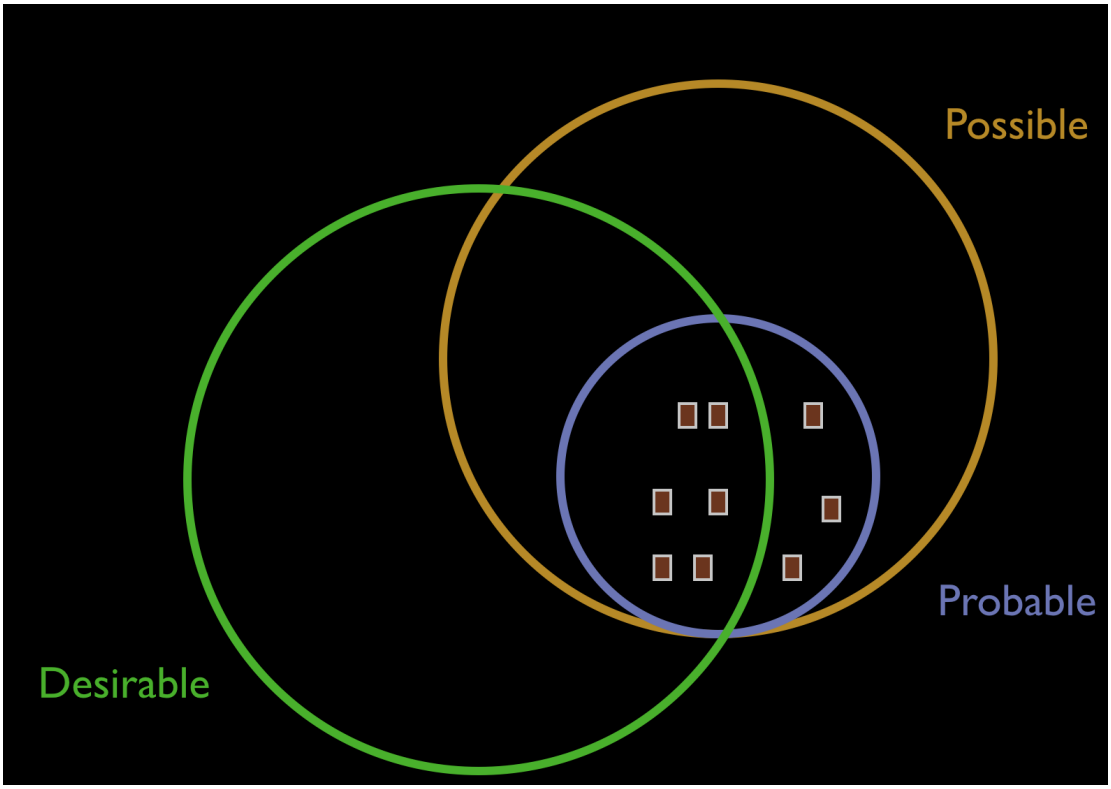


Figure 9.10: Chaim Gingold's presentation at GDC 2007 illustrated how a designer of a generative system carves up the possibility space with their generator into which artifacts are Possible, Probable and Desirable. Some bad artifacts may be *possible* to generate, some *desirable* artifacts may be *impossible* to generate, but the designer should build a system such that it generates some *probable* varied selection of mostly-good artifacts.[158]

Hard and soft failure analysis

The final way to analyze a generative space is by considering the *texture* of an array of generative artifacts. Are there a few good artifacts among many subpar ones? Are the good ones all of the same perceptual type? Are there hard failures?

Every generative system will have its own set of **hard and soft failures**. Soft failures happen when a system produces artifacts that are unpleasant, boring, ugly, or unfun. A soft failure can be met with the user's shrug, and moving on to the next artifact. By contrast, a hard failure is a disaster that ends gameplay or

ruins the experience for the user. For generated content in games, this is often a gameplay failure, like an impossible level, or a place or state to get stuck in.[6]

Hard failures feel unfair, and importantly, inescapable. Because of that, hard failures can sometimes be converted to soft failure by letting the user undo, escape, or reroll the content, or by giving them the impression that they *could* have done so with better choices, as the game generation for the destructible-terrain platformer *Spelunky*[s212] demonstrates:

> "Destructible terrain not only gives the player more agency, making the game more immersive, but it also makes the level generation easier, since the developer can safely generate areas that are walled off. The level generation system I ended up using for Spelunky depends on your inability to access the entire level without destroying the walls."[444]

Other forms of hard- to soft-failure de-escalation include:

- Generating player identities (name or appearance) can go from intentionally rage-inducing[169] to humorous [419] by adding a re-roll button
- *Spore*'s planet name generator had an immense blocklist of "bad words" to avoid generating offensive names, but would still generate odd names, which players learned to change with the cheat console[311]
- *Spore* initially promoted simulated consequences for creature designs (too short? No high fruit for you!), but when early users were getting weird with the Creature Creator editor, it was decided that hard gameplay penalties for the "wrong" creativity would quash the freedom they felt. When it shipped, the editor would not let the user exit without placing a mouth, the only part necessary for a creature to not slowly and boringly starve to death.

[6] I once fell into a crevice on a *World of Warcraft* region and had to figure out how to laboriously injure my character to resurrect at a non-stuck space, and that was on hand-authored terrain.

- *Keyforge*[s112], a generative card game with generated names invited a hard-fail by including politically and personally sensitive nouns in its generator. When it decided to recall them, it chose to ban them from play, rather than giving players a choice, which turned a hard failure into a harder-failure (Bad deck roll? Enjoy the mandatory recall shipping process). [427]

- Cultivating a delight in unfairness, and a community around sharing failure, making spectacular failure into a combination of Greek tragedy and spectator sport, like *Dwarf Fortress*[s58], *Caves of Qud*[s30], and *Desert Golfing*[s47] [139][422][333]

All of these examples are from games, which may be because games are a space that generates content with hard constraints, but which also restricts interaction with the generator, preventing the user from easily moving past bad artifacts. Design-focused generators (most casual creators) provide many failure-softening features like "undo" or sliders (which implicitly have "undo" built in). Those generators which have a hard-failure state, like 3D printing software that must make a watertight mesh with certain physical properties, aggressively constrain their generative space to one with no hard-failure artifacts, even to the point of losing potentially-exceptional artifacts to do so (as in systems like Nervous System's *Cell Cycle* and *Kinematics@Home* apps to design 3D printable jewelry[s146]). Any social space also has the ability to turn soft failures into hard ones, as even deleting broadcast tweets does not "undo" their publication to the community , as Microsoft saw with the generative chatbot *Tay*[s227][425].

Fortunately many hard failures can be calculated (not the social ones, though, as any large social media company has discovered). Soft failures, however, are almost always a failure of aesthetics, or preference, opinion, or a disappointment in the quality of the generated content. The "10,000 bowls of oatmeal" problem

falls into this category. To deal with this content-quality issue, we need tools that let us address the "texture" of a generative space. Are there large areas of poor-quality or broken artifacts, or are the bad artifacts very close to good artifacts? Likewise, where do we find exceptional artifacts: are they near good quality artifacts, so that users can follow a gradient to local maxima to find them, or are they near bad-quality artifacts, requiring a stochastic process to encourage users to search "bad"-space? Expressive Space analysis and tools like *Danesh* can tell us about properties of artifacts, but not about human heuristics and not about one of the most context-dependent ones, novelty.

Novelty analysis is the one of the hardest ways to analyze in generative spaces, as the definition of "novelty" is so deeply linked with the user's subjective individual tastes and past experiences. In the fields of computational creativity and artificial intelligence, interest is building in "novelty"-seeking algorithms, which optimize for artifact difference, rather than some fixed heuristic.[234][324]. Unfortunately, while this can find difference for calculable properties, it is not a solution that can handle perceptual novelty, novelty as experienced by users. In *Image of the City*[251], Lynch provides a few frameworks of novelty or memorability in addition to his better-known framework of edges/nodes/landmarks/districts/paths. He calls this property "imageability", "legibility", "visibility", or "apparency" (from art theory). In my blog post about generativity[73], I call it "characterfulness", the idea that an artifact is so distinct from its background and neighbors that it becomes a memorable character, in the way that undefeatable sprout in "Wall-E", the husband-and-wife chairs in Pixar's "Up", the tumbledown Ugly Duckling bar in "Tangled", or the mix tape in "Guardians of the Galaxy" become characters[7].

[7]Accidentally, these examples are all from Disney properties, as Disney is *really* good at characterfulness.

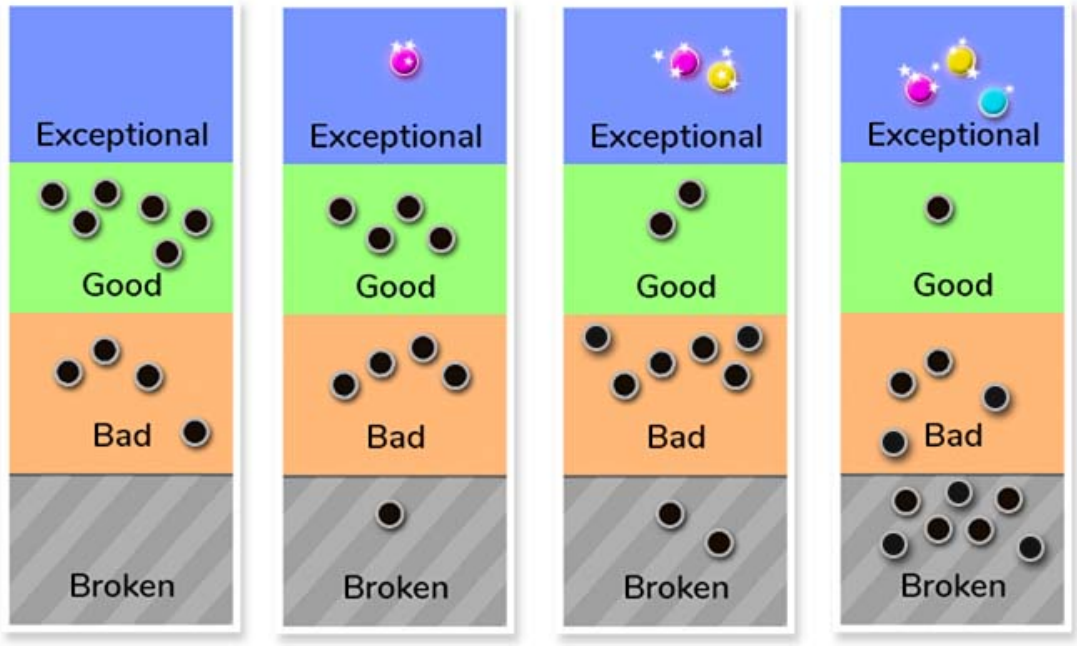


Figure 9.11: Different kinds of distributions are acceptable in different use cases. The far left example, with no broken artifacts but no exceptional ones, would be good for an on-line generator with no filtering, where hard failures are unacceptable (like an impossible platformer level) but it's ok for the artifacts themselves to be unmemorable (*Diablo II*[s48] map generation falls into this space). A space like the one on the far right, with a rich exceptional space among many bad and disastrous artifacts can be an exciting space for users to search for, and "own", the gems among the dross. Likewise, the middle two columns are common distributions, with a few broken outcomes and a few exceptional artifacts. These would require some filter, either by the user or the system, with a reward of the occasional rare find.

Lynch provides a list of several properties that create a feeling of "imageability" in city landmarks or paths:

- **singularity**: "some aspect that is unique or memorable in the context", e.g. a Gothic cathedral among glass boxes, a sharply man-made object in softly curving nature, or vice versa,
- **form simplicity**: a clearer, more memorable shape than the surroundings (the Eiffel tower, Singapore's Gardens by the Bay)

- **continuity**: blended-in or attached to the surrounding area (like Falling Water or New Zealand's Hobbiton)
- **dominance**: larger, brighter, more complex
- **clarity of joint**: a point of clear contrast between regions or parts, a cliff or sea edge, or a Victorian house with Thai Buddhist temple ornamentation
- **directional differentiation**: going in a contrasting direction (like a sharp spire on a flat piazza)
- **visual scope**: opening visual ability (windows, overlooks, buildings on stilts)
- **motion awareness**: creates ways to mark or view directional motion, like overlooking a highway or river, or providing directional clues on a one-way street
- **time series**: repetition as experienced temporally by walking past it (identical columns, lampposts, piers)
- **names and meaning**: as we will see in chapter 24, there are many kinds of semantically-heavy words or descriptions ("Diamond District", "Elephant and Castle", "Little Italy", "Rockefeller Center") that link abstract places to concrete socially-meaningful interpretations

Not every object in a city, or every artifact in a generative space *should* be characterful, though! Just as a cast of drably-dressed extras help a film character stand out(Figure 9.12), so can a space of non-exciting generative artifacts highlight a few memorable ones.

In the blog post, I propose four levels of uniqueness:

- **identical**: if two artifacts were juxtaposed spatially or temporally, the observer would notice that they were identical (even if they had subtle mathematical differences)

- **perceptually differentiable**: the observer would see that they are different, even if related, but after some time, might not be sure that they had seen that particular artifact or another one like it before

- **perceptually unique**: if asked, the observer *wouldn't* think that they had seen this before; it feels *new* when encountered

- **characterful**: this artifact is memorable. If asked about the system later, the observer would mention it, and might even give it a name.

In these categories we see "novelty" mean different things across long-term or short-term memory, in or out of the system. Non-identical things can be perceived as such across very short timescales, but after a few moments, we can only tell whether they are perceptually differentiable. If multiple artifacts are visible at a time (like trees in a field), they need to be differentiable, otherwise it will be felt that they are copy-pasted identical clones. If they are presented sequentially, longer than the short-term memory of the specifics, they can be identical without the observer noticing. If the system repeatedly shows artifacts that are only *perceptually differentiable*, the artifacts will start to feel very similar, and collapse into an experienced category, rather than individuals. If we want the observer to feel a sense of difference or progress (such as when exploring a generative space in a casual creator), then the system needs *perceptually unique* artifacts (or at least *perceptually unique* categories of *perceptually differentiable* artifacts). If we want the observer to leave the system with memories of artifacts that they will want to tell others about, then some artifacts will need to be *characterful*. When users

retweet a twitterbot's posts, they often do so because one post stood out from the others, by accidentally telling a story, making sense, or creating a punchier-than-usual juxtaposition.

Like the rest of the properties of possibility spaces, the goal is not to achieve one or the other form of novelty, but a distribution of long- and short-term novelty that is right for the use case.



Figure 9.12: In a movie example of a procedurally generated scene, the directors cast twins and triplets in the background of a scene in The Matrix to make an argument about not noticing small differences when presented with a much larger difference. One character is *characterful*, the others are *differentiable* but not unique. The character who created the scene mentions "the woman in the red dress" after Neo exits, and asks him if he saw her (the characterful artifact is *named* and *externally remembered*).[92]

In this chapter, we have seen ways of describing how generative pipelines create possibility spaces by turning **models** into **artifacts** directly or indirectly, and how one domain could be modeled with many different data structures, depending on the interaction affordances needed by the system. We saw ways of analyzing the **quality** of possibility spaces, from looking at the expressive range of artifacts, to the navigational qualities of the space (**locality**, **continuity**, and **extrapolability**). We saw ways of annotating or making sense of spaces (real-

world, virtual, and generative) with nodes, paths, edges, districts, and landmarks. We explored ways of talking about novelty, noting that spaces can have **differentiable**, **unique**, and **characterful** artifacts within them, but that this is emphatically *not* a mathematical calculation of the combinatorial properties of the model, but the perceptual experience of the artifacts. Hopefully, the lesson taken away from this chapter is that there are many potential shapes and qualities of possibility spaces, even for a single creative domain. In the next chapter, I will demonstrate that even narrow or shallow spaces can be a part of casual creators.

Chapter 10

Narrow and Shallow Possibility Spaces

10.0.1 Simple spaces

Many casual creators do not have a vast possibility space of generative artifacts. I don't think we can put a limit on *how many* artifacts the space needs to have in order to be a casual creator, as some very related apps exist on a spectrum from not-enough to definitely-enough possible artifacts. As well as the actual mathematical *number* of artifacts (a function of the model used to create them), we may also be concerned with the *variety* (perceptually different or characterful) artifacts in the space.

A **narrow** space has few choices for the user to make, or few options for each choice. A photo-editing app where the user selects a photo, and selects one of ten filters, is *narrow*: even if the results of each choice are meaningful and each artifact has a lot of different character, there are only a few possibilities to choose from and the space will not take long to explore. These are not necessarily *bad* casual creators, it just means that they will need autotelic pleasures *outside* of

fast exploration.[1]

In a **shallow** space, there are many choices to make, but none of them have a particularly distinguishing effect on the resulting artifact. In *WebGL Lathe*[s240], the user chooses one of several materials to whittle, then virtually carves away the material. This system has many choices (the order and location to lathe away material), but all of the artifact are variants of a lumpy spindle. This is *also* not necessarily a bad casual creator. The pleasure of exploring this space is not in discovering interesting artifacts, but in making many, many small choices continuously, creating flow. Shallow creators will often focus on tactile flow-enhancing pleasure: glitter, soothing sounds, continuous pleasant feedback, or autonomous sensory meridian response (ASMR) triggering sound and visuals[13].

We can see many historical examples of these kinds of possibility space in embroidery kits, paint-by-number kits[s159], and model-assembly kits. These kits have many things in common with casual creators, in that they are sold to a non-professional audience as a way of participating in creative activities. They exist on a spectrum from giving the user no choices, to no creative or consequential choices[2], to small but interesting creative choices. "Paint-and-sip" businesses are a current fad where patrons drink wine while following instructions to create a simple painting[27]. The possibility space is shallow, but each patron ends up making slightly different brushstrokes or color choices to make their "own" painting, while also enjoying the sensory pleasures of physical painting, wine, and friends.

We could try to categorize these systems and experiences as "casual creator"/"not casual creator" based on their creative possibility spaces, but it can be

[1]There may also be many non-creative choices (order of action, how long to wait between acting) that don't affect the artifact.

[2]Such as letting them choose the order in which they fill in each paint-by-number space, but not the colors.

more helpful to identity how these maybe-casual-creators add in ownership and surprise to a space that would seem to preclude that.

10.1 Four simple-space creators

Let's examine four "simple" casual creators from the Android app store.

- *Pixel Art: Color by Number* (Easybrain)[s173]
- *Pixel.ly 3D* (Color Apps For Free)[s174]
- *DiamondArt*[s49]
- *Super Sequin Simulator - DIY Flip*[s224]

As a reminder, this is the definition of a casual creator:

> A casual creator is a system that
>
> - privileges enjoyment of the creative process above productivity
> - produces artifacts within a limited-yet-meaningful possibility space, enabling automation and support, both passive (encoded into the domain model and system constraints) and active (responding to user actions)
> - supports a state of creative flow by restricting choice and preventing hard failures while allowing rapid iteration
> - results in the user's feeling of pride and ownership toward the produced artifact, and sense of pride in their own creativity.

Pixel Art: Color by Number

This is a Google Play Store "Editor's Choice" winner, so of the many color-by-number apps (Figure 10.3), we can guess that it is a better-than-average example. The user selects a pixel-art picture to color, then selects a numbered color from a set of pre-defined colors, and taps or drags across the pixels to fill them with

the selected color. There is a subtle visual cue if they have colored in the wrong pixel, but it is not prohibited. What are the users choices here, and what is the possibility space? First there is selection of the pixel object that the user selects (from hierarchical sets labeled "Nature", "Weird", etc). They also choose the order in which to color in the pixels, both which color and which area of pixels they want to pursue first. This *technically* makes a possibility space of many partially-colored pixel objects, though this space is not perceptually differentiable in a meaningful way, nor are users encouraged to explore it (just to move forward). The user can also select a photograph to make into a pixel art image to color, however, there is no interaction enabled to control how the image is posterized and pixelated (the process washed out a very bright photograph I fed it). While not a very emergent choice, "content selection" is still a rudimentary generative method, and being able to select a photo from the phone's camera roll (with more than usual personal and semantic meaning) increases the emotional significance of that artifact.

Possibility space: extremely narrow, limited to the selection of image or photo, though each image is characterful (either by being hand-designed around recognizable tropes like "poodle" or "sailboat", or by being a photo from the user's memory).

Strategy: It advertises itself as "excellent for relaxation", and leans hard into flow-inducing properties like limited choices and clear actions, as well as some computational support like auto-filling the pixels. This app is perhaps the outermost boundary of how *few* creative choices a user can have. While the user can feel accomplishment at finishing a picture, their creative input was very, very limited. The picture-to-pixels feature is a familiar form of generative automation in casual creators, though it suffers from lack of interaction[3]. Other apps in this

[3]I put in a picture of myself with bright pink hair on a blue sky. It muted the colors into

genre lack the choose-a-photo feature, and have no creative choices at all except the choice of pre-provided picture[4].

Pixel.ly 3D

Pixel.ly 3D is remarkably similar to Pixel Art, only the coloring is done on a 3D voxel object. The user selects a voxel object, then selects a numbered color from a set of predefined colors, and taps or drags across a 3D voxel object to "color in" the cubes. As they do so, all the cubes they touch spin around in a very juicy and responsive animation, but only the cubes that with the right number will accept the color. The biggest creative choice is that the user can change the predefined colors, which changes the already-applied voxel colors automatically.

Possibility space: Mathematically large (users can choose from a 3D colorspace color for each color in the set), but shallow. The range of ways to color a voxel flower has not many *perceptually unique* possibilities, and there is no automated or domain-model support built-in for finding good color combinations, so there are many bad artifacts in the space.

Strategy: This app clearly privileges enjoyment over productivity, has some automation and support (juicy animation and prevention of coloring errors, but no color support), is intended to induce flow with a stream of restricted choices, and in a small way, celebrates the user's achievement (once completed, the voxel object is displayed in a frame in the menu, in their chosen colors).

---

muddy middle-values, but I could not edit the result.

[4]As this choice of picture is often the *only* choice in these apps, the app itself may chosen, from among the 100s of color-by-number apps, for the type of pictures it has. Many of these apps advertise cats, Bible scenes, Disney Princesses or Pokémon art, to appeal to different customers.

DiamondArt

Like the first two, this is a color-by-number app, in which the user selects one of several templates of cute tropes (corgis, foxes, roller skates) to fill in. Instead of a grid, it has differently colored circles which can be filled in with selected colors by tapping or dragging. However, instead of filling the circle with a flat color, it is filled with a round rhinestone image in that color, which the user can swap for differently shaped rhinestones, flowers, pearls, buttons, or stickers (30 kinds in all).

Possibility space: A narrow and shallow possibility space of few perceptually-unique artifacts and narrow choices (one choice per circle). I completed a fox image, but felt only a small difference when I changed between rhinestones and pearls[5].

Strategy: The shallow possibility space does not make this a bad experience; indeed, the safety of it (no artifact in the space is either too good or too bad) lowers the performance anxiety and risk of experimenting with different bead types. The pleasure of the graphics (shiny beads and rhinestones) adds some interest, but there are no automated features or digital enhancements (like the phone-accelerometer glitter of *Super Sequin Simulator*) to augment the experience or add surprise.

10.2 Super Sequin Simulator - DIY Flip

I chose this app as an open-ended (yet still shallow) version of the click-to-color apps. In this app, users set the two contrasting color sequins in a piece of virtual "magic sequin" fabric (a briefly-trendy kind of fabric where brushing

[5]The jeweled flowers and ornate gems that comprised many of the options were distracting as image components, so that part of the possibility space felt low-quality and I avoided it.

against the sequins made them flip and show a contrasting color), and can "draw" in the sequins by dragging across them. As in the real-world referent, the primary pleasure is in the satisfying ripple of the flipping sequins and the play of light as they move, which they cleverly simulated with shading based on the phone's accelerometer. There is also a very rudimentary version of the color-by-number apps inside of this app, where you can color in individual sequins in a pattern, but lacks the features of the other apps, or the augmented sparkle of the main app.

Possibility space: The two forms of interaction are drawing by flipping the sequins, and selecting the colors of the two sides. The drawing is very rough, so that all drawings are really just the same scribbles, but they are elevated by the sparkly shading and subtle phone-tilt interactivity of the sequins. The colors are limited to sets of fixed combinations of two colors (black and leopard-print, green and blue), or a make-your-own where you can set either color or upload a photo. Like DiamondArt, this space is full of not-very-different artifacts, but in the same way, that gives the user license to be playful and focus on the enjoyment of ruffling their "artwork".

Strategy: The artifacts have little expressive range (both drawings and color combinations) but their combination, as well as the tactile pleasures of the app, makes for a soothing flow-state exploration. The flow encourages very rapid exploration of the drawing-space, while providing motivation for an eventual, slower, exploration of the color-combination space. This app has gone to great lengths to make the sequin-flipping pleasurable (the sequins glitter dynamically based on phone tilt) which encourages the user to keep drawing, even as the drawings themselves are of no interest.

For every deep or well-considered casual creator like *Let's Create:Pottery*[s123] or the *Spore Creature Creator*[s216], there are hundreds or even thousands of much

simpler ones, like these, and even bad or abusive ones (like the hundreds of poorly-made color-by-number clones). Some well-regarded casual creators, like *Bubble-Harp*[s23], still have a shallow or small possibility space, but make up for it with an increased emphasis on sensory pleasure, digital augmentation, safety, and flow.

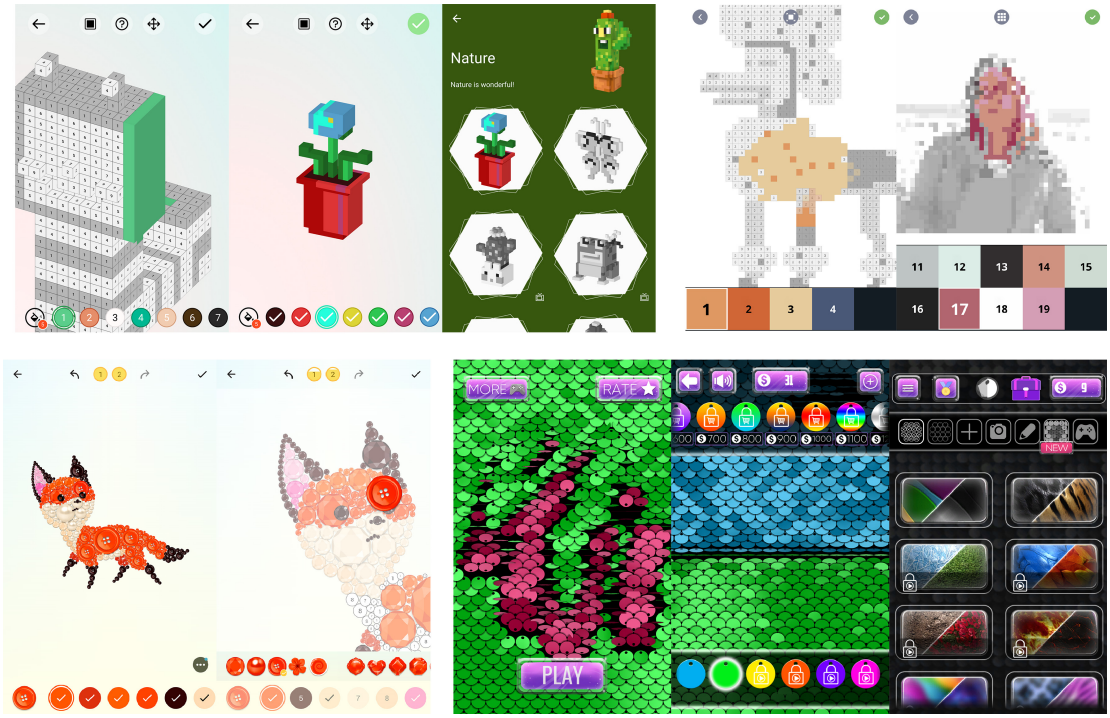


Figure 10.1: *Super Sequin Simulator - DIY Flip*, *Pixel.ly 3D*, *Pixel Art: Color by Number*, and *DiamondArt* are all shallow or narrow creators.

10.3 Strategies for simple spaces

The systems above demonstrate a few strategies for narrow or shallow spaces. Some systems have spaces that start small or simple, and later expand. Others have spaces that are narrow, but only allow a very slow exploration. Here are a few strategies for limiting or throttling the ways that users navigate small spaces.

10.3.1 Progressively-expanded tools

Casual creators often have an array of tools with which the user can edit the model. While some systems present all the tools immediately to the user, many **unlock** the tools gradually over the course of play. Examples of this tool-unlock technique can be:

- additional sliders appear for previously-uncontrolled parameters
- "discoverable" controls that aren't emphasized by the UI (the scroll wheel in the Creature Creator, clicking a name to realize that it can be edited)
- tools that lock after use and have to be re-unlocked with time, payment or task-completion
- new colors, options or ranges unlocked for existing tools
- new types of tools that have different properties

Each time that a tool is unlocked or added, it increases the possibility space. A designer of a casual creator system might choose to add unlocks for several reasons: to keep the users in a state of flow, motivate the user to keep using the tool, for the users to put greater value on a small set of hard-earned tools, or to get the user to pay money in the app. The unlock-trigger should be chosen with attention to what effect it has on the user's flow, ownership, and possibility space. Triggers may include:

- **time**: real time has passed in the game, or while watching an ad.
- **payment**: in either in-app currency, or real money (often money converted to an in-app currency)

- **task-completion**: quests/mission completed, some computational model of experience or level meets a threshold

- **discovery**: the tool is somehow *discovered*, either hidden in/by the system's UI, or by finding an object in some metagame around the system

For example, new parts for the *Spore* in-game *Creature Creator*[s215] (not the stand-alone one) unlock when the player has defeated other rival creatures, but the editor itself only re-opens when the creature has gained an in-game level. High-level parts give the creature health and attack bonuses, in addition to their aesthetic and creative functions, so this is an intentional limitation on possibility space to balance gameplay. It *should* have made these high-level parts feel more valuable as well. However when users had made a creature they liked, they would often rotate the part *into* the creatures body in order to get the bonuses without affecting its appearance. The *Creature Creator*[s216] (stand-alone) started with all parts unlocked, but parts could be added to it by drinking Dr. Pepper[95] or buying an add-on expansion set[96].

Let's Create:Pottery[s123] *also* starts with a limited set of tools (8 colors, 9 brushes). As the player gains coins by fulfilling orders (making pots to specification, quests which may require buying new tools), they can buy new brushes from sets like African, Aztec, Celtic, and Japanese, new materials, new clay types, and new parts like jug handles. The leveling mechanic gives the players a sense of progress, increases the possibility size, and due to the social-sharing site for pots, can encourage users to earn new tools to show off to other players.

These mechanics also limit the possibility space in the beginning, so that a beginner has fewer choices at the time when they have the least knowledge of the system. This encourages them to fully explore a *small* possibility space with a limited UI, rather than getting lost in options and menus (a lesson from flow

theory). This is a common pattern in many games, not just casual creators, exemplified by idle games like *Cookie Clicker*[s36] where the player is given only a single button or link to click, until they earn more UI.

Discovery of tools is less common than the other techniques, and more often happens *unintentionally* when the UI affordances are not self-evident[6]. It can also be an intentional experience; *Kid Pix*[s113] creator Craig Hickman recalls adapting the feeling of discovery and mystery inherent in using a computer for the first time into a creativity tool filled with of surprises and secrets[193]. StrangeThink's *Joy Exhibition*[s107] is a casual creativity tool where the user selects from 18 low-poly guns, then points it at a screen and "shoots" particle system paint pellets. Each gun has a different logic for what colors it uses, how the particles move, and what kinds of trails they leave behind. Thus, the experience is one of discovering a new tool by pulling a gun off the wall, then discovering all of the alien logic[7] of the tool through its use. *Art Sqool*[s7] (Julian Glander) hides the tools around a surreal 3D world, taking the user through an Easter egg hunt for new brushes and colors. Andi McClure and Michael Brough's *BECOME A GREAT ARTIST IN JUST 10 SECONDS*[s90] similarly has a set of tools (each key on the keyboard) mapped to nearly-uncontrollable glitch art techniques that smear pixels, spray a fine mist of geometric shapes, or swap colors that must be discovered through use. These games' particle brushes make me wonder about a possible procedural art game based on *Galactic Arms Race*[s81], where players compete for particle weapons generated by evolutionary algorithms, but the bullet systems paint canvases instead of destroying enemies.

[6]We joked about listing the "scroll-wheel to resize" feature on the Spore box, as so few users discovered it.

[7]At least one "gun" changes color based on how far it is from the screen.

10.3.2 Intentionally slow grokloops

Not all casual creators should be considered *good* casual creators, nor should all of the *common* patterns be considered good ones. Indeed, the existence of bad or self-defeating casual creators is one of the motivations of this dissertation. Plenty of systems for casual creativity provide an underwhelming possibility space, and some of these needlessly prolong the grokloop in order to extract time, money, or attention from the user. One common (anti-)pattern we can see in many casual creators is the need to perform rote tasks to gain access to creativity tools. This "temporal dark pattern"[445] makes the user grind through repetitive or tiresome tasks, either to pad gameplay time, or to monetize the attention of the user (ad-watching). In a non-sinister application, it can also be used to reward the user for getting through contextual information, as in the interactive virus documentary-and-editor *Virus: The Beauty of the Beast.*[s237][410]

This pattern is *very* common in mobile casual creator apps, which are almost all free (to compete in a space where all the other apps are free) and must monetize with ads or in-app purchases in order to earn money. *Make Donut - Kids Cooking Game*[s130], *Cake Pop Maker - Cooking Games*[s27], and *Real Cake Maker 3D - Bake, Design & Decorate*[s26] are examples of a very common variety of casual creator on the app store. These apps have a very simple choice-based pipeline with a few more complex steps (player-controlled 2D object placement, using user-provided photos for textures). The also pad out the experience by having many task-based steps for the user to perform through simple dragging actions (put the eggs, milk and flour in the bowl, stir, put batter in the mold, open the oven, add the cupcake pop sticks, etc). These uncreative actions keep the user interacting, and create a sunk-cost that encourages them to stick around past the barrage of ad-watching requirements in order to get to the advertised cake-decorating.

These constant interruptions (often between steps of the process) completely break flow and disrupt the learning process of the grokloop (I found it hard to remember my actions in previous steps after a 30 second ad), and even penalize using "undo", which can trigger another ad.



Figure 10.2: *Cake Pop Maker - Cooking Games* and *Real Cake Maker 3D - Bake, Design & Decorate* are apps marketed to children to get them to watch ads while waiting to use a casual creator. On the left, *some* of the two dozen steps required between creative-control stages of *Real Cake Maker 3D*.

Makers of generative systems often want to make a system with great expressive range[383] with lots of good artifacts[158]. Not every generative pipeline is capable of creating that kind of space. Whether a system is limited by the mathematics of the possibility space or its gameplay needs, there are strategies to deal with a narrow or shallow possibility space and still create a rewarding creative experience.

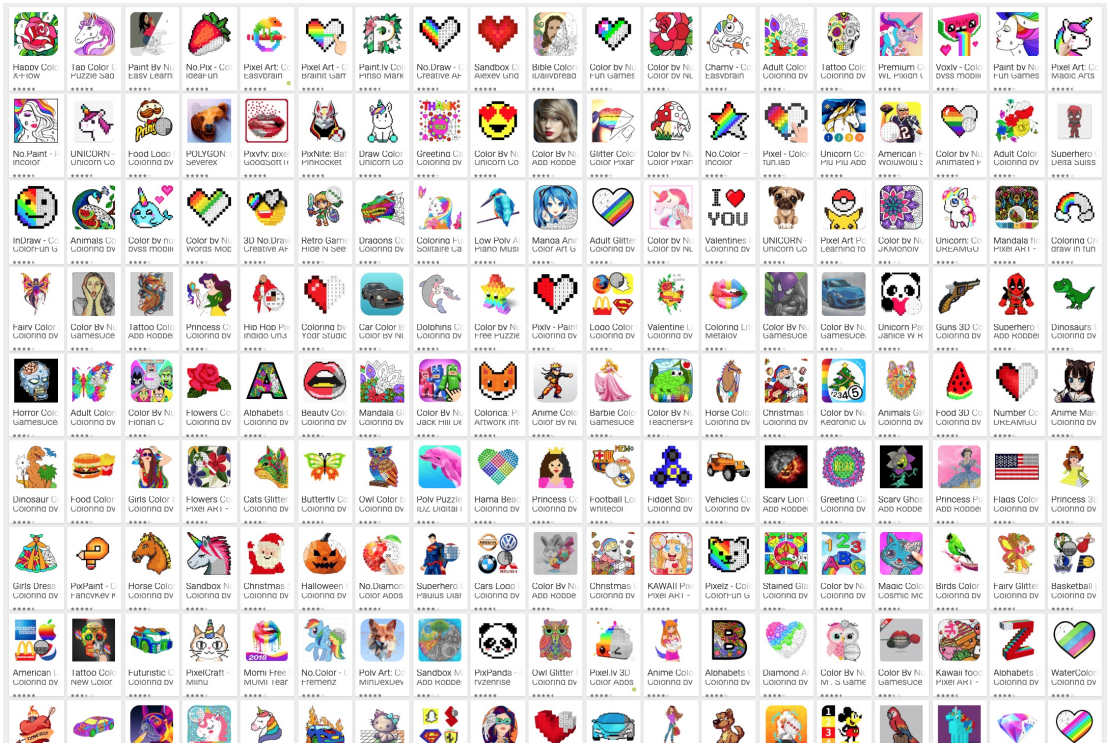


Figure 10.3: It's impossible to say how many color-by-number apps there are on the app store. This is as many screenshots as I cared to take, but the list keeps scrolling for as long as I refreshed.

Chapter 11

Parametric Spaces

11.1 "Pure" and "lumpy" generators

In this chapter, we will explore one common shape of possibility space, **parametric spaces**, which have an uncommon number of interaction affordances[1].

In addition to defining the possibility space, as we have seen in chapter 9, certain kinds of particularly-structured generative pipelines also enable new ways that we can *explore* and *annotate* possibility spaces. For example, a possibility space defined by grammars can be understood (visualized, explored) by changing the replacement rules that were chosen at each branch, or it can be shown as a tree of recursive expansions, or we could visualize the stack of expansions behind one given item in the output. There are often so many ways to visualize or interact with a possibility space that there is no clear "right" way to do so.

Over the course of this dissertation, I have seen a vast diversity of ways to structure a possibility spaces in hundreds of casual creators. I have attempted to diagram them in a variety ways, but have never found a format that can convey the *shape* of a possibility space for any useful set of them. One issue is that,

[1]This chapter was workshopped in the talks [72] and [77].

for even for quite simple casual creators, the possibility space is often not a nice mathematically-definable structure.

How do you graph the space of a user choosing a photo, or a piece of text? A mathematical abstraction ("they are selecting one photo from a roll of 100 on their phone" or "text is just a series of selections from 128 ASCII characters") doesn't usefully capture how the possibility space changes when someone selects an e. e. cummings poem, a political tweet, or lorem ipsum. All possibility spaces are *lumpy* when we consider the experiential properties that that artifact-space represents, but most are also lumpy mathematically, because they use generative pipelines that mix together different data structures to model a complex domain.

Academic generators (especially in computer science) often use a "pure" model without this lumpiness (pure grammar, pure parametric, pure choice-based) so that it can be represented in a clean data structure (a tree, a vector, a bitmask respectively) over which they can run known algorithms. Commercially-released apps (or publicly released artworks) tend to be messy hybrids with of a mix of heterogeneous generative methods and lots of interdependence, because most domains that they are trying to operate on have some edge cases or interdependence, like the *Ring Designer*'s[s190] ring model in Figure 11.1. For this reason, many of the techniques (hill-climbing, evolution, 2D maps of artifacts) from academic research cannot easily be used with these hybrid models.[2]

Fortunately one particular kind of structured possibility space has a great number of interaction and visualization affordances, reasonable cognitive complexity, and enough affordances to make up major components of commercially successful casual creators (if not whole casual creators). This is the possibility space that

[2]We wanted to implement "cross-breeding" of creatures in *Spore*, but our creature model is a tree full of distributed content-selected parametric objects. All of the different blending/crossover methods we tried (branch-swapping, averaging parametric tunings) resulted in terrible creature designs that still looked little like the parents.

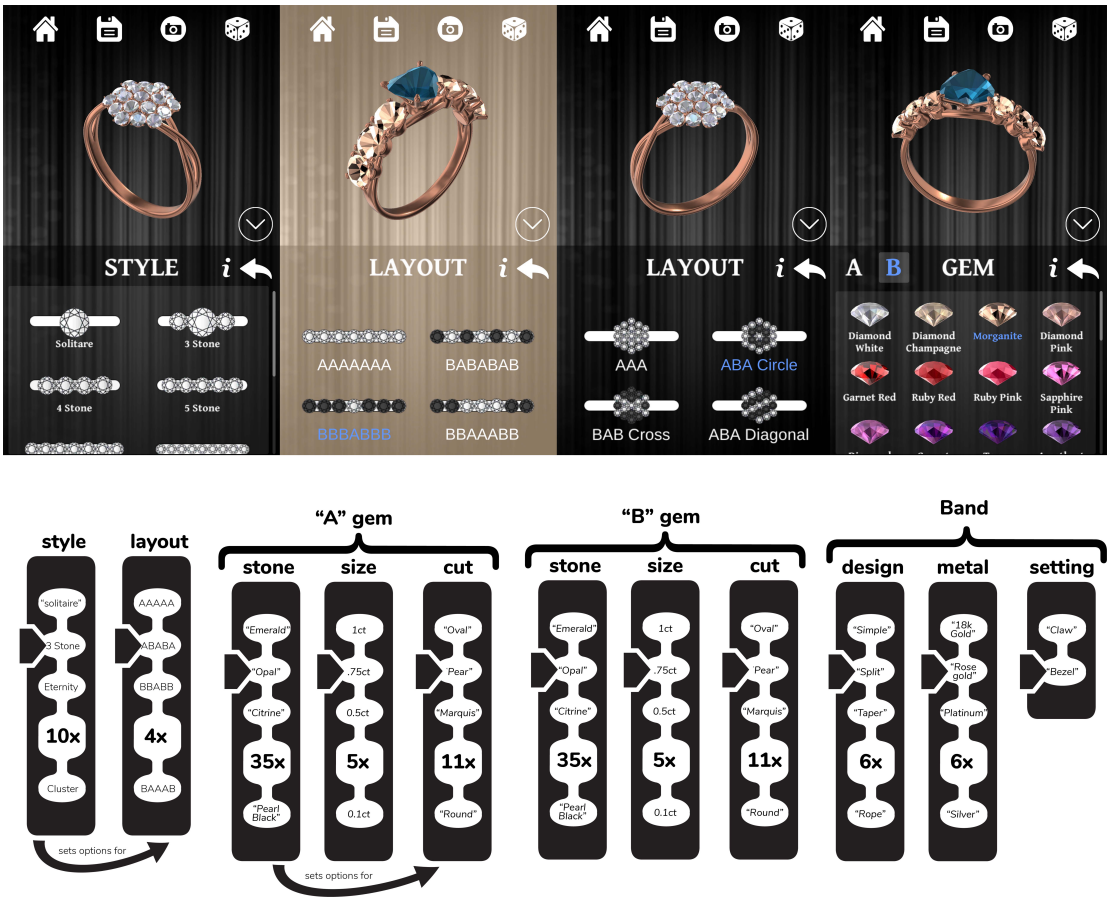


Figure 11.1: *Ring Designer*'s model of a ring. Even in this simple choice picker, some of the choices are interdependent, making the possibility space more complex than a simple combinatorial problem.

results from creating a generator which generates artifacts from an input of many numerical parameters, a **parametric space**.

Parametric possibility space

Parametric generative pipelines (see chapter 6) represent artifacts as arrays of numbers. Each number in the array represents some properties used to generate the finished artifact. For example, the shape of a virtual pot, as in **Let's Create: Pottery**[s123], can be represented by a set of a dozen numbers, representing the varying radius of the pot from its base to its lip. This representation doesn't

represent all ceramics, or even a particularly large possibility space, but it does create a very *clear* translation from the parametric data to the finished artifact.

Most parametric generators are less direct than that. For example, *Funky Ikebana*[s76] is a Lindenmayer system[242] I wrote to teach L-Systems in a generative methods class. The data of each flower is an array of about 20 normalized floating point parameters. Some control continuous independent parameters, such as the length of the leaves, hue of the base of the plant, or the angle of the branches. Some properties are relative to other parameters: one parameter controls the change in the color of the stem as an offset from the root color. Another parameter affects the length of the branches, but this is adjusted by the depth of the branches, to re-shape the possibility space so that no flower is too large. A few rare parameters are *non-continuous*: the number of petals and number of leaves are represented by continuous numbers, but are used as integers. Most parameters have discontinuous values at 0 or 1, but several (the hues) wrap around.

One interesting property of a parametric generators is that it can be modelled as an N-dimensional cube. That is, **for a generative pipeline which takes as input an array of length N, each N-dimensional vector is a unique artifact in that space, and each artifact in that space occupies a unique spatial location.** This ability to *spatially* reason about artifacts and their relationship to each other creates a number of useful affordances for navigating and visualizing the space (Figure 11.2).

A few properties of the possibility space will determine what affordances it has. We have seen these in chapter 9 as well, but in a parametric generator modeled as a cube, they also have geometric meaning:

- **continuity:** as the models get arbitrarily closer to each other, so do the artifacts

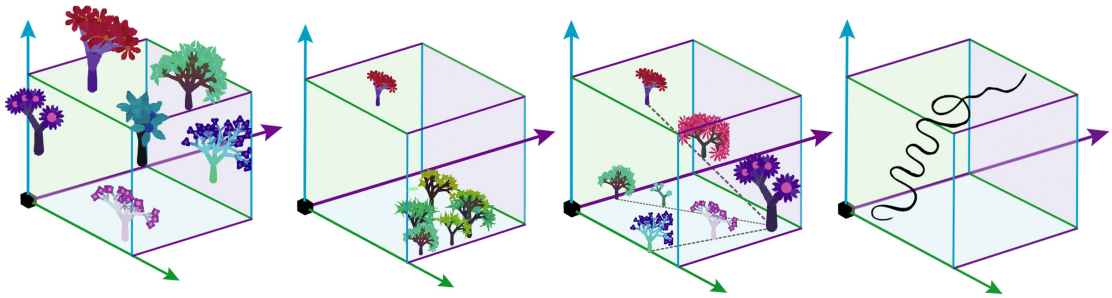


Figure 11.2: A parametric space modeled as a 3-dimensional cube (representing an N-dimensional cube). The flower generator shown is a mapping between an N-dimensional vector (representing flower properties of varying directness), which is a point in the cube, and an output artifact, a flower. Because these vectors are spatial representations of the inputs, spatial operations in N dimensions work the same way they do in 3 dimensions. We can lerp from one artifact to another, or sample points and create an N-dimensional bezier curve and sample along it.

- **locality:** similar models produce similar artifacts
- **predictability:** given a change in artifact along some direction, will continuing in that direct see the same change[3]

A space with locality enables hill-climbing algorithms and showing the user nearby artifacts. A space with continuity allows seamlessly *blending* between artifacts (essential for animation without popping). A space with predictability lets us make large jumps, in addition to small jumps. For example, Perlin noise has continuity and locality, but not predictability. Practically, it is often hard to have fully continuous generators. Some generators must have discrete options (whether to use one of three shapes as a motif, for example) and so split a continuous value into discrete "steps". This breaks the continuity of the possibility space: an artifact with "shape=.24999" may have a triangular motif and one with "shape=.25" have a circular one, and thus be quite different despite the closeness of their vector positions.

[3]The "butterfly effect" inventor Edward Lorenz says "Chaos: when the present determines the future, but the approximate present does not approximately determine the future."[114]

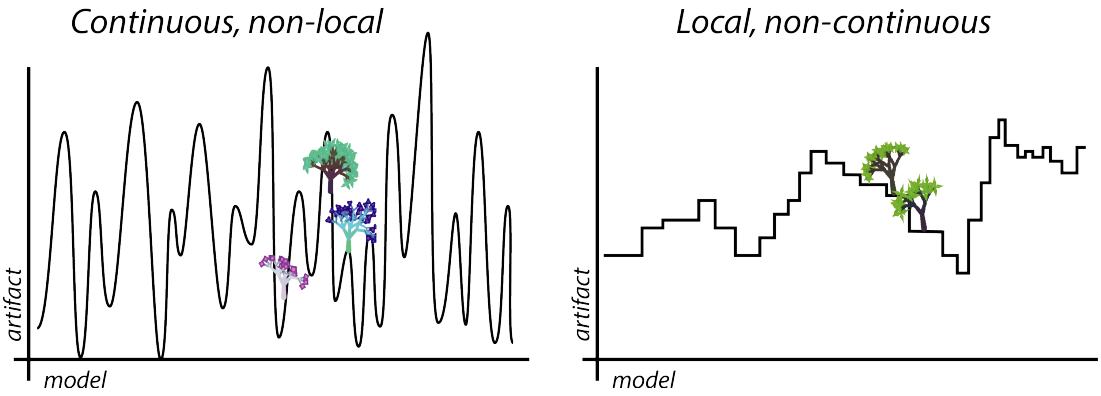


Figure 11.3: In a continuous space, models *very* close to each other will look more similar, but over larger scales, they will diverge rapidly. In a system with locality but non convergence, models close to each other will produce similar artifacts, but may have small discontinuities between them. The space is only ever *sampled* though, so technically non-continuous spaces (like fixed-precision numbers) will be perceived as continuous as long as they are local on that scale.

Note that these properties are measured by the difference in *artifacts*, not the difference in the model representation of the artifacts. The space of English words is notably discontinuous: we can represent words as sequences of numbers, but their position relative to other words in that space doesn't tell us anything about their meaning or whether they are valid words[4]. *Word2Vec* is a remapping of words to vectors such that these properties *are* maintained for word meaning[273], and *phonetic-similarity-vectors* does the same but for word pronunciation[304].

When we have a mapping of models to artifacts that preserves locality, we can play interesting games with it. Allison Parrish, the creator of *phonetic-similarity-vectors* created a simple text-field/casual creator[5] that generates star-diagrams for word encodings, as the player types, so that they can see word similarity. *Robot Mind Meld*[s191] is a game in which the user is challenged to synchronize with an AI by guessing what word is the interpolation between two given words, like

[4] "Whisper", "whimper", "whioper" would all be clustered in a letter-based representation, yet don't have any relationship to each other.

[5] `http://static.decontextualize.com/vecviz/`

"apple" and "pest" (answer: "bug"). Vector encodings with locality enable ways of interacting with and visualizing the invisible "wordspace" of language that are not available for other ways of encoding language.

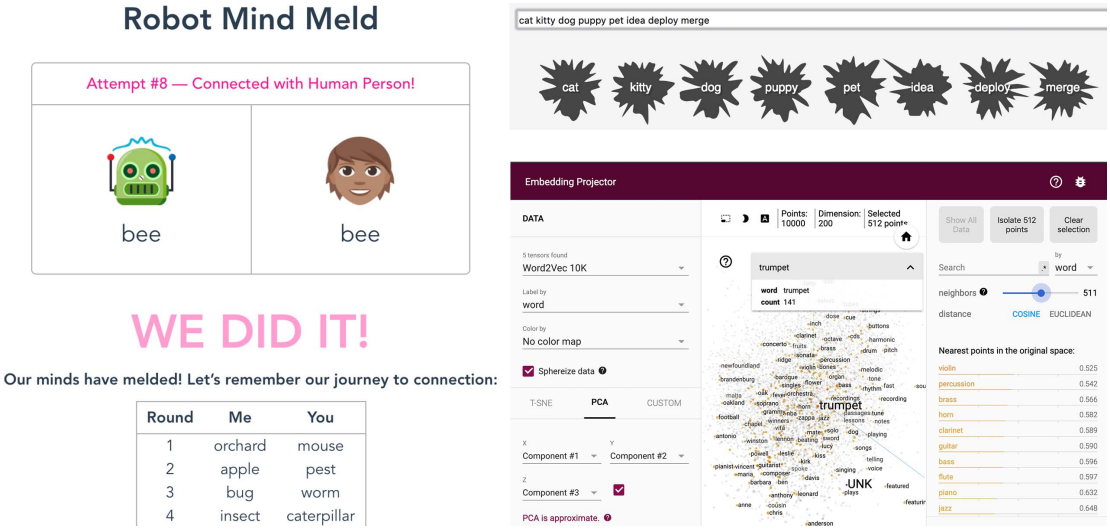


Figure 11.4: Allison Parrish's word visualization (top-right) and *Robot Mind Meld* (left) use vector-encodings of English words. Compare these to decidedly non-casual word-encoding explorers like the official Tensorflow projection viewer (`https://projector.tensorflow.org/`).

11.1.1 Displaying and mapping parametric possibility spaces

The process of building a generator is itself an iterative process within an intereactive generative system (the programming language/IDE in which you are building the generator). The same grokloop applies here as well: to iterate on a generative system, you must be able to form hypotheses, encode a change in the system, and then evaluate the result of the change. But in practice, the "gulf of evaluation" in a generative system is particularly difficult to bridge. If you have built a generator, how can you determine what the possibility space contains? How much good, bad, desirable, or novel content is in it?

The most common answer to this question is to generate large numbers of

artifacts, and browse through them, hitting refresh until you feel that you have an intuitive appreciation of what *sorts* of distribution you have achieved with the generator. But this approach can be **tiresome, misleading or insufficient**. Some artifacts may take time to understand - a user may not be able to appreciate them at a glance. Game levels, for example, may require play-testing to understand whether or not they are good levels or what kind of feelings they achieve. Generating content randomly may also mislead a designer, as they modify their generator: one randomly generated set of artifacts may be good, and the next set might be bad, from the luck of the draw, and so may not tell you, as a designer, whether your most recent edits have improved your possibility space.

Because of these issues, there is active interest and research occurring to improve the ways that we can *automatically visualize* the output of generators. Gillian Smith's[383] concept of "expressive range" seeks to map salient properties of generated artifacts ("linearity" and "leniency" of generated platform levels are her example) across a generative possibility space. To do this, many levels are generated, these properties' values are calculated from each level, and then each level is added to a bucket in order to render a histogram. The final histogram represents the frequency with which each possible balance of leniency/linearity occurs in a generated level. It is far from a complete view of a level generator, but it gives the designer a faster and more objective view of their generator than playing through hundreds of generated levels.

Smith's work is pipeline-agnostic: we do not need to know anything about the model being used to create the level in order to create the expressive range analysis histogram. If we have a parametrically-encoded model, then this histogram can become interactive in a few different ways.

Danesh[105][s41] expands on the "expressive range" framework in order to turn

it into a casual creator where novice users can slide tuning values or ask Danesh to "autotune" those values towards some heuristic. Danesh, unlike general expressive range analysis, *does* require a parametric space (for the sliders), and even needs a more or less continuous one with high locality in order to do hill-climbing (though can do a random-search optimization without that).

Expressive range analysis compresses artifacts which vary among *many* dimensions into a two-dimensional space. *Danesh* lets the user choose which pair of properties to map, selected of all the available properties defined for that generator. Each of these graphs is a 2D *slice* of an invisible N-dimensional cube (where N is the number of properties that can be measured). To display more than two dimensions at a time, dimensionality reduction (such as Principle Components Analysis, PCA) can be used to flatten all of the dimensions into a two dimensional plane, as *Dryad*[s56] does in Figure 11.5.



Figure 11.5: Dryad's multidimensional "landmark" trees.

11.2 Moving through possibility space

We could also imagine dimensionality reduction used to find "metasliders" that represent control over multiple sliders in some complex way. We can approximate this by taking two landmarks in the space and *subtracting* their difference to create

an offset vector (say, a prototypical oak and cactus in Dryad). This vector (oak - cactus) can be algebraically combined with other points in the space:

```
palm + oak - cactus
```

```
palm - (oak - cactus)
```

This should thicken the trunk and remove leaves, or move in the opposite direction. This algebraic combination of landmarks could itself be modeled as a slider. And though *often* blending sliders go from 0 to one, algebraically, there is no reason it could not go negative or greater than one.

Indeed, for each element from Kevin Lynch's framework (nodes, paths, edges, districts, landmarks), we can imagine ways to reinterpret them as vector space manipulations.[6] We can derive landmarks from user interest (*Dryad*) and use them as points to generate Voronoi regions, or use clustering to define regions (*Petalz*) and then select central landmarks at their centers (prototype theory). Paths can be created as commonly-traveled highways between popular landmarks, or as winding scenic paths designed to tour the less-accessed parts possibility space. Edges and nodes fall out naturally from these previously designed components: edges are borders between districts, or where districts hit the edge of possibility space, and nodes are places where multiple paths intersect.

Once we have these structural annotations of the possibility space, we can use them to create different forms of interaction:

- parameter sliders
- sliders blending between two landmarks

[6]It is *possible* to do this for a non-parametric space, but much, much harder.

- moving a point along fixed paths through the space (a tour of known good landmarks)
- moving a point along dynamic paths (from Perlin noise of other infinite function)
- district-relative movement (away from or towards boundaries, i.e., "give me something that is not easily cataloged")

These techniques have not yet been attempted in a casual creator (or any interactive generator) that I have come across, but it has been famously successfully used in Word2Vec[194] and less famously with Pronunciation2Vec[299]. These techniques require a space that has good locality but not too much predictability, but if a parametric space fits those requirements, it has the potential for some very novel ways to explore the space.

11.2.1 Animation

While I have discussed these traversals as something that the user is controlling, they can also be controlled autonomously or indirectly. Giving the user control lets the user find interesting artifacts, but if a space is *continuous*, then the movement through the space creates a smooth animation, which it turns out, is an interesting artifact in itself. These "morphing" animations resonate so strongly with users that just giving the users oblique ways to control the animation can be a casual creator experience[7]. I have experimented with controlling movement through the space with interactive and non-interactive inputs: time (animating along a random path through the space over time), Leap Motion hand-tracking, face tracking, and music (linking the sliders with the music's current amplitude at

[7] *Funky Ikebana* has been shown as an ambient interactive artwork at parties.

various frequencies). None of these are conducive to finding particular artifacts, but they are entertaining, induce flow, and in the case of the interactive ones like Leap Motion, let the users express themselves creatively by "puppeting" the alien motion through the space.

11.2.2 Optimization

How does all of this affect the grokloop? Aside from the Lynch annotations (landmarks, etc) how do the users understand their movement through the space? Fortunately, one need not understand the space as a whole 20 dimensional cube, but can get a sense of the space *around* the current location. Local navigation requires a possibility space with good **locality**. Since the *Funky Ikebana* generator primarily uses continuous functions as inputs to the L-system, its mapping from input vector to output flowers is continuous and also happens to be fairly local (flowers that are blue are perceived as being close to flowers that are purple-blue, for example).

The basic interaction of *Funky Ikebana* is an *iterative local* navigation. Seven to ten random starting vectors are selected, and the flowers are generated and displayed on screen. The user picks their favorite from the displayed flowers. Several new points near that flower's input vector are sampled, and used to draw new flowers on the screen. The user can repeat this as many times as they desire.

At each step, the user is evaluating a number of vector positions that are a short "jump" away from the previous flower. As the possibility space has more than twenty dimensions, this does not allow the user to select from all possible directions. Yet they can still find a "better" flower than the one they have currently. Each step they take is climbing up to a higher-rated flower, an algorithmic approach known as "hill-climbing"[340].

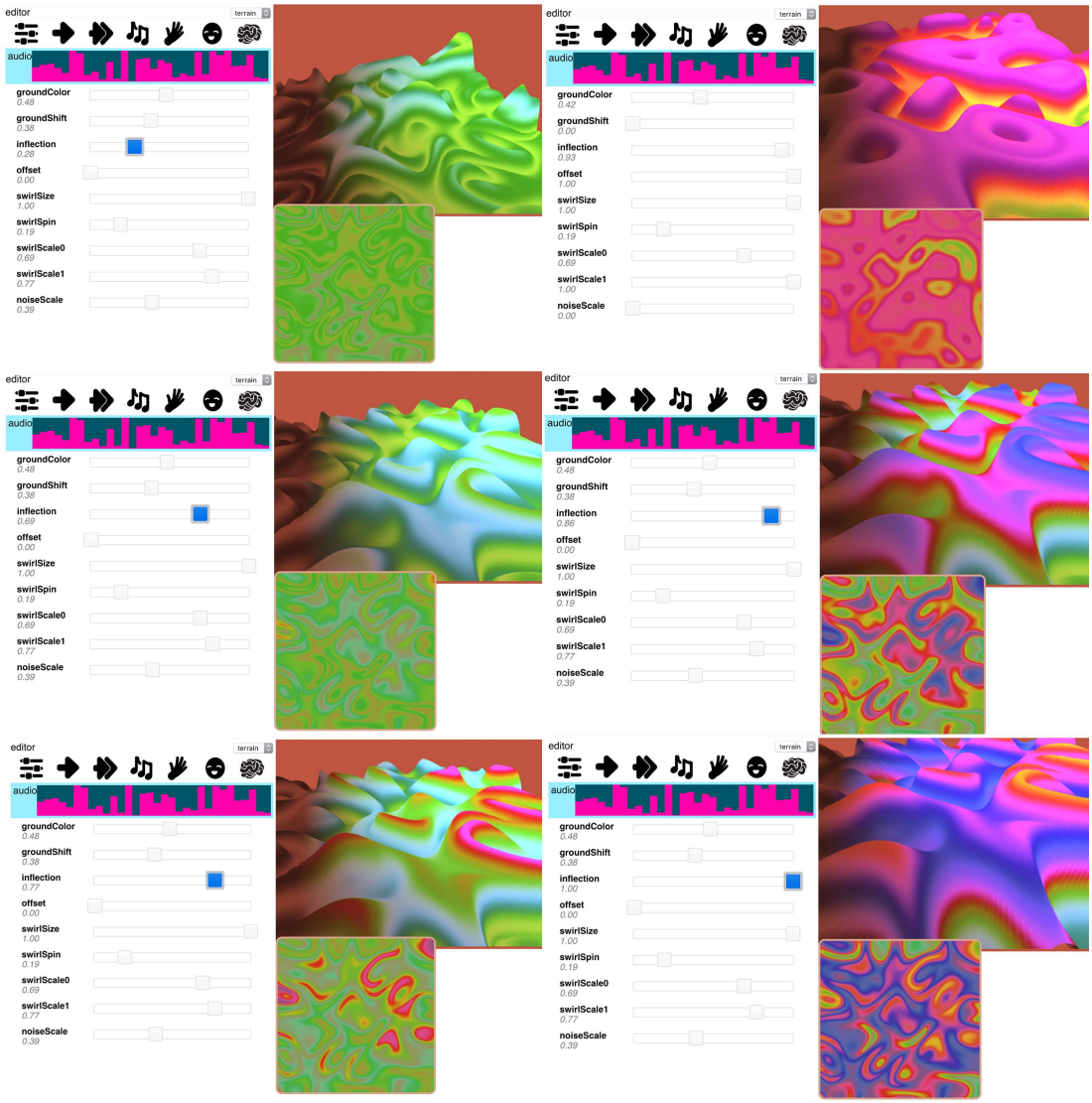


Figure 11.6: *Slidertoy*[s211] is a platform I built as a way for users to explore parametric spaces with a variety of interactive techniques. They can slide sliders to find a particular point/artifact in the space. They can set it to perform an animated randomwalk through the space, which morphs through the continuous spaces of artifacts. Or they can hook the inputs of the parametric system up to several different input pipelines that create arrays of numbers, like music frequency or the Leap Motion.(`http://www.galaxykate.com/apps/aof/`, `https://github.com/galaxykate/slidertoy`)).

Interestingly, unlike many automated hill-climbing algorithms, this does not rely on a fixed heuristic for "quality" in the flower, as many users seem to seek

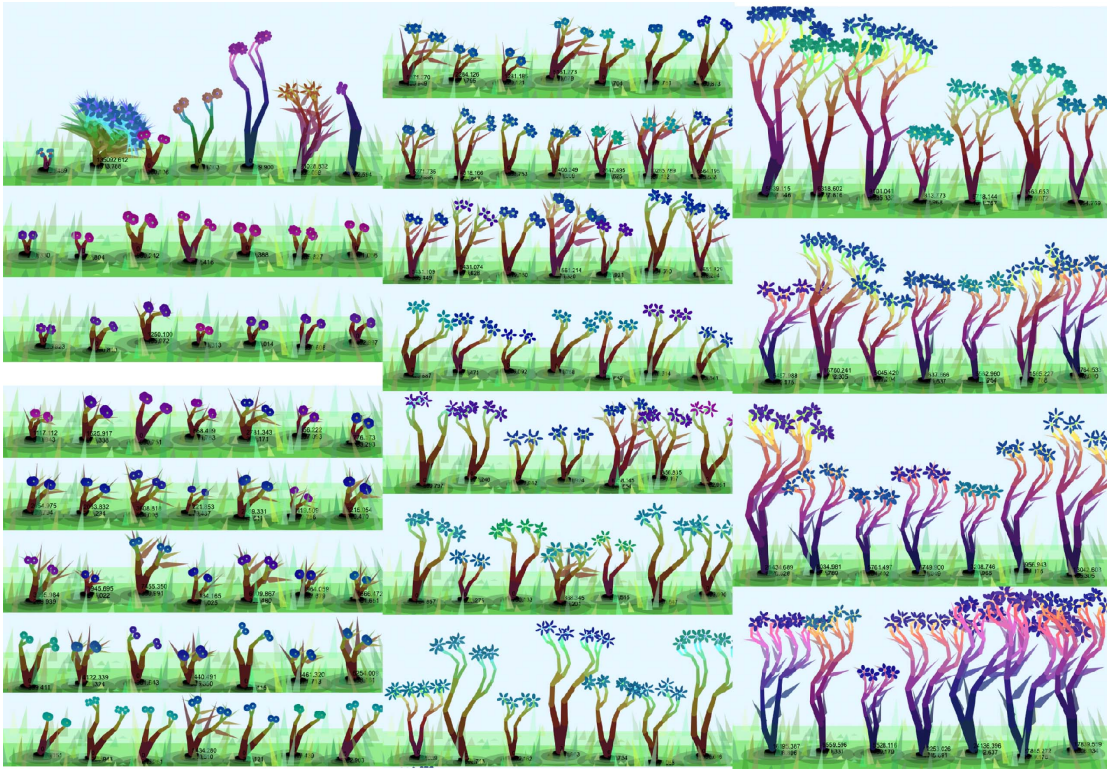


Figure 11.7: Gameplay in *Funky Ikebana*. Seven flowers are animating attractively (offsetting the L-system angle by a sine-wave of the time). The user clicks at the base of their favorite, which creates seven "offspring" with slightly-varied "DNA". They can do this very quickly, because the process of visual evaluation, clicking and refreshing is nearly instantaneous. This makes for a fast grokloop and an effortless journey through a wide range of floral possibility space.

one kind or direction of flower (optimizing to flowers which are "large" or "blue", for example), then grow tired of it, and begin optimizing for novelty instead. But even though each user sees only a small part of the possibility space, they get the experience of directing their own path through it, according to their own heuristics.

This form of local navigation may be performed by a human player, or an automated process, or some combination of the two. A common form of hill-climbing that makes use of the transformation between the possibility space of inputs and the possibility space of outputs is an Evolutionary Algorithm[17].

An evolutionary algorithms has three kinds of data and three algorithmic or generative pipelines. The data structures are as follows:

- a "DNA" data structure, a **genotype** or model
- an "unpacked" form that the DNA can be expanded into, a **phenotype** (the Evolutionary Algorithm term for the finished artifact)
- an **evaluation** that represents the fitness of the unpacked form

How should these structures be implemented? That depends on how the transformations will work. The particular implementation of each transformation will determine much of the user experience. It is these transformations which create the mapping from the possibility space of the genotype (usually a simple data structure such as an array of numbers, less commonly a tree of choices) to the more emergent possibility space of the phenotype or artifact.

- a transformation to grow or unpack a **genotype** into a **phenotype**
- a transformation to evaluate a **phenotype**, or an ecosystem of them, to create an **evaluation**
- a transformation to take the **genotypes** and their **evaluations** and return a new set of **genotypes** for the next generation

The data structures themselves matter little, except in the way that they enable new transformative methods, which themselves enable new user experiences. What properties do the genotype-phenotype-evaluation mappings have? Are they direct, or more emergent? A *direct* mapping will allow for the hill-climbing to more efficiently find a good solution, but an *emergent* mapping may have more interesting solutions to find.

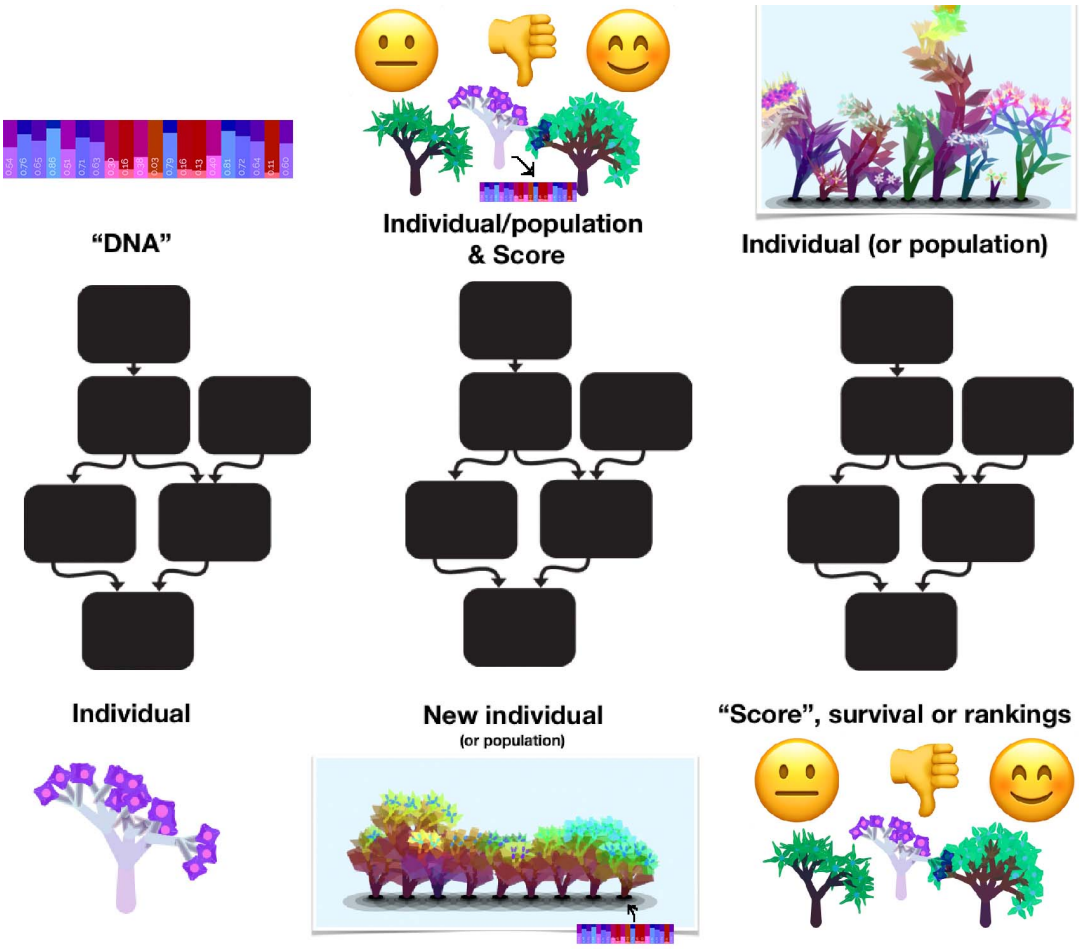


Figure 11.8: Genotypes transform into phenotypes, which transform into evaluations, which pair with the original genotypes to become a new set of genotypes. Each one of these transformations can be a generative pipeline(shown here as a pipeline of *Generominos*).

Each transformation is also an opportunity for a new kind of interaction. Can the user create new genotypes? With what tools? Can they "breed" together favorite artifacts? Can they edit the DNA directly, or must they use indirect methods of control like selection or simulation? And who or what evaluates the finished artifacts? In the next section, I examine a few of the rare Casual Creators that currently use evolutionary algorithms. Each one has a different take on what it means to "control" a hill-climbing algorithm.

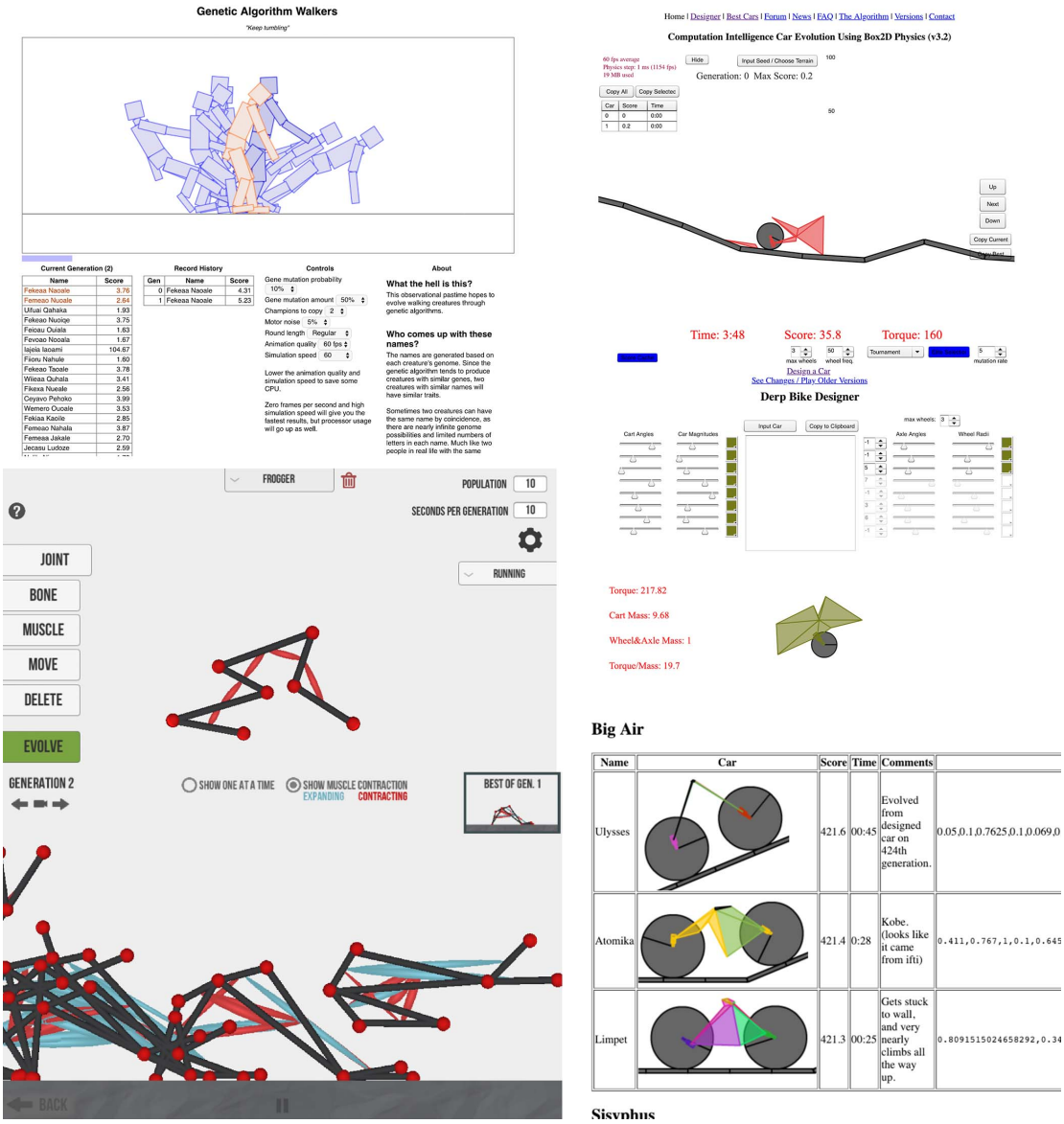
11.3 Hill-climbing as a casual creator

Genetic algorithms are used in industrial design applications[197][207], but are less commonly seen in entertainment or interactive applications. Many uses of evolutionary algorithms in entertainment follow an **evolution spectator** model. The early generations of an evolutionary algorithm are often humorously inept, creating bad art or flailing animations, but eventually evolve into competence or even surprisingly clever solutions. Watching this process is hypnotically entertaining, especially for genres that already celebrate failure, like the evolutionary algorithm that generates walk cycles for the bad-walking game QWOP[313].

BoxCar2D[s21] and *Genetic Algorithm Walkers*[s83] are examples of **evolution spectator** systems. In both, the user can select certain evolutionary parameters (number of "winners" that breed, genetic variance of offspring, but most of the experience is in letting the computer simulate many generations of evolving artifacts. BoxCar2D simulates two-dimensional cars made of simple primitive shapes, and visualizes them rolling over an increasingly difficult terrain until they eventually get stuck. Each time a car fails, a new car is generated using a genotype modified from an earlier "winning" car. *Genetic Algorithm Walkers* names each "walker" (giving a generative name to a set of tuning values for a walk cycle so you can root for a "person"), and simulates the two previous winners (in red) and ten of their children (in blue) attempting to walk forward, until all of the walkers have fallen and a new generation starts with the victors.

Evolution by Keiwan[s111] is more clearly a casual creator. Like NASA's experiments in evolving muscle-movement patterns in tensegrity robots[207], the user first defines a graph of muscles and bones to create a creature. Then dozens of that creature, each with different muscle tunings, are simulated in a task (walking, climbing) to determine the winners for the next generation, and it becomes a

spectator app again. At any time, the user can go back to the editor to restructure the body, and restart the tuning process again.



Name	Car	Score	Time	Comments	
Ulysses		421.6	00:45	Evolved from designed car on 424th generation.	0.05,0.1,0.7625,0.1,0.069,0
Atomika		421.4	0:28	Kobe. (looks like it came from ifti)	0.411,0.767,1,0.1,0.645
Limpet		421.3	00:25	Gets stuck to wall, and very nearly climbs all the way up.	0.8091515024658292,0.34

Figure 11.9: (left) The muscle editor and simulator in *Evolution by Keiwan(, and the simulator in Walkers, which has no editor. (right) The simulator, parametric editor, and high-scoring user-discovered artifacts from BoxCar2D.*

In all of these cases, the evolutionary heuristic is computationally-simulated success. Another common form of evolutionary casual creator is an **curated**

evolution system, where the user (or many users) provide the heuristics for hill-climbing (or evolution).

An old and renowned example of this is *Electric Sheep*[s62] which is advertised as a screensaver. It can be downloaded for free and run on a computer, where it will download "sheep", a phenotype of several parameters which are then expanded into a flowing animated "fractal flame" which displays on the user's screen. Behind the scenes, "everyone who runs the screensaver also participates in its creation"[129], by selecting favorite sheep, but also by spending computer cycles to render the frames of new sheep. Viewers can up vote or downvote sheep, and popular sheep are used to create new sheep which are distributed out to the users for evaluation. This process has been running since 1999, and thus the community has, as a unit, been exploring the parametric possibility space together. Crowd-sourced voting is a very indirect form of control, though, as not only is there not direct control over the hill-climbing, there is no individual control over that indirect control. Recognizing this, users have created parametric editors[s158] to create and edit sheep with more direct control. In these editors, users can generate new sheep, edit them, or even "breed" them together.

11.3.1 Ownership in genetic algorithms

Electric Sheep is a project well supported by the community, which documents and annotates the sheep that pass through the system. For an environment that is mostly designed for the crowd-sourced semi-anonymous sharing of generated images, they have created tools that allow individual users to take credit for which frames they have rendered, which new sheep they made, and the descendants of that sheep.

PicBreeder[355][s169] is a curated evolution system for evolving strange gener-

ated pictures by clicking on one of several descendants. Each image has its own page on the site, which shows the ancestry and descendants of each image, the user who discovered/made[8] it, as well as a way to buy printed merchandise of that image.

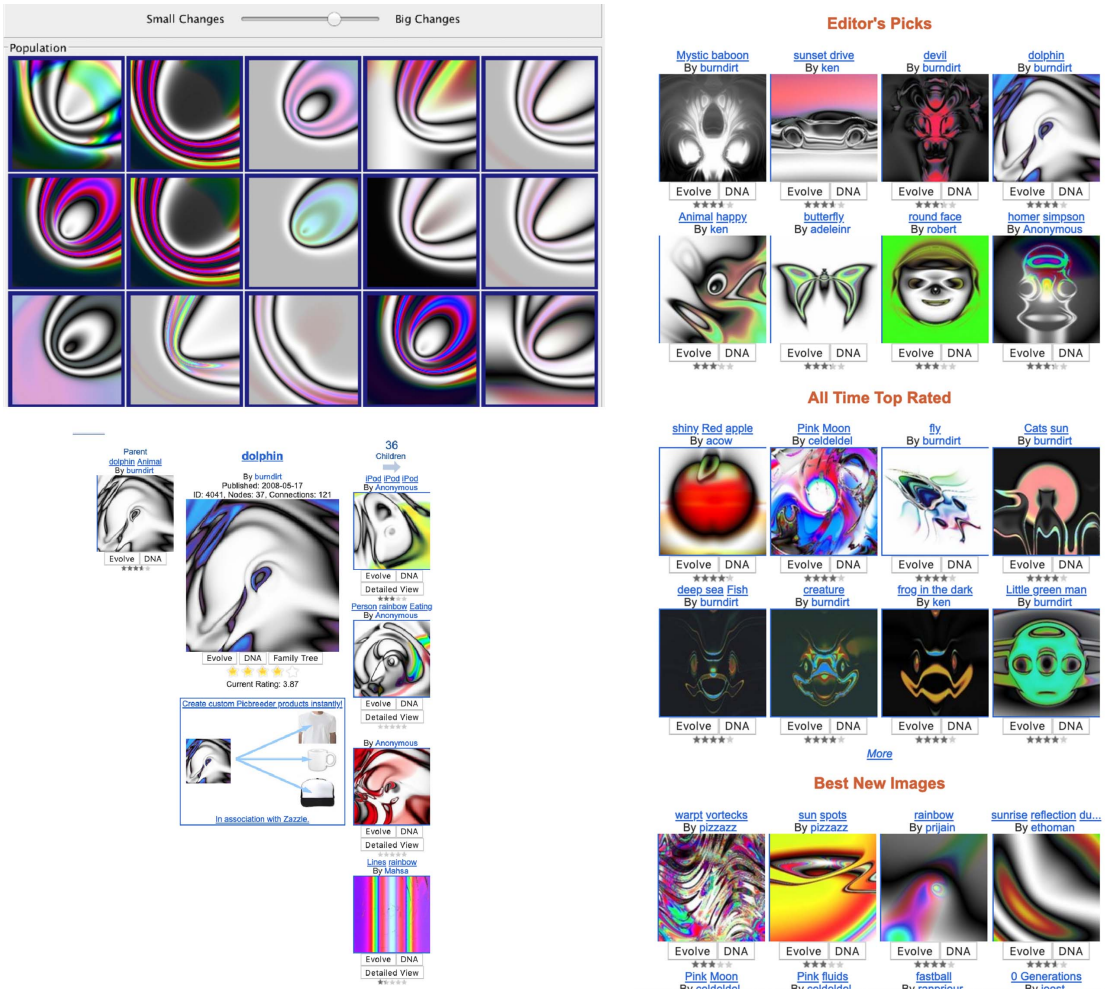


Figure 11.10: One of the images with the most descendants on PicBreeder (`http://picbreeder.org/search/showgenome.php?sid=4041`).

Why is this important? What is behind the apparently common desire to give credit to a single user for discovering part of the possibility space? Like the Lynch

[8]I had a vigorous argument with Ken Stanley, one of the creators of *Picbreeder*, in which he argued that mathematically, all the images already existed and users were discovering them, but I argued that to the users, it felt like creation. We were probably both correct

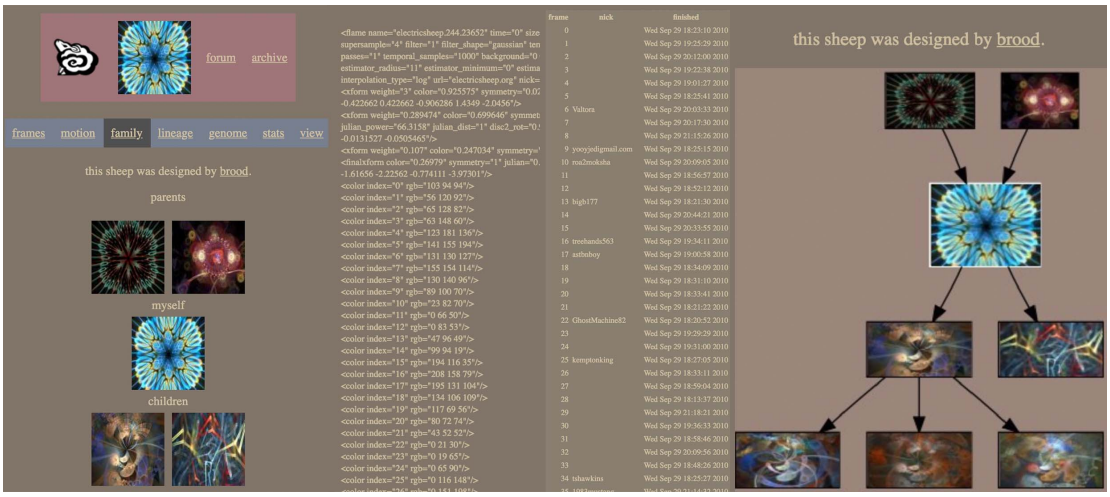


Figure 11.11: Family tree of a sheep, with its creator, its parents, its descendents, and users who rendered one of its frames/ (`http://electricsheep.com/archives/generation-244/dead.cgi?id=23652`).

landmarks, this creates meaningful annotation of a space. Ownership and social status is a very powerful signifier, and creating an overlay of social meaning over a possibility space further amplifies any structures in the system.

Petalz combines this technique with the difficulty of accessing parts of the possibility space. Since moving through the possibility space is a challenge, they can promote users who have, through struggle or accident, gotten to rare parts of the possibility space. This creates social structure between two players (now there are haves and have-nots) and motivates actions between players:

> "Because flowers are genuinely novel artifacts, their seeds have potential economic value that can allow skillful breeders to be rewarded for their efforts. Furthermore, the transaction in which one player buys seeds from another creates a new kind of social interaction that links the players in the transaction."[322]

Mendel[s133] is a single-player game in which the game generates L-system flowers from DNA[9]. These flowers, though, are distributed around an island, and so are more difficult to collect. When the player finds a new one, this discovery is

[9]Sounds familiar!

celebrated by encouraging the player to name the flower, as well as telling them genetically how close it is to their other discoveries. This gameplay lets the player *map* the space of flowers using landmarks and their relation to each other (genetic similarity), as well as declaring ownership (even in single player) over parts of the possibility space that they have explored.



Figure 11.12: Scenes from *Mendel*: naming flowers, collecting them, and seeing them in relation to named specimens.

11.3.2 Parametric patterns in Machine Learning

Machine learning is a popular area of interest, but often these systems are black-boxes with little interactive playfulness. However, it seems that some machine learning developers have either seen casual creators like *PicBreeder*, or have reinvented their patterns from scratch. If an artifact from a machine-learned embedding can be represented as a series of numbers, then it is not surprising that the same techniques can be used for traditionally-programmed generative pipelines, and machine-learned ones.

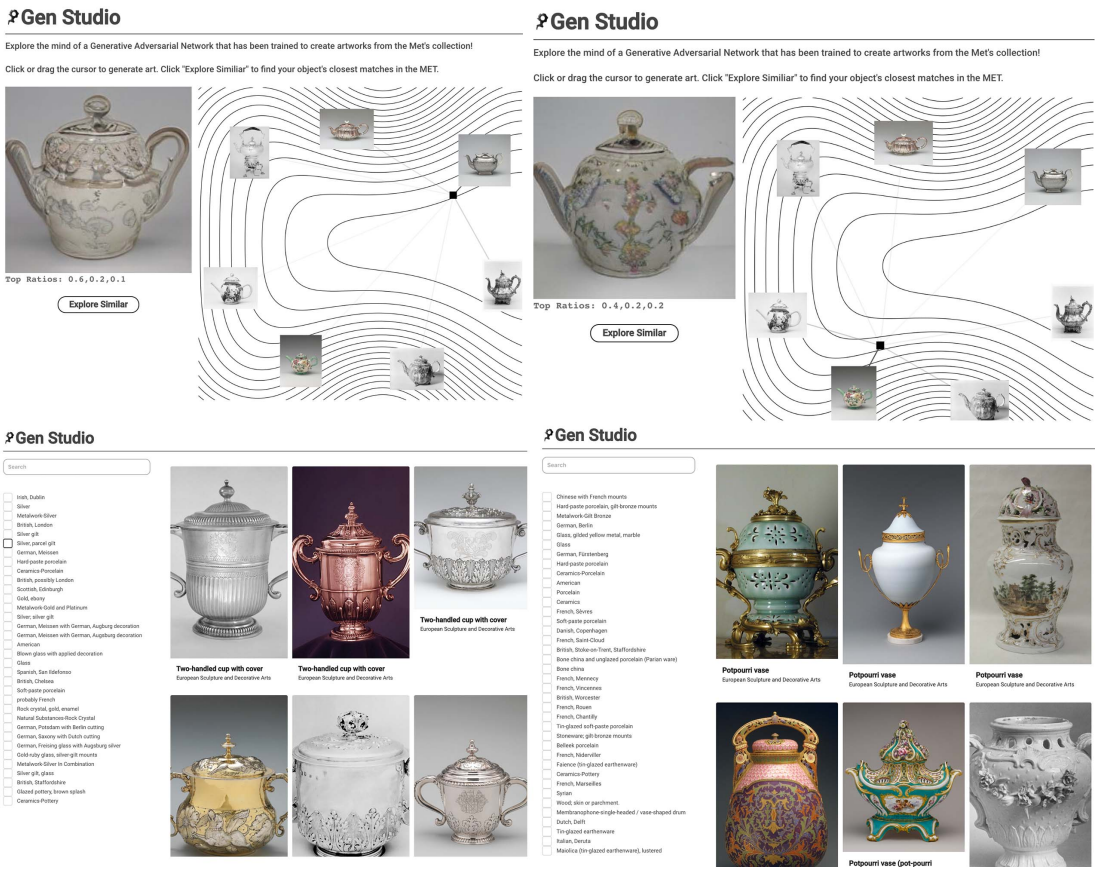


Figure 11.13: A joint project with Microsoft, MIT, and NY's Metropolitan Museum of art, *Gen Studio*[s84][180] maps out artifacts from the Met's collection as a wonderfully Lynchian map of landmarks. The user drags a point spatially, which creates a vector point somewhere between the sample *real* artifacts, which drives a GAN to make a synthetic artifact that reads as that percentage of each artifact.

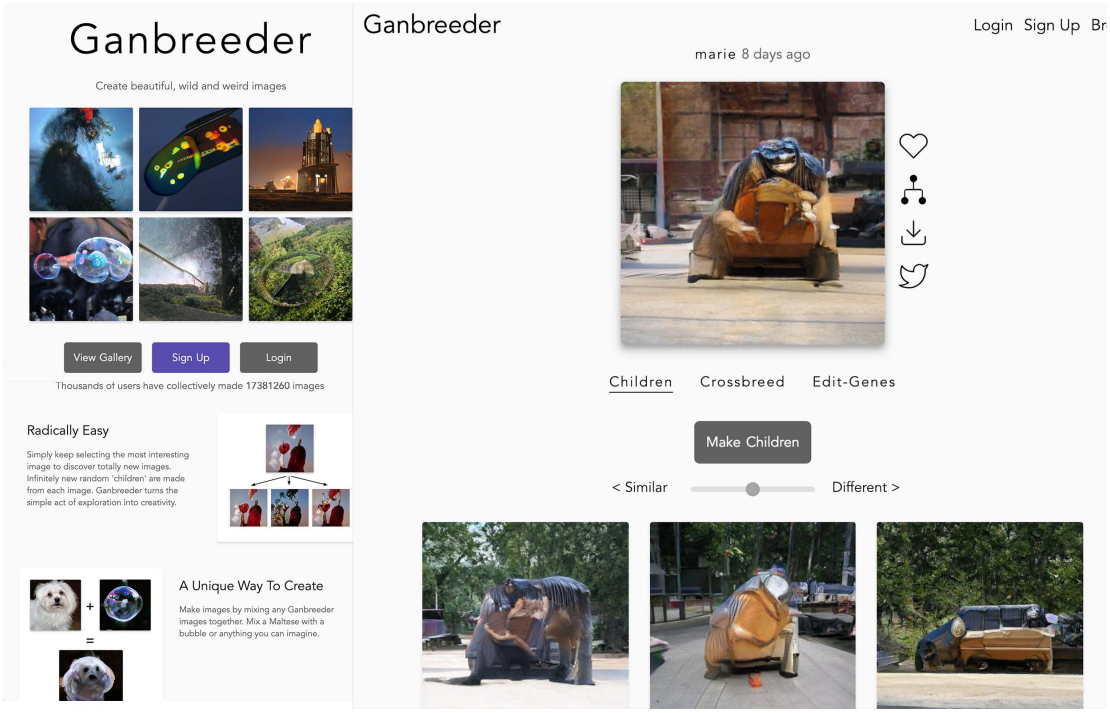


Figure 11.14: Interaction-wise, *GanBreeder*^{s80} is almost an exact clone of *PicBreeder*. Users can breed or mutate GAN-generated images, and crowd-voting surfaces the current most exciting "landmarks" to evolve.[373].

Chapter 12

Drawing Tools

> "Sim City is a paint program for making cities" - Chaim Gingold, "Magic Crayons", GDC 2007

What does it mean to draw? And why are so many casual creators *drawing tools*? In this chapter, I examine casual creators that use *drawing* as their primary form of interaction, and *paths* as their central data model. There are hundreds of casual creators whose possibility space is "a path in space"; how do these creators manage to have such different ways of interaction, feedback, and visualization?

12.1 Drawing on different perspectives

Drawing can be defined as the action of making marks on a surface[118], a practice humans have been engaged in for tens of thousands of years. But something obvious is missing from this definition: *why* do we draw?

Is it to record information? Pick up any "how to draw" manual, and it will have advice on perspective, accurate shading, creating outlines and forms, that is: many ways to accurately render a scene, real or imagined. Implicit when we say "drawing" is the idea that the goal is to create representational images,

images that record or capture something accurately. To be good at drawing, a "draftsman" like Dürer or DaVinci, is to be good at "capturing" a likeness. This is only one reason to draw though: we also make marks because we enjoy the marks themselves, as in the beautiful lines of calligraphy that transcend the words that they record. The dot artworks of Yayoi Kusama record her own intentions, or in the case of her interactive dot-placing works (Figure 12.1, the intentions of the gallery visitors[230]. We also draw for the pleasure of the drawing motion, drawing lines in wet sand with a stick, in snow with a glove, or in the air with a sparkler. The marks of an artist like Jackson Pollock record his gesture and the random physics of scattering paint, as Rosenberg describes in contemporary history *The American Action Painters.*

> "At a certain moment the canvas began to appear to one American painter after another as an arena in which to act—rather than as a space in which to reproduce, re-design, analyze or "express" an object, actual or imagined. What was to go on the canvas was not a picture but an event.[..] The painter no longer approached his easel with an image in his mind; he went up to it with material in his hand to do something to that other piece of material in front of him. **The image would be the result of this encounter.**"[326], (emphasis mine)

Archaeologist and art historian Whitney Davis considered it plausible that the art of mark-making, and thus drawing, arose from a gesture that left a record of itself. If true, the first 'paintings' were, like the Action Painters, representations of a gesture itself rather than a symbol to represent an external signifiers:

> In making a mark for the very first time, one could notice its visual or kinesthetic similarity to the depictive gesture. In the most literal version of this scenario, a depictive gesture accidentally leaves a trace in a permanent mark; here, "freezing" is occurring at the very moment of making both gesture and mark.(Whitney, 1993)

So if our experience of drawing can privilege any intention (recording a scene, making an aesthetic image, or leaving traces of a gesture) so too can our drawing

tools. When we build digital *drawing tools*, the distinction can be forgotten, and many tools are uncritically built to optimize for the control that a good draftsman would need to express their expertise. Precision, control, predictability are critical to digital tools with this goal, as well as in physical drawing tools like pens and pencils. It is easy to *assume* that representation and communication are essential elements of drawing, but they need not be. Reexamining this assumption allows us to find diverse drawing practices (in historical practice as well as digital systems) where this is not the case. From a casual creators perspective, we can consider *all* of the ways to experience drawing, and for each one, imagine a variety of tools that would support those experiences.

Pleasurable or meditative drawing provides a new motivation. *Zentangle*[s247], a proprietary form of meditative drawing practice, is an illuminating example, as it very explicitly advertises itself as a practice, not a way to get good art. The Zentangle site [406] characterizes it as "fun and relaxing" with easy-to-perform actions that lead to unexpected results:

> "The Zentangle Method is an easy-to-learn, relaxing, and fun way to create beautiful images by drawing structured patterns. We call these patterns, tangles. You create tangles with combinations of dots, lines, simple curves, S-curves and orbs. These simple shapes are the "Elemental Strokes" in all Zentangle art. These patterns are drawn on small pieces of paper called "tiles." We call them tiles because you can assemble them into mosaics. Zentangle art is non-representational and unplanned so you can focus on each stroke and not worry about the result. There is no up or down to Zentangle art. If fact, you can most easily create Zentangle art by rotating your tile as you tangle – always keeping your hand in a relaxed position. You don't need to know what a tangle is going to look like to draw it. You just need to know the steps. The result is a delightful surprise. As you use the Zentangle Method to create beautiful images, you likely will enjoy increased focus, creativity, self-confidence and an increased sense well-being."[407]

It also describes zentangling as ceremonial and timeless, citing the history and

meditative quality of repetitive mark-making, and mentions many psychological and personal benefits to the practice. Very little mention is made of the end products. The goal is not to produce better figurative or communicative images, but to make a better (or happier or calmer) person.



Figure 12.1: Interactive mark-making installations are popular in museums. These provide a wonderful space of casual creator examples, as they have to balance *some* creative expression on the part of the participant with automation or structure that creates a cohesive whole for the museum space. Yayoi Kusama's "Obliteration Room"[s151][230], Joseph L Griffith's "Drawing Machine 1"[s53][170], Karina Smigla-Bobinski's "ADA"[s3][171], Ryan Feddersen's "Coyote Now"[s37] coloring book mural at the Santa Cruz MAH[134].

12.1.1 Mark-making

The human practice of mark-making goes as far back in recorded history as recorded history does, for when we record history, we do so by making marks, and by making them permanent. Indentations on a clay tablet, were baked into permanence in ancient Assyria;the knotwork record-keeping of the Incan empire took momentary intentions and set them into permanence. Oak galls, lumps of misshapen wood formed when a wasp buries its eggs in a tree limb, are boiled, guided with capillary action down the tube of a split feather, and indelibly stain parchment and paper with the marks of the holy books and legal documents of Medieval and Enlightenment Europe. There are less-permanent marks, too: the sand mandalas of Buddhism, brushed away after creation, or the shape of footprints on a beach. We make marks with intentionally and without intention, permanently or ephemerally, for purposes serious and frivolous, in old clay and in digital storage. And of course, we make marks for art.

Why examine the "mark" as an individual entity, as this chapter does? For several reasons. We can analyze the mark by the properties that define it as a mark: its temporal permanence, its limited spatial existence, the necessarily linear nature of the gesture that created it. These properties will form an ontology of "mark-ness", and we can use these to understand the different ways that marks can be represented digitally, and knowledge of the diversity of digital representations will illuminate their diverse affordances. *How we represent a mark in code defines how we can interact with it.*

Putting attention onto "mark-making" also allows us to shift the focus from the mark, to the act of making it. What tool makes a mark? How is a tool held? What does it mean to "hold" a tool, when we are speaking digitally? What role do hands play in mark-making, and what are their affordances? What is a "gesture"

and how does it relate to drawing? If you can make a gesture without hands, can you also draw without hands? And this is the final purpose of this chapter on mark-making: by documenting what we know to be true of mark-making, we can discover the boundaries of our current assumptions. In this chapter you will see systems that make marks without gesture, without hands, without humans, and without a surface to make a mark on.

Anatomy of a mark

How can we represent a mark digitally? In the physical world, a mark may be made from presence (ink saturating paper), absence (the wax preventing ink from saturating paper, e.g. in batik), or difference (the height-change of carved lines in clay or sand). For now, let us consider the canonical stroke to be the mark of a ballpoint pen. This mark has several interesting properties:

- *It has a beginning.* There is a moment at which the pen nib hit the paper and *began* the mark.
- *It is continuous.* There is some span of time or distance in which the pen was not removed from the paper.
- *It is linear.* Time and space are linked, and over the course of some time, the pen has moved a distance, possibly changing direction, pressure, or width, but not jumping across or through the paper, or into the next room. Each point on the path it leaves is linked with a time, a single point, and a set of properties at that time and place.
- *It has an end.* At some point the stroke ended and the pen lifted from the paper. Our stroke exists for an interval of time, and according to the previous bullet point, also for an interval of spatial positions and fixed distance.

We can represent the position as a function of time or of distance (if the speed changed). We can also represent each of the changing attributes of the stroke (pressure, opacity, width, etc) as a function of the time (or distance).

This is our basic anatomy of a mark. There are several ways of representing marks like this digitally, with different affordances, as I found cataloged by early graphics pioneer (and Pixar co-founder) Alvy Ray Smith in his 1995 memo "Varieties of Digital Painting". [381]. Smith's memos on computer graphics were among very few ontological analyses of digital mark-making that I've found, and I was captivated by the idea that we could catalog all the ways that computers record and use strokes (and discover new ones).

His ontology includes two painting types:

- **discrete** painting, where the system records an approximation of marks by writing pixels in a pixel buffer. This system often uses a **read-modify-write** loop when adding marks-on-top-of-marks
- **continuous** painting, what is often called "vector graphics". A mark is a "continuous stroke defined by the spline", which must be rendered to be used as a painting operation, but can be manipulated independently (scaled, e.g.)

It also includes two accretion methods: **non-maxing** (writing directly to the buffer) and **maxing** (writing to an intermediate buffer for certain threshold-checking calculations, before then writing to the buffer). He also reminds that "Any imaging operation can be performed under the brush here, not just compositing". In current drawing tools, *Photoshop*[s168] has brush strokes that can blur underlying pixels, smudge them along the stroke, move backwards in the history of the buffer, or even fill them with machine-learned pattern matching from surrounding pixels[46].

The CS6 version of Illustrator (a vector-based illustration tool) has a robust brush-creation tool, and demonstrates a variety of brush types in its creation options:

- calligraphic: the angle, roundness and size along this continuous stroke can each be determined by different inputs (random, pressure, stylus wheel, tilt, bearing, and rotation)
- scatter: stamps some selected source object along the stroke, with size, spacing, scatter and rotation determined by inputs (similar to the calligraphic brush)
- art: a selected object is stretched continuously along the stroke
- pattern: like the art brush, the source object is deformed along the stroke, but the stroke is chopped into short sections so the pattern repeats. Can also have a custom head, tail, and corner version of the source object.
- bristle: creates multiple offset profiles of the original stroke, and creates continuous semi-transparent sub-strokes from them.

Based on the many brush examples I've seen[1], I propose adding to Alvy's ontology four ways of *shaping* strokes:

- **stamps**: "stamps" a source object along the stroke, without deforming it.
- **ribbons**: creates a geometric shape from two splines offset from the original stroke, and two splines for the two end caps. Some object (or sequence of objects) is distorted to match that object

[1]Based on brushes in *Illustrator*, *Tiltbrush*, *Oculus Medium*, and all the other art tools in this chapter.

- **extrusions**: a profile shape is extruded along the curve, creating a 3D tube.

- **diffusions**: the original stroke is multiplied and offset, or branches outwards, or breaks into simulated particles. Each substroke can rendered be as stamps, ribbons, or extrusions. Examples: the tree, drip, and loopy brushes in *Kid Pix*, the *Illustrator* bristle brush category. Can become **confusion** brushes when the generativity overwhelms the ability to control it.

Diffusion brushes are an interesting category, as they inherently involve a lack of control, and often use some form of generativity in the diffusion (like the Kid Pix Lindenmayer-system trees). You may remember that the casual creator tradeoff is "lose control to gain power", so these brushes are fruitful tools for casual creator experiences.

It is also important to consider how strokes are stored and whether or not they are *accessible* after having been recorded. For a traditional digital paint program like MS Paint, the canvas is a simple pixel buffer, and most marks just write a color into an area of the buffer, or start a function that recursively writes color into expanding areas until a stop condition is reached. The buffer is never erased, only added to. An Illustrator graphic has no persistent buffer as its canvas, but instead has a scene graph. If the scene graph is changed (a mark is added, a color is changed) the scene graph renders itself onto a temporary buffer and onto the screen.

These two approaches are **additive** and **functional rendering**. In additive rendering, there's a buffer that gets added to and evolved each time, based on the previous state of the buffer. Nothing is ever discarded, it's just built over.

Functional rendering is rendering without a buffer memory. Each time it renders the whole finished product from scratch, doing whatever subrendering or compositing it needs to do along the way.

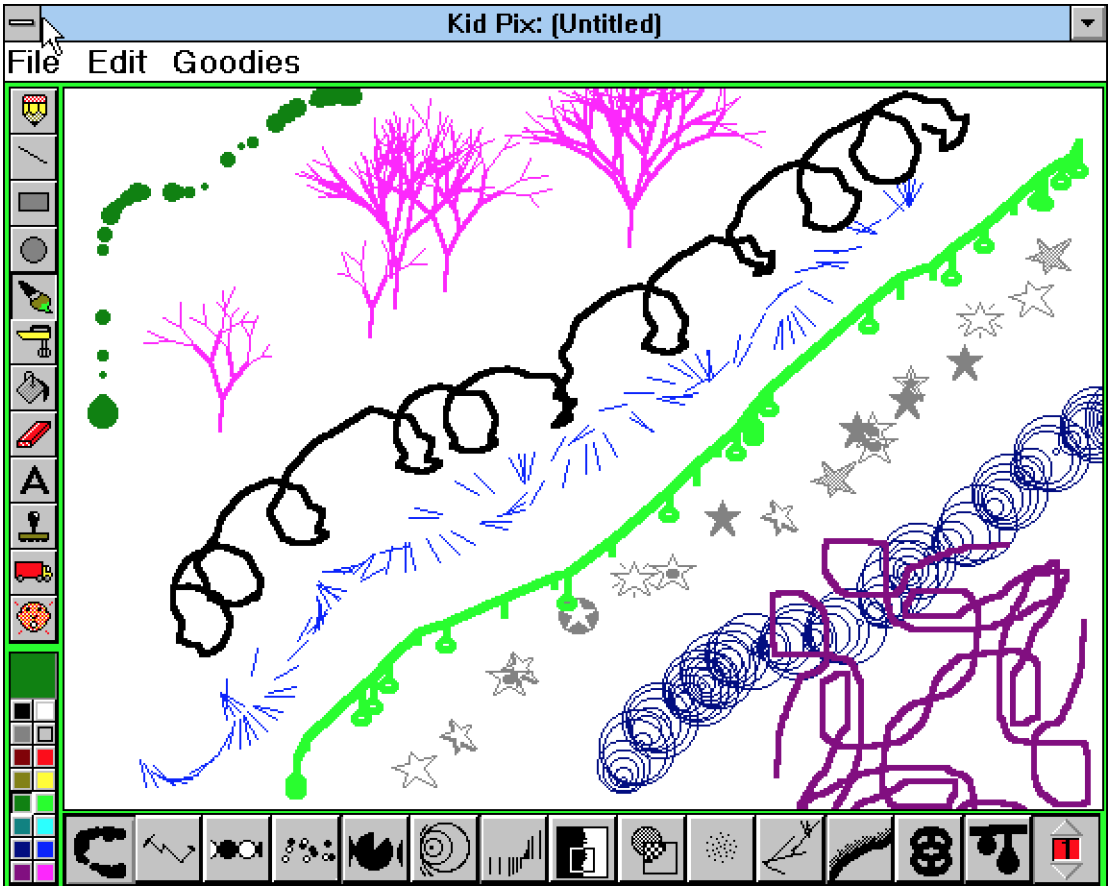


Figure 12.2: Diffusion brushes in *Kid Pix*.

The boundary between these two is surprisingly flexible. In a buffer-based drawing engine like Processing or HTML canvas, I may have a scene graph in memory that I want to redraw each frame. I can "erase" the buffer before drawing each time (setting each buffer pixel to a background color), and thus achieve functional rendering. OR I can choose not to clear the buffer, and each functional rendering will be made on the top of the previous rendering, additively. This technique leaves the past trails of a moving object visible, which can be good for inexpensive trail-rendering, notably used in the Windows Solitaire win animation and the Microsoft 3D Pipes screensaver. There's also an interesting middle path, where the previous rendering is partially obscured. In my work, I often draw a

partially transparent rectangle on top of the previous rendering, leaving the trails visible, but fading out over time in an attractively-veiled way.

Two competing VR drawing apps, *Oculus Medium*[s153] and *Google Tiltbrush*[s235] seem similar at first, with identical inputs, and both using a gestural curve model to record their marks. But under the surface, one is an additive and the other is a functional rendering, and this changes their interaction. *Oculus Medium* uses voxels, a 3D version of the pixel buffer. The user can "mark" more voxels into space, or write over previous voxels (or write over them with blank space to erase). But one can't go back and edit an earlier mark, other than by undoing to the point it was made. TiltBrush is fully functional, where each stroke is stored as a vector, then "rendered" into the scene graph (which is later rendered onto the screen in 3D). This allows modification of previous strokes, and more interesting ways of rendering strokes as animated curves or particle effects or patterned ribbons.

Tiltbrush and *Oculus Medium* are both intended (if unintentionally) as tools for good artists (note how their promotional materials show famous Disney animators and illustrators using them[258]). They want to be flexible enough for anyone to build anything, so they kept their rendering very directly related to the input.

Non-stroke mark-making

Strokes are not the only way that a mark can be made. Consider "regions" of a surface. These may be defined by a stroke, as Illustrator vector objects are, or they may encompass the entire surface. If the surface is a bitmap, a grid of pixel data, then some function can be run on each pixel in the region. The flood-fill operation is found in *Microsoft Paint*[s141] and *KidPix*[s113], but also in many other drawing tools. This operation take a single gesture, a click inside a region, and recursively applies a color to the pixel then to its uncolored neighboring pixels,

until all reachable pixels in the region are colored. This action is a template for many other actions, such as Photoshop's Lasso selection tool, which uses the flood-fill algorithm to recursively add similar pixels to a selection group as long as they are a similar color to their neighbor. The resulting "mark" is a selected region, to which other operations can be applied.

Regions

One example of a non-stroke "mark-making" interaction is the act of designating a flood-fill for a region. Several forms of casual creator are based around the selection (or application) of region-based flood fills. Embroidery patterns, cross-stitch patterns, and paint-by-numbers all consist of a preexisting set of regions, with proscribed fill colors. The user's activity is applying "strokes" to fill each region using paint or thread. The user makes no creative choices in the activity, but I would still considered it a creative activity because the user experiences the pleasures of seeing the picture develop from their own labor.

Coloring books add a dimension of choice: the regions are still fixed, but the user selects their own colors for each fill, then makes the pencil marks to fill those colors in. The user finds enjoyment in seeing the emergent results of the color applications, as the colors interact with each other visually, sometimes in unexpected ways. Digital coloring books remove much of the labor pleasure, as tapping an iPad screen, or worse, clicking a mouse, is a less physically-pleasurable feeling than the soothing frisson of dragging a soft pencil across paper, but the user still feels the pleasures of emergence and fulfillment.

Some activities do not start the user with an existing set of regions. Quilters, for example, may choose to recreate an existing pattern, to design their own pattern from existing components, or to create something with their own choice of

shapes. In the *Zentangle* activity[s247], the user is encouraged to freely draw looping lines across each other. These self-intersecting strokes create a pattern of regions, like a coloring book that emerges from the semi-directed actions of the user. The user then outlines each region and fills it with a different repeating fill pattern, of their own design, or of those suggested by other users. The finished patches of contrasting colors, lines, patterns, and dots create a shimmering patchwork effect that can be attractive without any particular skill on the user's part. *Zentangle* is an example of a casual creator *not* in a digital space: the users give up the fussiness and control of hand-drawing a particular image, instead following a set of abstract procedures in order to get a surprising and emergent design. Although the user could at any point make the choice to start drawing as they please, they constrain their choices to very few at any moment: where to loop the lines, which pattern and color to pick, which piece to fill in next, and how to continue or vary the pattern they are currently drawing. This limited choice paired with continuous action encourages a flow state, and indeed, this is the goal of the *Zentangle* experience.

There don't yet seem to be many casual creators that make interesting use of floodfill. Click-to-color apps are common, and work much like the color-by-number apps discussed in chapter 10. *Photoshop*[s168] has content-aware fill, which has potential for casual use, but is technologically advanced.[46] A few systems also use machine-learning to automatically "color" in images, like *Edges2Cats*[s61] and *Neural Doodles*[s148](which will be discussed further in chapter 22) which take user-provided line drawings and colored regions and fill them with machine-learned cats and impressionist paintings, respectively. But this process is not interactive, except for the creation of the line drawing itself.

Rendering

It is possible to consider 3D rendering to be a form of mark-making. When a 3D scene is ray-traced, are "marks" being made? An image is certainly appearing. It *is* useful to consider it as a form of mark-making, because when it is juxtaposed with interactivity, there are interesting designs that emerge. Though I have never seen this done, consider a ray-tracing app that allows the user to draw into a pixel buffer, but for each pixel they fill, the ray-tracing algorithm traces that pixel's light path, and colors the stroke with the 3D rendering of what is behind it, making a digital coloring book where you could scribble in the wireframe outlines with automatically shaded colors.

There are also other per-pixel algorithms that are worth considering as mark-making, such as Google's DeepDream. In DeepDream[278], for each patch of pixels in an image, a pre-trained neural network is used to optimized that patch towards being recognized as a dog or a tree, or other label. Like the raytracing example, we could imagine a drawing tool that allows the user to "draw" into the image, but each stroke written into the image would be an iterative optimization of the path's pixels towards some heuristic. The user would be painting with a "doggification" brush, with each successive stroke applying yet another optimization pass on the underlying pixels.

12.2 Modes of drawing in casual creators

Now that we have a robust ontology of all the things that can compose a possibility space of marks (thus creating a drawing tool), we can examine the ways in which we interact with drawing tools. I propose four new **modes of interactive drawing** in an autotelic experience, to provide alternatives to the

traditional one of representational sketching.

12.2.1 Conformance

The first mode is **conformance**. In this mode, the user is making marks, but they are making marks according to a pre-established goal. Success at this activity is measured not by creativity, but by conformance to the guidelines. Paint-by-number apps[s159], described in chapter 10 are one example of conformance "drawing", but there are also many learn-to-draw apps in the app store, which have a similar conformance play style, where the user traces over lines in a drawing. Handwriting and calligraphy apps such as *Calligraphy Handbook*[s28], *LazyDog calligraphy and cursive writing practice*[s120] and *HelloChinese's*[s92] character-drawing mode fall into this category as well. Though it is rare in games, in the action-adventure game *Okami*[s154], the user collects "celestial brush techniques"[93], patterns which can be traced with the Wiimote to cast spells.

This mode is *not* inherently conducive to casual creativity, though it is often conducive to flow. The addition of small moments of creative control (such as picking brush styles and colors in *Calligraphy Handbook*) can bring these systems closer to a casual creativity experience. I include this category in the chapter because it illustrates how drawing *is not* inherently a creative activity, and that an app that has a "drawing mode" is not instantly a casual creator.

12.2.2 Operational

In **operational** drawing, the user draws *in order to control some action external to the drawing. Okami* actually contains at least one of these in the brush technique "Bloom", which can either activate a tree to bloom by drawing a circle, or to freely draw a path along which trees and flowers bloom, which can be used

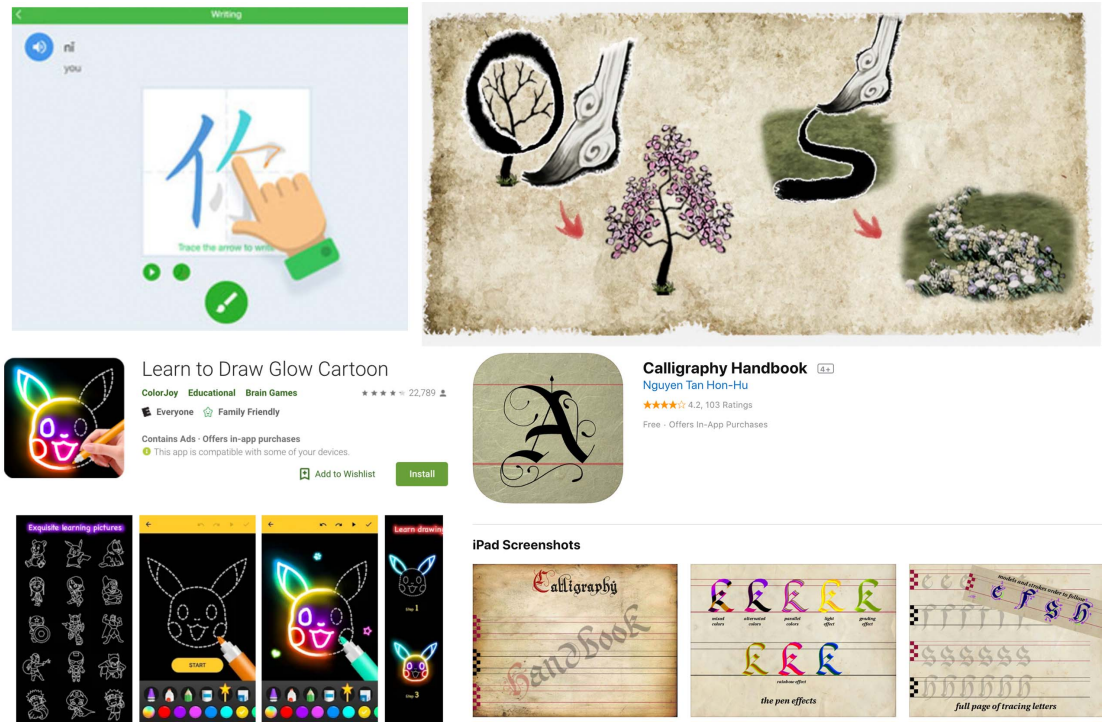


Figure 12.3: Conformance drawing in *HelloChinese*, *Okami*, *Calligraphy Handbook*, and *Learn to Draw Glow Cartoons*.

in combat to slow enemies. The Nintendo DS, with its stylus-controller sparked a boom in using "drawing" to control tradtional game movement. In *Kirby: Canvas Curse*[s114], the player draws rainbow-colored paths that participate in the system's in-game physics, allowing Kirby to slide or roll along them to traverse the level. While "creative" solutions are a part of the game, it never feels like the player is *creating an artifact* as much as they are *controlling a game character*. There are however, clear casual creators that fall into this category.

Line Rider[s124], a 2006 online Flash game by then-student Boštjan Čadež,[451] is a casual creator where the user draws lines, which are then used as a sledding hill by a tiny physics simulated person on a sled[2]. Users began creating intricate and wildly creative levels where the sledder's fall would synchronize with

[2]With a physics simulated scarf.

music[418], staging user-led contests[100] and collecting and archiving levels[183]. *Draw Your Game*[s54] is a creator much like *Line Rider*, except that users draw levels on paper (with black-for-platforms, red-for-danger, and blue-for-movable-objects pens) which can then be photographed to load into game levels, and then traversed interactively by a virtual character.

Operational drawing tools may or may not be casual creators. If the goal is to operationalize drawing in order to get past something, then they are not. If, on the other hand, the drawings are used to *create* something of some permanence that the user would look back at (or save, or share), then they are casual creators.

12.2.3 Accumulative

Recall above where art historian Rosenberg says that Pollack's action-painting is "not a picture but an event", a "result of [an] encounter". A drawing system can be an **accumulative** record of actions or forces.[3] In this mode, there is an interesting set of forces operating on the world, and the movement caused by those forces leaves a visible record of markings behind.

All four of the interactive installations in the image at the beginning of this chapter are accumulative works. "Obliteration Room" shows the accumulation of human attention to a space (especially low-down, where children can reach). The rings of "Drawing Machine 1" are built up from the radial motion of the armature, but also marked by the imperfection of its intentional shakiness. The "ADA" creates clusters of directional strokes, as the users spin it. "Coyote Now" is layered with the meticulous coloring, childlike scribbles, political slogans and profanity of Santa Cruz, all overlaid on top of each other.

[3]I tried out a number of names for this category, including "trail-blazing", "erosion", and "force-map", but no English word captures "a record of forces over time". Probably there is a Norwegian word for "the directional patterns of footprints in snow that shows you the group dynamics of the people who have passed".



Figure 12.4: Lava flows, erosion, and footprints are a frozen record of forces and dynamic movement. Even if we can't "read" the geology, their shapes implicitly tell us about the directions and collisions that happened here (CC-licensed photos from Pxhere, Gina Dittmer, and Tari Noelani Mattox).

This mode works well with agent and particle based systems. In *Action Painting Pro*[s2], the player controls a character jumping around a traditional platformer level, however, as the player moves, they leave behind a cascade of paint particles, recording their traversal in a vivid smear of colors. Pickups in the level change the painting style and color, so the goal is to reach the exit, while also producing an interesting painting. A reviewer of the game describes their accidental painting, and we can hear the casual creator properties of surprise, pride, and ownership in the way they narrate it:

> "Yeah! I didn't intend to draw a perfect dragon (or sea serpent), it just turned out that way. If you pay close attention you'll notice that the body's two humps are the trail of a platform character jumping to the right. The head was created when I collected a spraycan or something and my green expulsions somehow crafted a perfect visage, complete with an eye that is very much a window to the reptilian (aquatic or otherwise) soul behind. It's beautiful."[377]

StrangeThink's *Joy Exhibition*[s107] is another game where users control streams of particles with a video-game trope. But instead of a platformer, *Joy Exhibition* borrows the interaction style of a first-person shooter, complete with paint guns

that are evocative of alien weapons. Instead of bullets, these guns fire paint sprays that react with the screen in different ways, changing color with distance or angle, smearing, blurring, or speckling the wall with different colors. There are no editable layers or undo button, the player is encouraged to just build up an accumulation of paint until they are satisfied. Accumulation-mode drawing tools usually are *not* implemented as functional rendering (or editing of components), instead they additively write and rewrite a buffer.

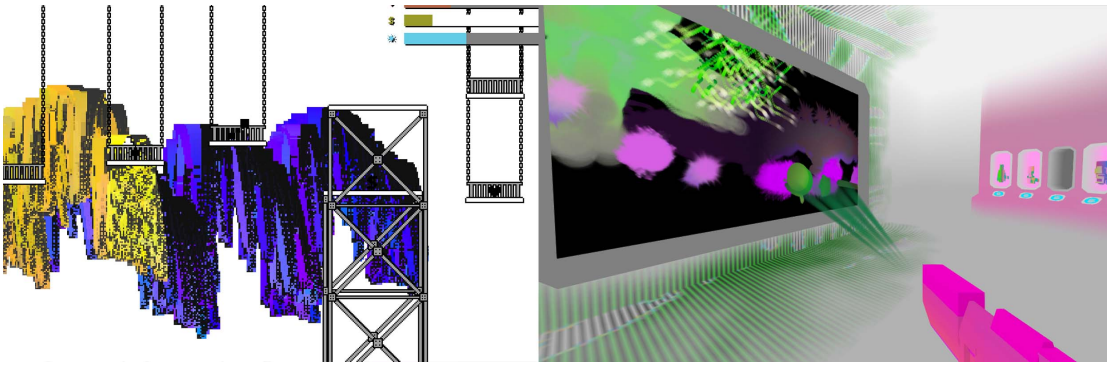


Figure 12.5: Action Painting Pro and Joy Exhibition.

Accumulation-mode drawing can also function synergistically with body-tracking systems, where we want to record force, gesture or body movement. One particularly wonderful and long-lived genre of this is the use of *still images* or moving images as *stamps*. Norman McLaren's 1968 dance film *Pas de Deux* uses multiple exposures to turn ephemeral dance moves into a record of spatial movementFigure 12.6[265]. When the Kinect came out, it was discovered, by artist and technologist Matt Bell, that it could be used to composite multiple 3D "still images" (voxels) into a single frame, which he used to create art videos[28]. DoubleFine's *Happy Action Theater* repackaged many of the experimental art Kinect scene's tools into a commercial Kinect party game, including using the 3D compositing trick to create "Future Booth", a time-travel themed photo-booth in

which users compose their photo not only spatially by temporally.[4]

These tools don't use paths, and so may be a fuzzy example of a "drawing tool". We can see this same technique used *along a drawn path* in *I/O Brush*[s104][338], a 2004 project in which a "paintbrush" with webcam and position tracking can draw a stamped stroke. The stamps are sampled from the world by pointing the paintbrush at them and pressing a button to capture the image (as a still or video clip), like a paintbrush soaking up paint. The drawings, then, become collages of blinking eyes, ladybug patterns sampled from a purse, or animated motions of toys. Like the three accumulative cinematography/Kinect projects, I/O brush records motion, then combines that along a secondary motion of a brush stroke.



Figure 12.6: *Pax de Deux*, Matt Bell's Kinect experiments, and *Happy Action Theater*[178][28][265].

Body-tracking can be used to drive particle systems. In my own work, *Paintthrush*[s160][67], I combine body-tracking, accumulative drawing, and particle systems into a casual creator. *Paintthrush* is a simulated world in which numerous birds are swooping around the space leaving behind brightly patterned semi-

[4]They don't cite Bell or any of the other Kinect artists in their GDC talk, but I have suspicions that they saw them.

persistent trails. The birds are attracted to the user's hands with a springy force, allowing the user to guide nearby birds with their fingers, or close their hands to increase the force and pull the birds into a tight swarm. The birds themselves paint into a 2D buffer behind the 3D scene. The intention behind this project was to make a casual creator where user's could *draw* indirectly with swarms of particle birds (very much a "diffusion" brush from the ontology above). The drawing is poorly-controlled, as the user indirectly directs the birds with their hands. It ended up feeling *too* indirectly controlled to be a satisfying art tool, though the birds were fun to interact with.

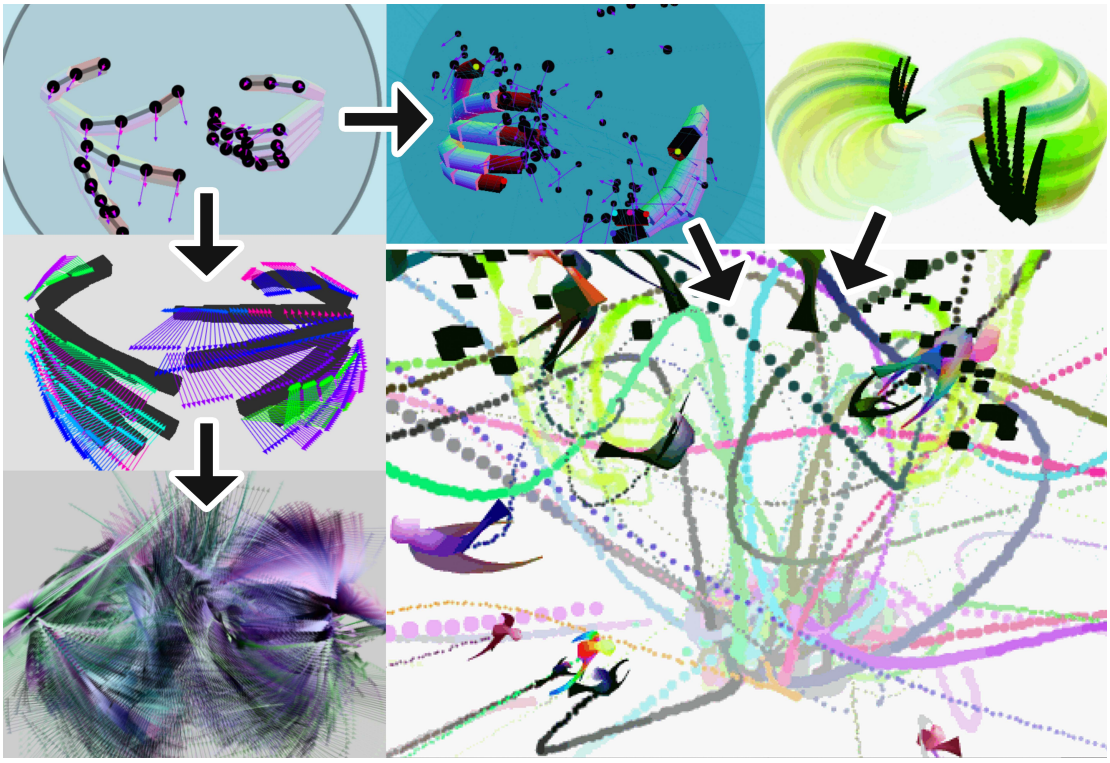


Figure 12.7: *Paintthrush* is a mixed particle-system/accumulative drawing tool. This image shows a number of prototypes that I went through, including some where there were no birds, but the acceleration vectors were drawn *accumulatively* onto the canvas, creating an appealingly shaggy record of hand motion.

12.2.4 Transformational

The final kind of drawing mode is **transformational**, where the strokes created by the user are somehow *transformed* or elaborated on by a generative system. The user is "drawing" in a traditionally-controlled mark-making sense, but the output is surprising, constrained, automated, or generative. Consider the *Spirograph* (a brand-name for a range of mathematical gear tools[243][s214]. I alway enjoy using the *Spirograph* toy as an example, as it is one of the strongest examples of non-digital casual creator; it is also a transformational drawing tool. With the Spirograph, users are in one way performing a very traditional drawing action: selecting a colored pen, moving it to leave a mark, and then picking it back up. But the Spirograph constrains their motion and guides it (by way of interlocking gears) into a more geometric, neat, and surprising path than the user could have made on their own. The user cannot draw the same possibility space of paths that they could freehand, but by trading that control, they get a great amount of augmentation and power, and a reliably good drawing. Their uncertain human gesture is *transformed* into a serenely mathematical one.

Kid Pix[s113], a 1989 (with many updates) computer drawing tool for children, shipped with many different brushes that the user's could draw with. A few were ordinary (discrete, closely-placed stamps to approximate a smooth curve, drawn additively into the buffer). But *Kid Pix* became famous for its many transformational brushes. These created Spirograph patterns, multiplied the stroke symmetrically, scattered multi-colored dots along the path, or came to life (dripping, growing trees, spiraling around the stroke). Craig Hickman, the creator, intentionally made these brushes as a rejection of traditional digital paint tools (and their lack of surprise)

> "In addition to "traditional" computer paint program tools like the brush, pencil, eraser, circle and rectangle, my program would include

> **tools that would be surprising and visually unusual.** Most computer paint programs were designed in a way that made them seem like second class citizens, always attempting to mimic traditional art making tools, but never doing a particularly good job of it. In fact, the computer can make marks in a variety of interesting ways that traditional tools can't. It can also do a good job of creating comic tools. This is where the "wacky brushes" came from."[193], (emphasis his)

Hickman used a variety of generative methods to create surprising brushes (*stamped*, *diffusion*, even some *confusion* brushes) even though they all end up written into a low-resolution black-and-white pixel buffer. Here in 2019, we have many more generative methods (and the computing power to run them), as well as different display options. Drawing tools are experiencing a renaissance in the current virtual reality boom. *Oculus Medium*, *Google Tiltbrush*, and Facebook's *Quill VR*[s183] are all tools for 3D drawing in a virtual environment. Quill and Tiltbrush both use a model of paths through space, which are rendered as textured 3D ribbons or extruded tubes. By repeating many strokes, users can build up planes and surfaces to create 3D shapes, or just make sketches in air. Neither of these tools are pushing themselves as casual creators[5], but *Tiltbrush* has a range of surprising and augmented "brushes" to paint with, including a few brushes of animated blinking scattered stamps, diffusion brushes of multiple intertwining threads, extruded neon tubes, and recently, a set of brushes that animate their settings according to music, so that any drawing with them becomes a music visualizer[281].

Transformational brushes can be used with many different generative pipelines, even very complex ones. One useful property of transformational drawing tools is that, if they update continuously while the user is drawing, the user has a

[5]I attended a marketing seminar for *Quill* in which the announcer derided the possible option to support casual users.

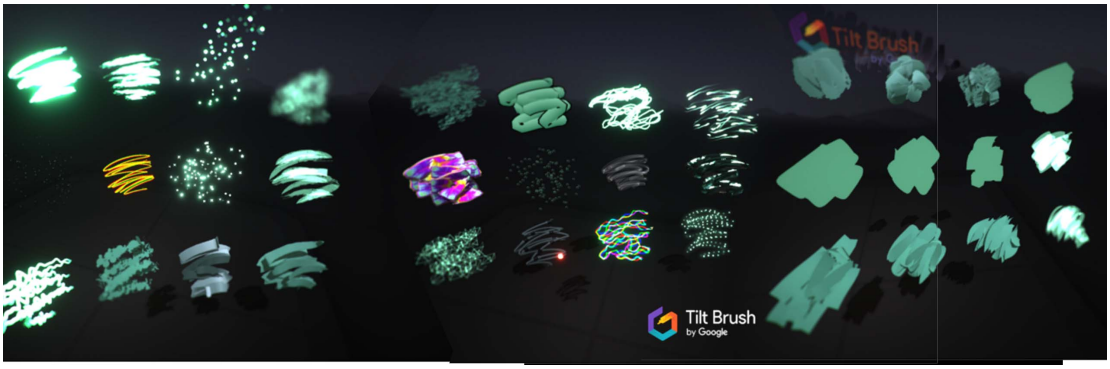


Figure 12.8: Brush types (from [239].

continuous grokloop telling them how their movements are shaping the strange new artifact that is being created. I discovered this property when I was working on a 3D drawing tool, *Pendant Maker*[s164], which had been commissioned by a (now-defunct) 3D printing startup to help casual users make 3D printable object without getting the art-paralysis that occurs when you ask people to "draw something". In *Pendant Maker*, the user draws curves into a small buffer, curves which are truncated to the last few dozen segments, are continually extruded into a 3D tube, are flanged and flared according to some sliders, and then cloned around a central axis to create a mandala-like shape. As a casual creator, it constrained the space to only 3D printable forms (watertight and limited overhangs) that could be quickly printed in a store. I *hadn't* anticipated that drawing with a continuous feedback loop would be so powerful. Users would doodle aimlessly, change direction, and *just keep drawing*. I had added an *undo* slider, but it was almost never used. People just went with the flow, feeling in control, but also empowered.

Like any casual creator, transformational drawing tools can play along the whole interaction-directness spectrum, from control to lively chaos. While *Pendant Maker* calls back to the Spirograph's enforced mathematical precision, my more recent work *Jazz Composition* (2018)[s105] is intended as a drawing tool that rebels against control. This tool lets the user draw into a gray background, ei-

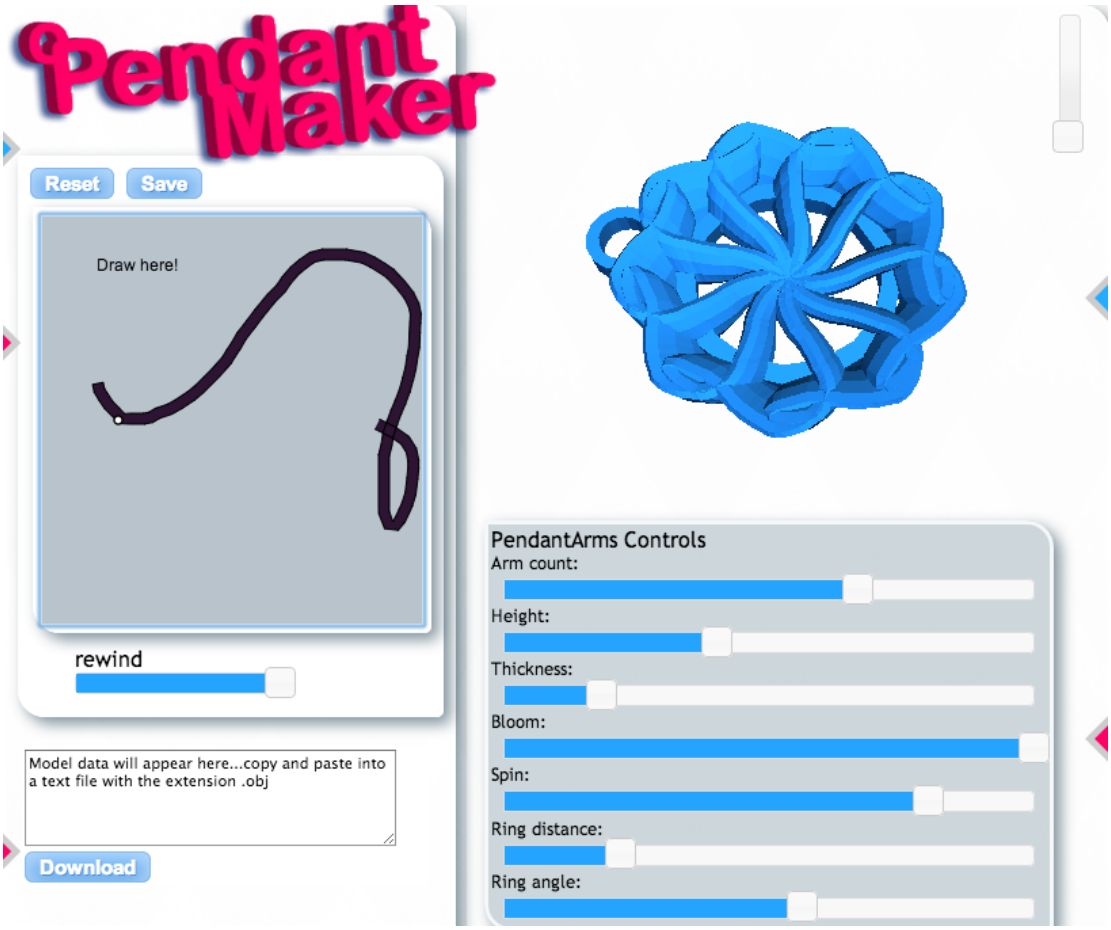


Figure 12.9: The *Pendant Maker* UI. It became quickly apparent that users didn't even look at the drawing buffer, they could draw while *only* looking at the extruding cloned 3D paths.

ther with a mouse, a touch interface, or with a Leap Motion hand-tracking device (starting and ending the stroke by pinching or releasing thumb and forefinger). When the user starts a new line, the system generates a new brush style, either a continuous line, a filled continuous ribbon, a filled region, scattered dots, or a multilayered diffusion brush made of many brush strokes offset from the original. The user only sees the tools when they start drawing, without control over it, but as they draw out the curve, it begins to pull away from where they have placed it. Behind the scenes, the stroke is a series of Bezier points, which are also attached

to simulated physics particles with a simple skeleton of springs which keep the line connected but also provided muscle twitches to animate it. So as they draw, the complexly-rendered stroke comes to life and twitches according to its own rhythm.[68]

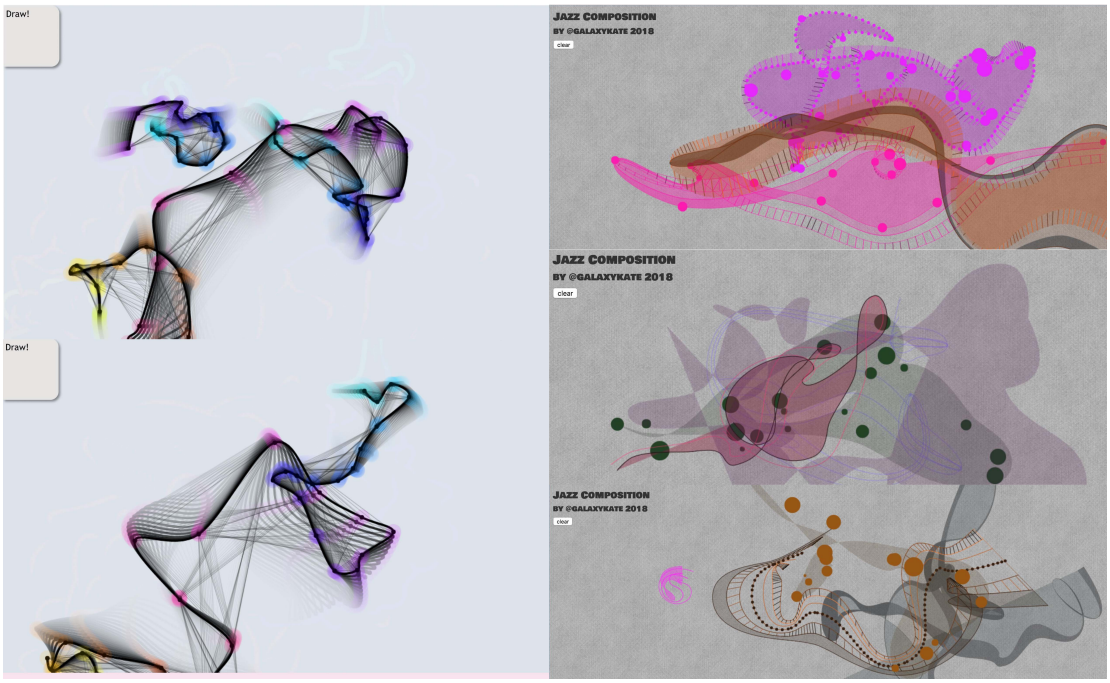


Figure 12.10: An earlier prototype of *Scribbles*[s197] (`http://www.galaxykate.com/apps/Prototypes/Scribbles/`), an earlier spring-particle drawing tool using accumulative drawing to show its motion over time (left), and several drawings from *Jazz Composition*(right).

Drawing tools are a very strong casual creator subcategory. Not every drawing tool is a casual creator: some have too little support (*Oculus Medium*) and some have too little creative emphasis (*Kirby Canvas Curse*). But for many other casual creator drawing tools, the interactive paradigm of "drawing" creates a strong and fast grokloop. The gulf of execution is neatly bridged by how natural it is to move our bodies in a drawing gesture (and how much internal proprioceptive feedback our bodies provide for that action). If a drawing tool provides continuous visible feedback of the stroke we are drawing (and for a drawing tool, it feels like a

betrayal not to have that), then the gulf of evaluation is crossed instantly (as long as we only care about aesthetics). Drawing tools almost always have a continuous groklook, which lets them get away with wild and alien logic in their generative algorithm. In the next chapter chapter 13, we will see a space of casual creators for which this is *not* the case, and how they deal with a painful slow grokloop while still eking out some casual creator experience.

Chapter 13

Slow Creators

So far in this dissertation, a major focus has been on the grokloop, the loop through which the user iteratively refines their mental model of a space through interacting with it. It should be clear by this point that much of the power of casual creators comes from the speed and ease with which users can make this loop: coming up with a model of the system, forming their hypotheses, testing them, and evaluating the results to update the model.

In the previous chapters, we have seen how this happens in a vector-based possibility space, and in possibility spaces shaped by drawn marks. In both chapters, the examples had output that was primarily visual. Visual artifacts can be evaluated very quickly. The **gulf of execution** was the primary gating factor in these systems, so the chapters discussed fast forms of interaction, like gesture, sliders, or selection, that allowed the users to quickly affect the system and move through the possibility space. In a work like the *Spore Creature Creator*[s216], as the user continuously moves the skeleton or the rigblocks of the creature, the system updates the model and dependent layers (skin, texture, animation), and the user can almost instantaneously decide whether or not they like the effect. In this chapter, we will examine creators where the step of *evaluating the artifact* can be

laborious, slow or difficult: casual creators with a problematic **gulf of evaluation**. Casual programming IDEs are slow-creator in which both the evaluation *and* the execution are difficult, but that is a large enough subspace that it gets its own chapter after this (chapter 14).

Lets examine a few categories in which evaluation is implicitly slow (regardless of their input/interaction approach).

13.1 Music

Expert musicians may be able to look at a score or audio-form to "read" a piece music, but there is no way for novice musicians to visually evaluate what a piece sounds like.

Musikalisches Wurfelspiel[s50](German for dice music)[446] is the earliest genre of music generation tools I have found. In this activity, which dates to the mid 1700s, players would roll dice to pick bars of music from a table, socket them into tile-based rows, and so create a fully-formed waltz or other piece of music. These were parlor games intended for casual users, not creativity tools for professional composers. One 1770 version was titled: "A Tabular System Where the Art of Composing Minuets Is Made So Easy That Any Person Without the Least Knowledge of Musick, May Compose Ten Thousand, All Different, and In the Most Pleasing and Correct Manner"[1] but many of the other titles likewise reference something like "ohne etwas von der Musik oder Composition zu verstehen" ("without understanding anything of music or composition"). As a physical game, players could make up their own rules, even adding a bit of an entertaining evaluation mechanic (see chapter 16):

"Of course, the point of such games was not the production of works of

[1]Quiz: How many themes from this dissertation can you count in this description?

> music, but diversion. There were a number of ways the game could be played: a group of players might sit in a circle and take turns throwing the dice, so that each successive measure was determined by a different player; players might each create their own waltzes, which could then be compared with one another; the entire piece might be written down and then performed; or players might take turns performing the piece as it progressed. In any event, the game offered a musical and pleasant way to spend a portion of an evening, with the added benefit that the music produced by what appeared to be total chance was orderly and coherent, no matter what the skill of the player"[446]

Like many casual creators, the systems were designed with a strong sense of the domain space and implicit rules to create only *safe* compositions. Also like many generative systems, they had to be written by domain experts who could create artist-in-a-box rule sets: "what was required of the compilers of games such as this one were little knowledge about how to put the game together and an understanding of the formal design of waltzes, contredanses, and the like". This embedded safety was important: the poor players of *Musikalisches Würfelspiel* would have faced a dreadful grokloop. Not only did they have to manually copy out the selected bars, they then had to perform them, and only then did they know if they had a good composition.

When music generation tools made the jump to a digital space, this step was replaced by a music synthesizer which could play the newly created compositions. A review of *Will Harvey's Music Construction Set*[s143] promises to take the "drudgery" out of music composition:

> "If you have ever tried to write your own music, you know it can be a lot of hard work. If you don't have a piano or other suitable instrument handy, you have to go strictly by theory without a chance to hear your ideas expressed. Those who usually sit at the piano and improvise until they hit upon something have to laboriously transcribe what came out of their hands onto staff paper, note by note. Will Harvey's Music Construction Set and your C-64 will solve these problems for you and take the drudgery out of composition forever. Not only does this

> program let you quickly and easily move notes around the staffs with a joystick or a KoalaPad, you can immediately hear your composition interpreted by one of the best performers in electronic music–your C64's SID chip"[4]

Music Construction Set[s143] is perhaps the earliest of a common paradigm of digital music creation tools which let the user directly place notes to be played. More recent casual-focused creators of this type (see Figure 13.1) divide up a staff of music into beats (usually 4 per measure) and let the user place different notes on the tiles. The notes are all in a single key, and snapped to the beat-marks, but the user must create their own melodies by placing notes. Virtuosic users of these systems often compete with each other for the best recreations of songs[99], but it can be difficult for a novice user to create compositions that sound *musical*, rather than like streams of unrelated notes. One *Mario-Paint*[s131] fan app that has since diverged from the original (DanielX's *Mario Paint Music Composer*[s422]) lets users create templates of notes, then repeat them. I found this to greatly increase my "music"-making ability: previous tile-based composition tools underestimated the power and musicality of being able to repeat and alternate leitmotifs. *Google Song Maker* (bottom-right)[s88] is a recent version, but does not innovate on the status quo.

WolframTones[s243] uses a similar fill-in-the-tiles approach, but control over the notes is ceded to a cellular automata. Users can reroll to find a new melody or arrangement, but can also control the instruments playing, the style of music, and the scale from which the notes are selected. These pieces are highly repetitive, with no overall structure, so to listen to one scrap of it is to hear as much as needed. However, rerolling and listening to the result requires several mouse clicks on different parts of the screen; and since rerolling is the primary way for

[2] `https://danielx.net/composer/`

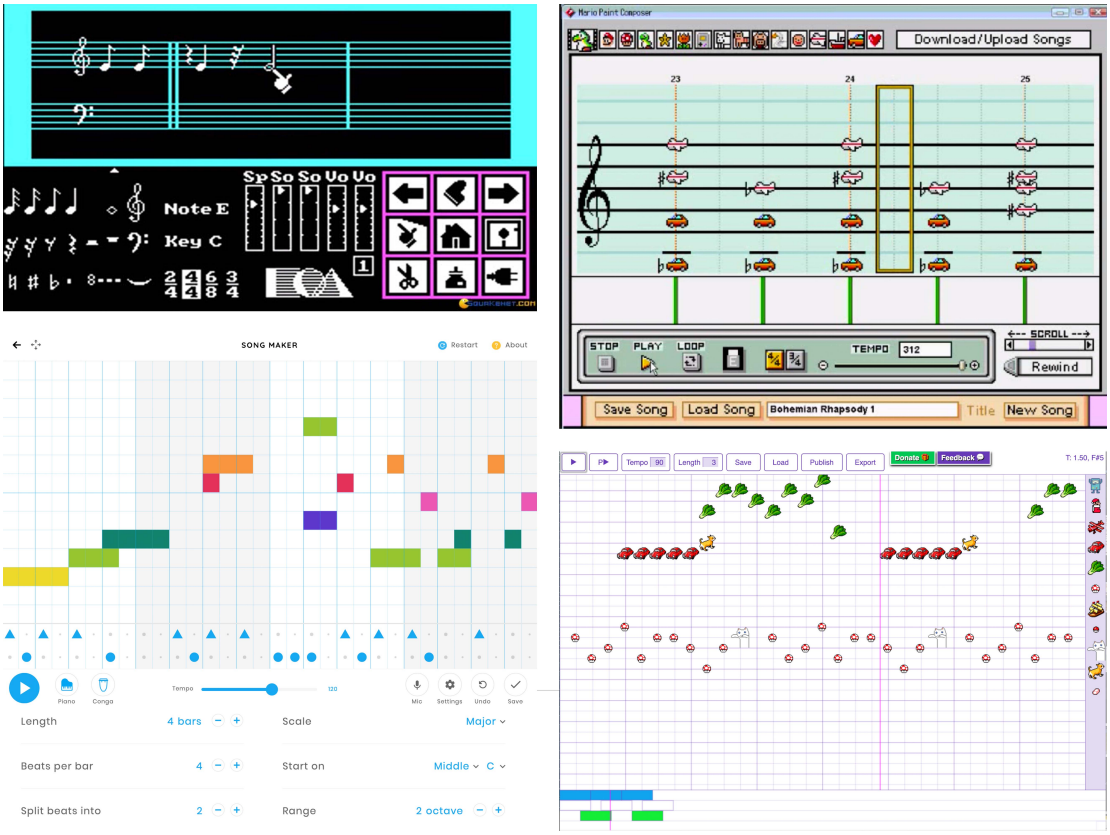


Figure 13.1: *Music Construction Set* (top-left)[421], *Google Song Maker* (bottom-right), the original *Mario Paint Composer*(top-right)[s131][417], DanielX's *Mario Paint Music Composer*) with templated sections.

users to explore the space, this has an unnecessarily long loop.

In two other indirect music apps, a variety of generative or casual creator methods can unleash a harmonically-tuned series of notes. In Scott Snibe's *Bubble Harp*[s23], a 2011 iPad update of a 1997 gallery piece[s23], users draw paths made of points on a touch pad. These points are regionized with a Voronoi algorithm[138]. The system traverses the resulting edges, lighting up each on in turn and playing notes according to their length. Adam Smith's *Ethereal Dialpad*[s68] also uses a touch interface, in one of several modes in which the user can draw curves, conduct particles, or light up a procedural grating. In all the modes, the dragging action activates a swoopingly-harmonic ambient musical swell. I would consider both

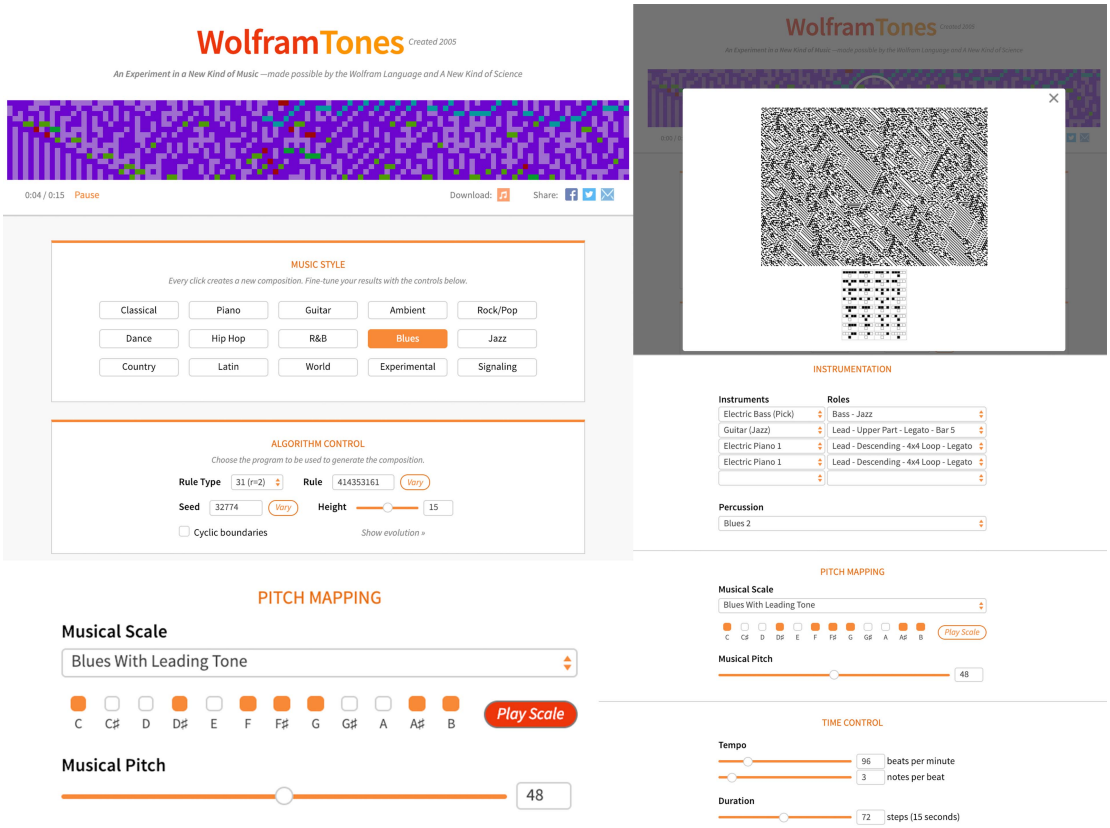


Figure 13.2: *WolframTones* and its many indirect controls

of these apps to be a class of **wind-chime** casual creators[3] in which the user's primary interactions are to set up a procedural space (scale and arrangement) and then create motion that activates the volume of activity, but not the notes themselves (like rustling a wind-chime).

4

These generative **windchimes** are pleasant, and the limited options (touch or don't touch) can create a state of flow. Several systems add *visuals* to generative music systems, either as visual/music wind-chime systems with touch interaction (Brian Eno and Peter Chilver's *Bloom: 10 Worlds*[s17] or Jono Brandel's collabo-

[3]this seems to be a subgenre peculiar to sound and animation casual creators

[4]Electroplankton[missing system:electroplankton] is another phenomenal example of generative interactive wind-chimes, but has too much complexity to cover in this chapter.

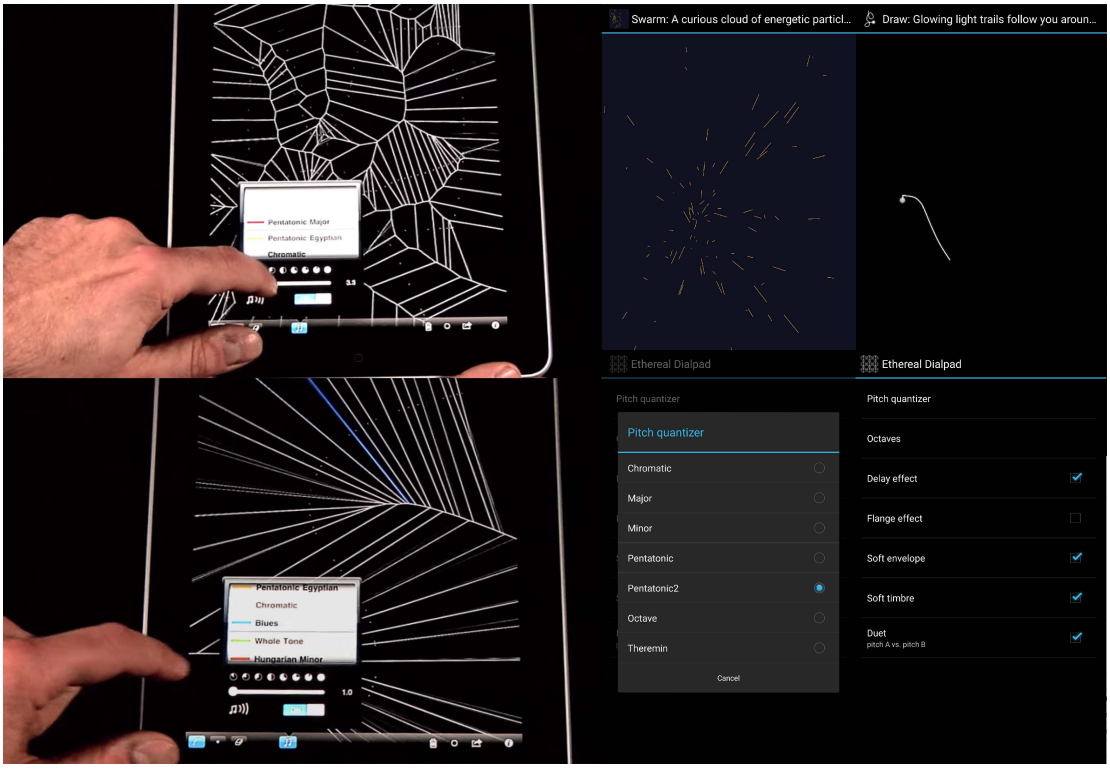


Figure 13.3: (left) *BubbleHarp* and (right) *Ethereal Dialpad*, two semi-autonomous gesture-based music apps.

ration with Japanese electronic musicians Lullatone, *Patatap*[s163]), or as systems where the user *only* controls the settings of the music and images, like the game *Panoramical*[s161]. In both *Patatap* and *Bloom*, the user taps the screen (or presses keys), and with each press, a note and graphic element co-occur. The user may have little control over what graphic or where it appears, or what note, but they control the timing, and that leads to a very direct feeling of "controlling" the music. All of the difficult animation, note progressions, and harmonics is left to the generative system.[5]. In contrast, *Panoramical*'s only controls are a set of 9 parametric sliders, each of which controls some aspect of the generative world that the player is traveling through (at a fixed speed, on rails) and the generative

[5]A friend reported that her toddler enjoyed "playing" *Patatap* by kicking the screen. While he was too young to appreciate the music progression, he understood the cause-and-effect of "kick screen, get noise"

sound. By sliding one slider, bass kicks might get louder and jagged red rocks poke through the environment, but sliding another one might turn the bass into chirps, and retract the rocks in favor of a flock of glittering particles.

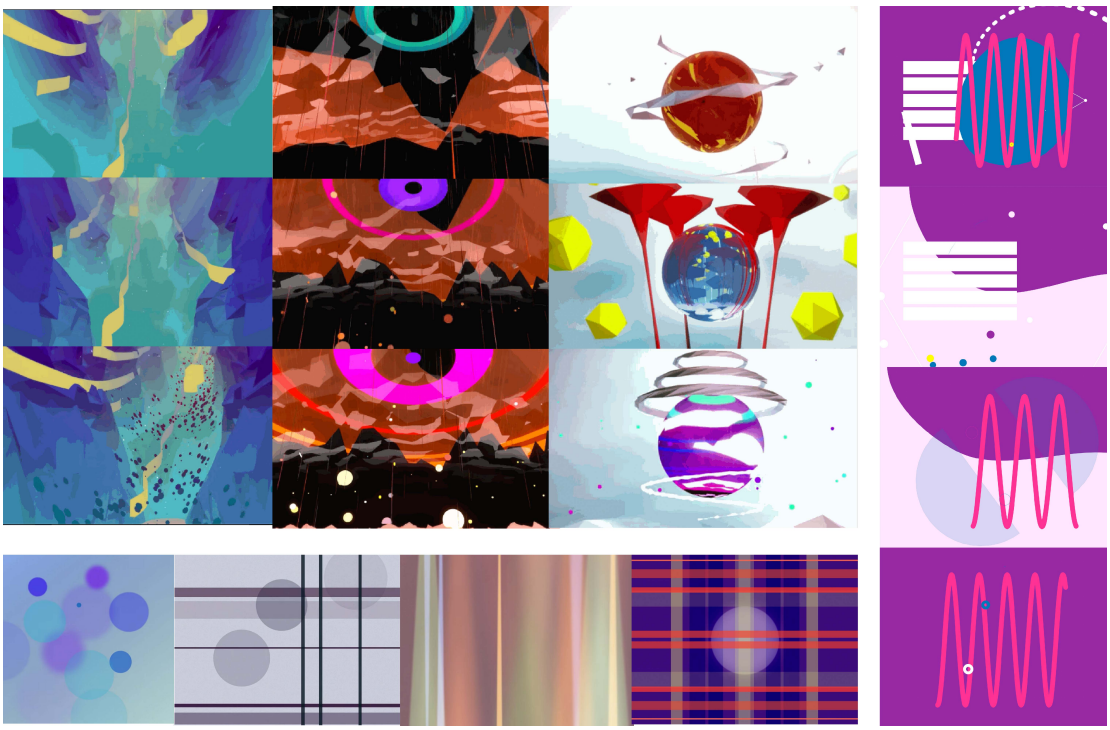


Figure 13.4: (top-left) Some of *Panoramical*'s many worlds. (right) *Patatap's* geometric shape animations for a single setting (it has multiple color and sound profiles to toggle through). (bottom) Four of *Bloom*'s ten worlds, which are the least complex and least interesting of these visualizations, perhaps to not outcompete Eno's music.

Google's *Otto Fischinger doodle*[s156][160] is a rare example which combines tile-based direct control with the indirect control of generative animate artworks. In this doodle, which celebrates the work of mid-century abstract animator Otto Fischinger, users place "notes" on a tiled/socketed musical staff, like *Mario Paint* or the *Chrome Song Maker*. Instead of just playing a sound, they play a sound and an abstract geometric animation sequence. This attempts to make an abstract geometric animation that expresses the music, as Otto Fischinger's animations

were famous for.

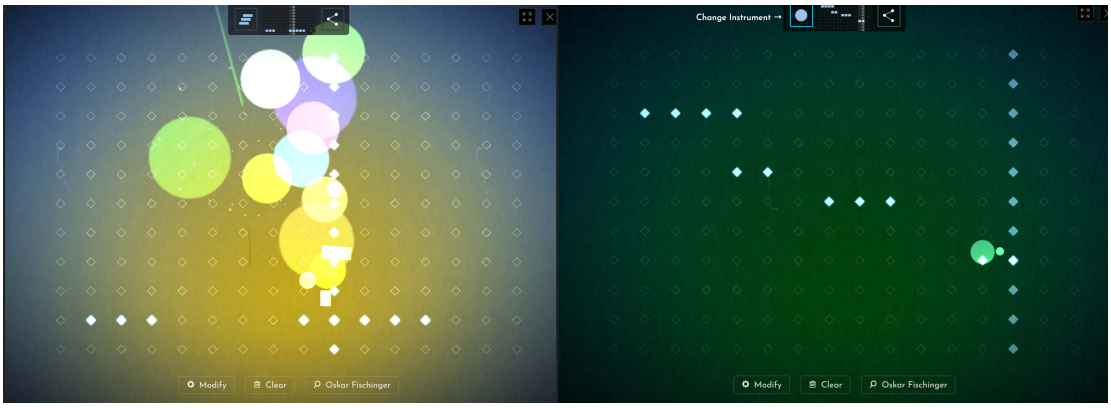


Figure 13.5: Google's *Otto Fischinger doodle.*

It is noticeable that with the exception of the *Musikalisches Würfelspiel* and *Music Construction set*, none of these are made to make complete pieces of music, or pieces that are more than a handful of bars long. Music must be experienced over time, with the time depending on the length of the piece itself. A short audio blip is faster to evaluate than a symphony, and a pop song's hook (or *WolframTone*'s melody) would be somewhere in the middle. For continuous music (like wind-chime creators), the overall structure is never viewed, but the user is kept in a perpetual state of flow, only remembering to a short horizon behind them, and making actions to impact the music only a short time ahead.

Is there a way to casually-create long works, or is it best to stay with the proven patterns used to make short works? Musikalisches Würfelspiel may hold one answer: if the genre is constrained enough, then the user will feel confident that their contributions (or lack thereof) will not make much difference, for better or worse. Perhaps the closest modern equivalent to *Musikalisches Würfelspiel* is Google's *Giorgio Cam*[s85]. While it runs on top of Google's state-of-the-art machine learned image recognition networks, rather than paper and dice, it has a top down structure that does not vary, and templated sections of text that can

be socketed in. Instead of a waltz, it produces a bad text-synthesis based on the image-labels it calculates from user provided photos ("Got DRINK in this spot, could be DOG-LIKE MAMMAL but maybe not"). Having played it at parties, its interaction style of pausing the rap to wait for the next photo input makes it amenable to ad-hoc play structures (passing it from person to person, deciding as a group what to use for the last photo), just as we saw with the dice music 300 years ago. Perhaps it *is* possible to make a casual long-form generator, but it requires a significant measure of both safety and playfulness to get people to spend 3 minutes listening to a robot rap.

13.2 Generative literature

NaNoGenMo[s1456] is an annual exercise where creators build AI authors that can generate novel-length text. As part of building a generator, a creator must occasionally evaluate what their AI is producing. Do recent changes in code make its output worse or better? But how can creators evaluate *a novel*? Novels notably take a long time to read, so NaNoGenMo authors must evaluate their generated novels in an abbreviated, lossy, compressed form. Could a creator make a system that has subtle shifts in narrative, language, dialog that would require *actually reading* the novel to evaluate? Or are we limited to only generated novels which can be adequately understood in small quickly-evaluated pieces?

Like we saw with music visualizers, the most common success stories in this space are the short, snack-sized versions of the medium. Some like Max Kreminski's *Blackout Poetry* generator[s15][226], Emma Winston's *Blackout Poetry Maker*[s14][437], and *Erasures*[195][s67] take text input from the user and then let the user playfully decay it, either word-by-word (Winston's), randomly (*Erasures*) or constructing

[6] `https://nanogenmo.github.io/`

viable sentence paths through the words (Kreminski's). All of these have some elements of casual creatorness, whether they are trying to surprise the user with generative (or random) refigurings of the text, or acting as an art tool to nicely-format and style hand-selected blackout poems (Winston's).

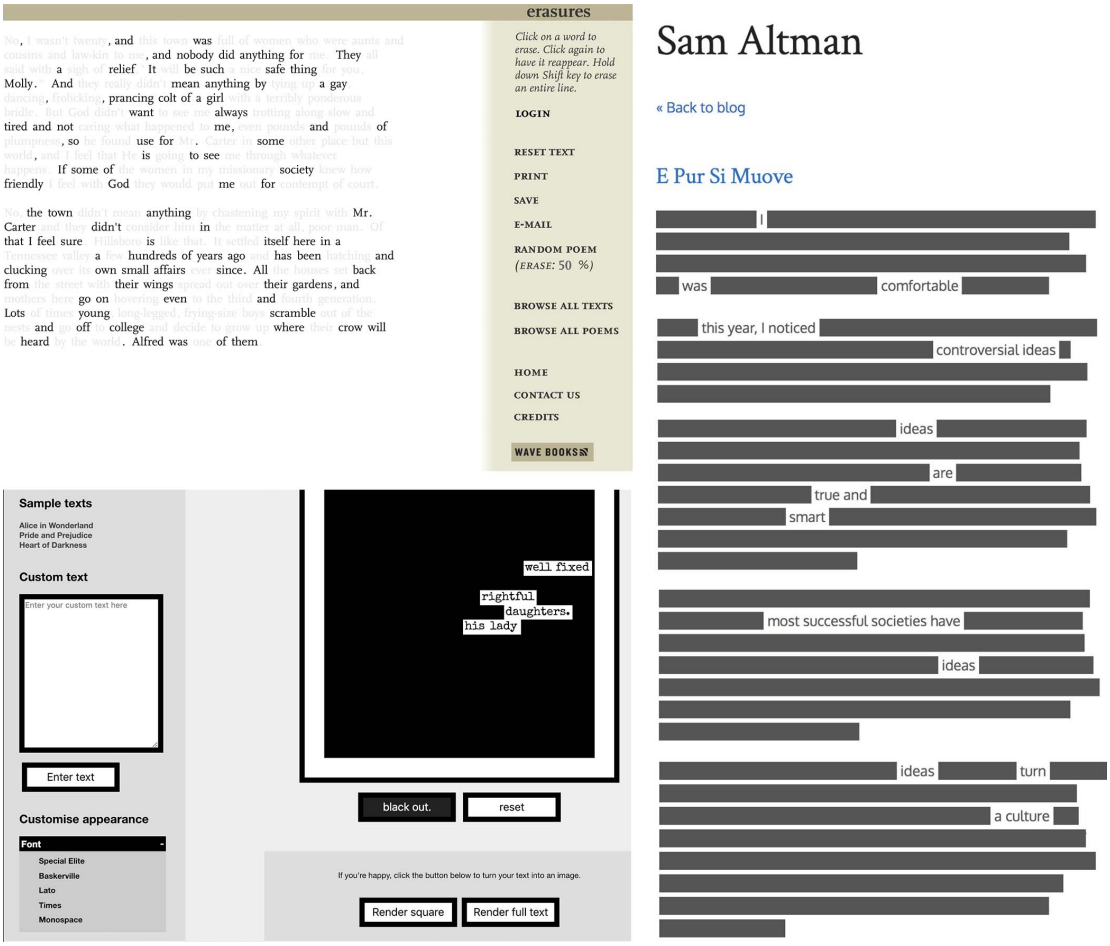


Figure 13.6: (right)Max Kreminski's blackout-generator bookmarklet and poems (`twitter.com/maxkreminski/status/941466797420638208`), (bottom) Emma Winston's blackout poetry maker [437], (top) Erasures.

So, too, are there physical or pencil-and-paper versions of literary casual creators, such as *Mad Libs*[s128], a 1958 game that started as a creative exercise between two television writers[342]. In *Mad Libs*, one player reads off parts of speech or other types of words ("body part"), and the other player or players suggest values

for them ("nose"), which the first player writes down. Once all the blanks are filled, the story is complete, and the players read it aloud, with hopefully humorous results. The narrative structure is provided by the template, but specifics are provided by the players, so the resulting story still has a beginning, middle, and end.

A different form of structure is provided by Raymond Queneau's *"Cent mille milliards de poèmes (Hundred Thousand Billions of Poems)"*[312][s181], in which readers are encouraged to cut apart the poems lines into strips and recombine and reorder them. This work was part of the Oulipo movement, 1960 French movement formed to discover new possibilities through artificially constraining the creative process[399]. Like the *Musikalisches Würfelspiel*, any recombination is a valid poem *because* the lines were carefully written by a domain expert who knows that lines like "Till firemen come with hose-piped tidal wave" and "Replies like this the dumbstruck brain may tease" reference the lines before or after them, and so conjure up logical relationships in the spaces between lines. Of course, this approach will not create longer structure, nor is there a way for the system to enforce that structure. Users place lines until they no longer want to place lines, and this is unlikely to lead to structural punchiness, at least with non-expert users.

One recombinatorial poetry project that *does* enforce some structure is the *pentameter pentominos* project[s165][305] from the Puzzle Poesis research group at the University of Virginia. This project attempts to tile pentominos over the syllables of Shakespeare's sonnets such that when cut apart, each word remains unsplit. Users can attempt to reassemble them as a puzzle, but when I played with a set, I found that there was physical satisfaction in rebuilding the puzzle *not* in its original order. Each word that I built onto my little poem came with other words attached in the pentomino, which I than had to leave orphaned on

their own line, or find another piece to hopefully close out multiple lines at once. I found this to be a more flow-inducing exercise than unconstrained refrigerator poetry[s186] (its closest relation), as I had fewer, more emergent choices.

This created local structure (and encouraged accidental iambic pentameter), but still did not create overall structure. For a proposal for that, we can turn to another Oulipian. In "Prose and Anticombinatorics", Italo Calvino proposes a labyrinthian computational tool to calculate possible crimes given aesthetic and functional constraints (one cannot sin against oneself, "favor those solutions that are the most harmonious and economical."). From the perspective of those of us with a view of the intervening 33 years of failed constraint-based plot generators, this may seem like a ridiculous system, but the desires that fuel this fantasy are real, even if the fantasy were not:

> "This clearly demonstrates, we believe, that the aid of a computer, far from replacing the creative act of the artist, permits the latter rather to liberate himself from the slavery of a combinatory search. allowing him also the best chance of concentrating on this "clinamen" which alone can make of the text a true work of art."[48]

What technique do we need for building *long-form* stories? We could imagine tools for people writing novels, but as we have so often seen from other casual creator spaces, we *can't* build tools unless we know the domain space, which usually requires *abstracting the domain* ("music is a series of notes on a staff") or taking only a narrow slice of the domain with known rules ("1700 waltzes"). *Auctor*[s9] does the first, by representing story as character. This app lets users create lists of characters and assign them to projects. Each character has spaces for story goals, motivations, height and hair-color, and can be randomly generated. But these generated characters have very little personality or style on their own, and though the system also has provocations to encourage the author to move the character through an arc, they are all very general, abstract, or trite (hero's jour-

ney). The other approach (narrow slice) there has been a renaissance in guided storytelling games, with a diversity of games about very specific moments in a preset story (*Bluebeards Bride*[s18]), trope-heavy genres (*Lasers & Feelings*[s118]), or story-telling approaches (*Fiasco*[s73], *Dread*[s55]). Each of these systems uses their narrow scope to create mechanisms that guide players into telling good stories, often providing robust support for character motivation arcs and satisfying conclusions (one of the hardest storytelling tasks). So far, most of these examples are non-digital, as face-to-face negotiations and physically manipulatable cards seem to be more important than computational support. Ian Horswill and Ethan Robison[199] have experimented with using computational support (SAT solvers) to interrogate players of the game *Dread* about their characters, to support what they see as a common problem of players not fully expanding the implications of their back-story. We may someday see a casual creator with a generative pipeline that "solves" user-provided stories as part of the interaction, if systems like *Storyteller*[s221] are released. However, we already see casual-creator approaches to solving parts of story-telling that humans have trouble with.

So far the approaches that we have seen have used small amounts of generativity to supply suggestions or prompts to the user, who then writes the story. Are there systems that *generate* full readable text itself? There are certainly systems that generate full stories[307], but these are almost never interactive. I wanted to know about how users could interact with a story-telling engine, so I put the question to Twitter users who had participated in NaNoGenMo[s145], the "national novel-generation month" competition to build generative novels of 50,000 words or more. While these systems are not interactive, the *process* of making and tuning them is. If each kind of generator that people were building were a possibility space (procedural surrealist travelogues, for example), how did the authors of the

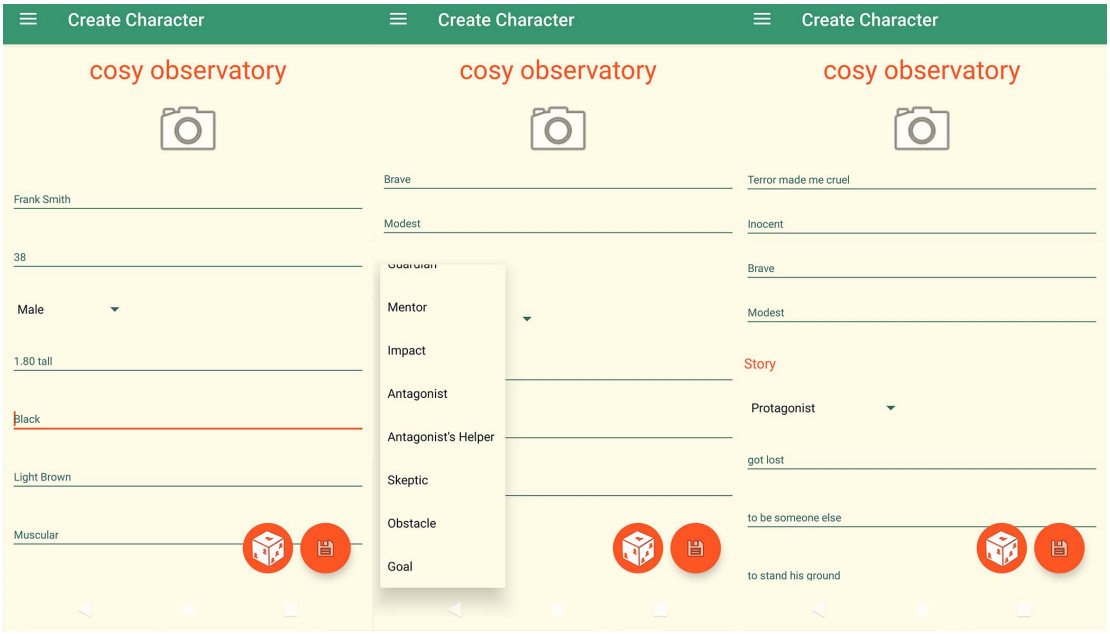


Figure 13.7: *Auctor*, an Android app for generating and tracking collections of characters.

systems explore the space? We can assume that their interaction with the system was through writing code, but were there any visualizations or structural techniques to bridge the gulf of evaluation? These are a few of their answers:

> "I mostly look at small sections of output, most of my NaNoGenMos aren't really trying to be large-scale coherent. I've made tools in the past which count the expansions of particular grammar nodes etc, so I can see which ones need more variety." Martin O'Leary @mewo2

> "Mine (We Verb) was *extremely* simple, so I mostly checked in on sections to see if things were as I expected them to be. I used a list of verbs in English, so I'm *reasonably* confident of the output, but I definitely couldn't tell you that it's 100
> So I guess in my case my technique would [be] working with a corpus with known content and spot-checking to see that things are running as expected. For context, I was creating an automated novelisation of John Riddell's poem "We Verb", and so all I needed to do was follow his structure of "We <verb>" until I exhausted the corpus." @mattlaschneider (Matt Schneider)

> "Ours (with @ScientistSloth) had 4 broad sections with content ramifications, so those each needed regular checkups separate from the



Figure 13.8: "Coen brothers movie"-making social game *Fiasco*[s73] and *Lasers & Feelings*[s118], a one-page story game about Star Trek fan fiction.

page-size bits As in @ScientistSloth made a little tester that would gen (insert int) "bits" and a dropdown would tweak them based on which of the 4 sections you wanted to test" @tanyaxshort (Tanya Short)

> "I'm still working on the thing I started for the last nanogenmo. It is now a specific set of pre-seeded pages. Previous nanogenmos have had similar enough structure throughout that a small sample can stand in for the whole." @v21 (George Buckenham)
>
> "In December we spend all of our time creating engines that can read an entire generative novel written by another bot, and then automatically generates a podcast in response." @DavidLublin (David Lublin), who was unfortunately joking
>
> "What I did for Annals: tune for small sections, retain early iterations as the beginning of the work so that the long form shows the evolution of the generator For tarot of the Parrigues I was tuning small sections as well (card descriptions), but even the long form there is not super long. Another thing I have thought of but not done: a grammar that only expands partway at first, allowing inspection of high level structure until you are happy with that and grant it permission to continue" @emshort (Emily Short)

For most of them, we see the same technique of systems like WolframTones: create a system that is mostly uniform and continuous (or has divergent subdivided sections that each have this property internally), so that the entire space is approximately evaluated by looking at a small sample. Emily Short's approach had the interesting outcome of recording the *path* through the possibility space as she iterated on her generator, and preserving that as the structure of the work. David Lublin's tongue-in-cheek suggestion is actually the most intriguing of this set: can we have *some virtual agents* performing the evaluation work for us? The least-fun and most familiar version of this is the spelling-and-grammar checker of any word-processing software, but can we make that more fun, playful, and generative?[7] As we will see in chapter 16, for a casual creator, inaccurate and even unhelpful evaluations can be fun *in an autotelic setting.* A writing tool that generated reviews or feedback from a crowd of virtual readers would be interesting, though listening to a podcast would be nearly as long of an evaluation gulf

[7] Is Microsoft's Clippy a casual creator?

as reading the output.[8]

13.3 Slow creators: Game-makers

Almost every game player has a dream of making their own game, but game-making (especially digitally) is technically demanding. Over the years, many *many* game-design tools have been built and released, and many of them target a casual audience. But as many game designers know, *building* a game is only a first step, and play-tests and iterations are a brutally hard yet indispensable part of the game creation process. How long does it take to evaluate whether a game (or a game component, like a level) is good? Some content may be evaluatable visually, like icons or character designs. But rule sets, maps, tuning, and other systems often require repeated play-testing to evaluate. Even expert designers, who can evaluate level designs at a glance, can be surprised by the outcome of testing[379]. And game testing can require anywhere from a few minutes of casual play by the designer themself to hours or days of play from many players, especially for heavily-competitive games that need sophisticated balance control. Adding a new *Magic: the Gathering*[s142] card can take a full year of iteration and play-testing.[327]

As we have seen in literature and music, there is no such thing as a robust general purpose casual creator for "general" games. "Games" covers a vast range of experiences: chess, *Fortnite*, Twine games, *Panoramical*, *Sim City*, *Hatoful Boyfriend*, *Hypernom*, and *Her Story* are all games, but it is impossible to imagine a tool that is useful in creating all of these, much less an approach to automated

[8]The code has since broken, but one #Nanogenmo, I prototyped a "You've already won NaNoGenMo" casual creator. A recursive Tracery grammar created nested Scheherazade stories, which by default could be thousands of words long. The user could select subgrammars from a menu (cowboy, scifi, ancient Rome) to swap out, then save out the resulting story (to win NaNoGenMo) or the grammar, to continue editing it in a less-constrained editor.

assistance in all cases. Many "easy" game tools aim to reduce difficulty and complexity, like *GameMaker*:

> "MAKING GAMES IS FOR EVERYONE: We have done everything possible to streamline the development process and give you all the resources you need to get started [...] Making games development accessible to everyone means taking away the barriers to getting started. Using our intuitive 'Drag and Drop' development environment you can have your game up and running in a matter of minutes without ever having to write any code! GameMaker's built-in language (GML) helps you learn to program as you go and not jump in at the deep end of coding."[145]

GameMaker may be only a littler closer to a casual creator than *Unity* is. But it does exist at the middle of a spectrum, which extends from general-purpose coding tools to casual creators for domain-specific game-making. *GameMaker*'s ontology is based on spatial objects with some internal properties, that interact with collision mechanics and each other, all rendered to a screen. It can offer some assistance (visible objects default to having collision physics) so that a "drag-and-drop" approach can create simple games.

GameMaker's place in the spectrum is shared with many other systems designed to make particular kinds of games. These exist for every genre of games: RPGs (*RPGMaker*[s193]), hypertext games (*Twine*[s236]), visual novels (*Ren'Py*[s187]), idle games: (*IdleGameMaker*[s96]) and the oldest digital casual game-making tool: *Pinball Construction Set*[s171]. Like *GameMaker*, these make assumptions about the possibility space, guide the user to work within those assumptions (through UI and limiting model possibilities), and create computational support for running (evaluating) the artifact that user is building. These tools are usually not *primarily* autotelic, and they have a lot of empty, meaningless, or broken possibility space, and lack automation or generativity to help players move rapidly through the space. Whether or not we want to consider any of these as *casual*

creators, we can use a casual creator lens to examine the way that they trade a large possibility space for greater domain-knowledge and generative assistance.

Some game-making tools move further in this direction by creating a very limited possibility space within a game genre. *Sketch Nation*[s209] is an examples of a game-maker where the possibility space has been constrained to a safe quality, in these cases, of few templates of popular game genres. *Sketch Nation* has a number of template games with fixed gameplay logic, including match-3, *Doodle Jump*, and *Flappy Bird* clones. It allows the player to select from 3 different creation-difficulty levels. At the lowest level, users simply replace the graphics in a given game, and can play "their" game as a reskinning of an authored sequence of platforms. At the highest level, they can set a point-condition and draw their own levels out of tiles that they define (even giving tiles special movement choreography). This starts the user at a safe, playable space, and only lets them have access to game-breaking choices when they have chosen to play at an advanced level.

These games still have little to no automation, either in execution or evaluation. Several casual creators have emerged (from academic origins) as examples of ways that *automation* and generativity can be used in a game-maker. *Game-o-matic*[s79][413] is an editorial game-maker, which attempts to guide users to design games that make procedural arguments (i.e.: "taxes eat productivity, government makes taxes, get rid of government"). The user builds a graph of verb-relationships between game objects, and *Game-o-matic* uses its robust library of logic about collision-mechanics (and which verbs they can represent) to create a game in which all the verbs of the graph are represented by game mechanics. It also contains a solver constrain the mechanics to a set that has an achievable win condition and a plausible lose condition. Once the graph has been built (and graphics chosen), the user's exploration of the game space is through hitting "try another". This

shortens the gulf of execution to a single click, though the gulf of evaluation still requires reading the new rules of the next game, figuring out what to do, and playing it. While the system can easily juggle new mechanics, it is difficult for players to instantly pick them up.

Wevva[s241] is another casual creator for game-making (one of few that explicitly have adopted the "casual creator" label). This project is one of three related systems (*Wevva*, *Cillr* and *No Second Chance*)[283] that use mostly-parametric representations[63] of game properties to map games within a parametric possibility space. This space isn't continuous (there is no midpoint between "give points when hit" and "end game when hit"), but it can be navigated by randomly re-rolling a game, or by locally exploring the space around a randomly-created (or previously authored) game. One of the designers (Michael Cook) narrates a tutorial video in which he rolls a random game and attempts to fix it up into something nicer:

> "This game has just been put together for us and we don't know what it is in advance but it might inspire a really cool idea for a game. Looks like when I tap I can move this circle around? And I can actually catch this bug in this circle, and that stops it from...touch a butterfly... maybe? So maybe there can be some kind of game where I have to things inside this perfectly.. maybe? Seems a little be hard. Let's see what we can come up with. So I like this idea of moving around when you tap it, but I think I want to make it a little bit bigger so its a little bit easier. *he changes the circle size* Its kind of like we're putting a pen around these creatures, so instead of insects maybe we want to try an animal that you herd or look after, maybe cats or dogs or something like that *he changes the bug graphic to a cat.* And we'll make them a little bit smaller as well so the don't dominate the screen too much."[270]

Like *Game-o-matic*, the gulf of execution *can* be reduced to a single click, and thus exploration by random sample, *or* in this case, the user can also navigate with sliders. While there are many ways to navigate a parametric space (as seen

in chapter 11), this tool only enables randomly sampled game, and slider-edited ones. In their other tool, they recognize that some game space may be too sparsely populated with "good" games to be search-able.

> "Randomly mutating multiple sets to produce a new random game, and then trying to figure out what it is, can be a fun interaction loop. If you aren't a researcher interested in design spaces, however, the proportion of playable games remains too low for the mutation approach in Cillr to be ready for end-user consumption. "[283]

They *do* use one of the most powerful techniques for a low good-density possibility space: users are encouraged to discover good games, even if that can be difficult, so that they can share them and "own" the discovery of that game. Aside from motivating users through a difficult not-so-autotelic experience, this seeds the sample-game database with a number of new landmarks, from which new players can begin their explorations. Even in game-makers that do not have a parametric space this "social reward for good new landmarks for re-editing" pattern seems to be common and motivating, as in games like *Super Mario Maker*.

These systems have made good progress towards solving their gulfs of execution, either through a traversable possibility space or generative power. Games, however have a *killer* problem in crossing the gulf of evaluation. While *Game-o-matic* can produce many games, all of which it has calculated to be solvable, it cannot tell when a game's mechanic is intuitive enough (through some magical combination of graphics, UI and metaphor) to be immediately understood. Part of this issue may be a balance of how much we **trust** that a generator's artifacts will be **worth our time** to play-test. In *Game-o-matic*, the play-testing time is low, but the trust is even lower. *Wevva* may succeed better because while its play-testing time is the same as *Game-o-matic*, the higher-quality graphics, more traditional mechanics, and the player's own involvement in the tuning have given them more reason (and need) to engage in the play-test.

Ludi[s126][39], a generator for *Go*-style games, can make a large possibility space of possible rule sets. It also has a simulator for playing virtual tournaments for potential games, so that by the time a game is surfaced to *human* players, the players have confidence that it will be interesting, and so are willing to invest time in learning its rules and strategy. While *Ludi* isn't interactive (except to its creator), it provides an example strategy for dealing with a arduously-long grokloop by building enormous levels of trust in its AI to find good-enough-to-try artifacts.

Game-o-matic has calculators that can approximate how *good* a game is, but may not be accurate enough to create confidence that any given game is worth the few minutes of our time to learn. If we improve that calculation's accuracy, can we overcome that hurdle? This may not be an approach for so-called "general" game makers, but for games where the rules are known and familiar, systems have developed more advanced, useful, and accurate metrics that may help build *safer* and more *trusted* casual creators.

In Gillian Smith's *Tanagra*[s225][384] system, the user and system participate in a mixed-initiative editing of a platform game level. The system has four tasks:

> "1. Autonomously create levels in the absence of designer input. 2. Respond to designer input in the form of placing and moving geometry. 3. Respond to designer input in the form of modifying the beat timeline. 4. Ensure that all levels are playable."

To trust the system as a collaborator, the user must know that it can appropriately follow guidance on things like leniency and rhythm. While the rhythm can be seen visually in a level, *Tanagra* must calculate the leniency in a way that is invisible and relies on the user to trust its calculation, or engage in a lengthy play-test.

One advantage of my older research in generative platformer levels[84] (one of the inspirations behind Smith's *Tanagra* project) was that all of my difficulty

calculations were performed by calculating jump arcs. To help my debugging, I visually drew the arcs that the player would make by jumping from any point in the platform. I colored them red for failure and green for success. Thus, for any platform jump, I could interpret at a glance how much of the platform was "safe" to jump from. When I could see the logic behind the system's ability to calculate difficulty, this visualization gave me the confidence to playfully explore the space without play-testing. Remember that we want to move the user through the space *not only to make a valid artifact* but to *give them confidence in their knowledge of (and ability to navigate through) the space*. A brilliant helpful AI seems like a win for usability, but we need to make the user view it as a part of the system that they can understand, trust, and predict. Even for high-tech AI support tools, *visibility* may be better than *accuracy*.

Sentient Sketchbook[240][s199] is an editor for simplified *Starcraft* maps that has many strong casual creator patterns. It lets the users "shop" for variants in the local possibility space, or edit the map directly by coloring in pixels. Like *Tanagra*, it has a limited domain that it tries to model, and has *six* traits that it can measure (resource safety, safe area, exploration, and "fairness" for each of those values). It also can display those calculation as well as navmesh, unused areas, and choke-point analysis. The variants that are presented are chosen by an AI, but it is unclear if the user can see all the overlay information on them before selecting them. Either way, *Sentient Sketchbook* is an example of a system that has developed casual-creator-friendly solutions for bridging both the gulf of evaluation *and* execution for a rapid grokloop, a rarity in game editors.

What is at risk when we create game editors that have slow grokloops? In *Spore*[s215], we found this out the hard way. After the wild success of the *Creature Creator*[s216] (and reasonable success of the other editors that shipped with

the game), an expansion, *Spore Galactic Adventures*[s217], was planned that would add two new editors to the game, a planet editor (an interactive version of my planet-generating brush-scripts) and an adventure creator. Many of the same team worked on both the *Creature Creator* and the *Adventure Creator*, but the *Adventure Creator* was a terrible flop. Users were publishing broken, unplayable levels, and even when we tried to build our own levels, we found ourselves making brittle, hard-to-iterate, conservative creations. The studio never shipped another *Spore* expansion.

What was the difference? The creature creator had been designed around safety and rapid iteration. One *couldn't* make a broken creature. By contrast, between the planet editor and adventure creator, there were hundreds of potential game-breaking things that could happen. Terrain could be too steep to climb, or have wells that players could get stuck in, or simply look worse from the ground than they did from the gods-eye editing perspective. Quests could be uncompletable, or hard to navigate, and there was no way to tell the player that a large quest's graph of conditions was broken or impossible. Even the navmeshes were secretly and invisibly calculated, so players got wedged in the geometry of player-created towns. We offered players all the *parts* that they would need to build a game, but the editor did not enforce how those parts fit together. And worst, our grokloop (if anyone ever play-tested their game) was minutes to hours long. No-one iterated 1000 times, as they did with the creature creator, nor did they have the confidence to believe that their level was worth playing through *once*.

One motivation of this dissertation was to discover *what went wrong* on the *Galactic Adventures* expansion, that turned a team of casual-creator pros into a team whose products made players doubt themselves. In this chapter, I have argued that **slow creators**, while not impossible to make as good casual creators,

need a *very* careful approach to succeed.

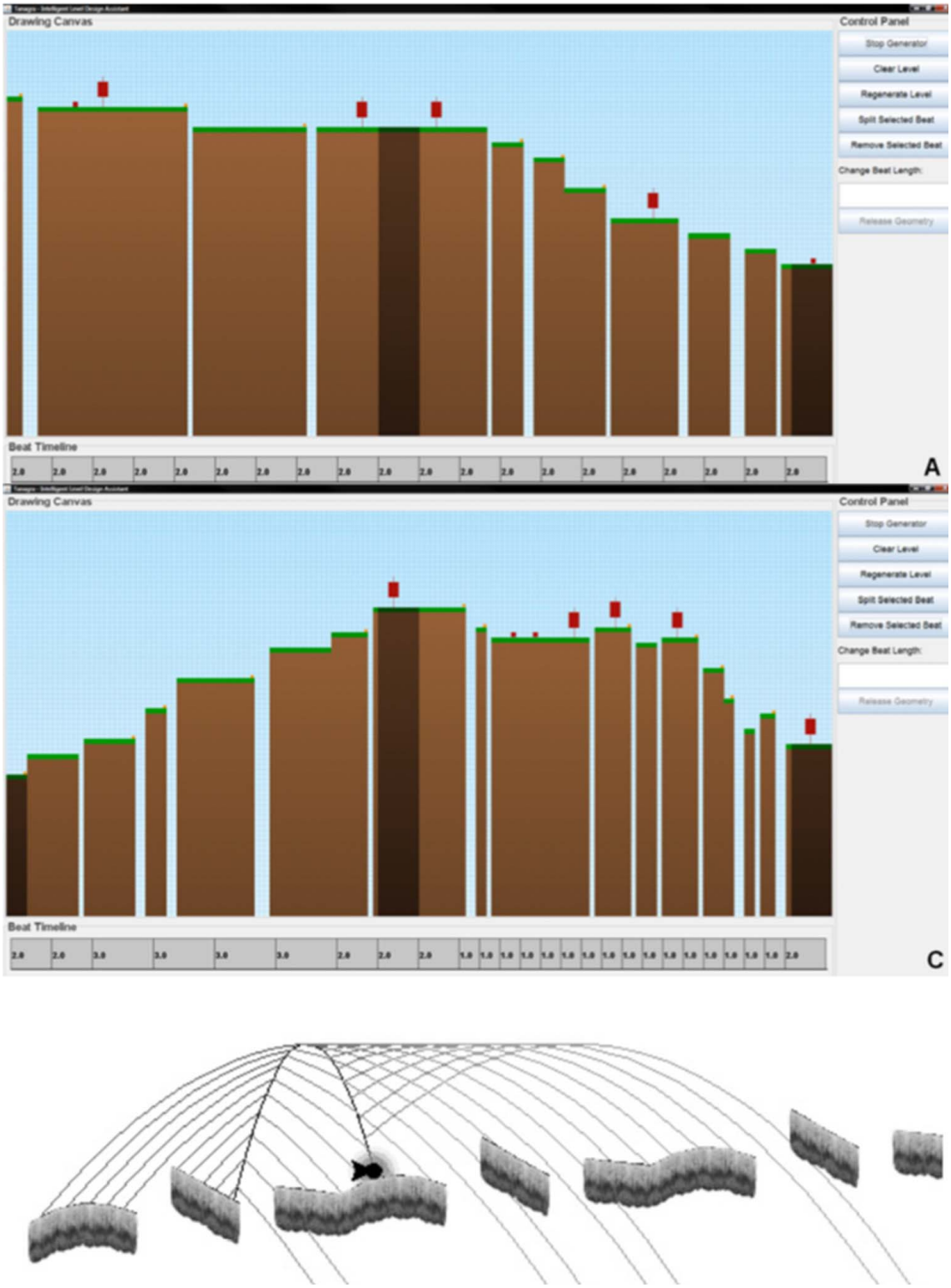


Figure 13.9: *Tanagra* and an image from my master's thesis on generating platformer levels. *Tanagra* is the smarter system of the two, but it has fewer ways to prove that visually to the user. (sadly, all the color images of my original system have gone missing, but the trajectories were colored to show successful and unsuccessful jumps).

Chapter 14

Casual Programming

> "Simple things should be simple, complex things should be possible."
> - Alan Kay

In the previous section, we explored the ways that "slow" tasks can still benefit from casual creator interventions. Even when a type of artifact is slow to produce or edit (the *gulf of execution*) or slow to evaluate (the *gulf of evaluation*), a casual creator system for that artifact can begin to bridge those gulfs with shortcuts like partial or entertaining evaluations or structured or generative creation.

Writing code is a task that is highly socially valued, yet acknowledged to be difficult, painfully dull, and stressful. Consider the many educational efforts, from historical attempts like *Logo* to the contemporary project of Hour of Code, that attempt to remove the intimidating aspects of coding (like syntax errors or typing) while simultaneously foregrounding the generative and emergent properties of programming. Languages (or platforms) designed for this purpose often exchange the **flexibility and control** of general-purpose programming languages like C or Javascript for **safety and power**.

Does this sound familiar? It should: **we have arrived again at the definition of a Casual Creator!** Lets examine a few languages and platforms that

exist in this space to see what properties they have and what features they have included (and excluded) to serve this purpose. In all of these systems, there is a *language* that defines a possibility space of a program, but nearly all of them (with the exception of *Logo*'s original form) have an associated IDE that structures how the user inputs the program and how the program is run and evaluated. Programming language specialists are used to considering a language as a pure abstraction, apart from its IDE. However, the language's properties define what kind of IDE can be constructed for it, and the user experiences the language *through* the IDE (especially for casual IDEs). To analyze these systems as a casual creator, we must look at the *experience* of programming *within* an IDE. They must be considered as a whole system.

14.1 A Few Existing Casual creator languages and IDEs

14.1.1 Logo

Logo[s1251] is technically a general purpose programming language [298], but it is better known as the language with which students move a turtle around a screen, leaving behind a glowing trail. The turtle can be either a triangular icon or a physical robot, but the purpose of both the virtual and physical turtle is the same: to enable what Papert referred to as *body-syntonic* and *ego-syntonic* reasoning. In *body-syntonic* reasoning, the beginning programmer uses their previous experience of their body, the ability to move forward or backward in space, the ability to turn left or right in different amounts, and what those two abilities can combine

[1]Papert capitalizes it as "*LOGO*", while most modern writing uses "*Logo*", as it is short for "logos" or logic, rather than an acronym.

to produce. You can ask a child to imagine a figure skater leaning as they move forward and ask them to imagine what path they would draw, or how fast they have to turn to avoid hitting a wall ahead. *Ego-syntonic* reasoning is the reasoning that comes from being a person with likes and dislikes and agency. Together they let the language's users "play Turtle", imagining themselves as the turtle and thinking through the actions they would take from that perspective. "'When you walk in a circle you take a little step forward and you turn a little. And you keep doing it.' From this description it is only a small step to a formal Turtle program."[298]. Though Papert's examples emphasize the role of a student carefully planning (or learning to plan) a problem, he also briefly recognized the role of emergence when talking about 'squirals' (the student-invented term for square spirals): "Angles close to 90 produce a surprising emergent phenomenon: The arms of the galaxy like twisted squirals were not actually programmed into the procedure. They come as a shock and often motivate long explorations in which numerical and geometric thinking intertwines with aesthetics." He calls this increase in power, the ability to play with inhuman patience and infinite repetition, "mathpower": "In LOGO, the concept [mathpower] empowers the child, and the child experiences what it is like for mathematics to enable whole cultures to do what no one could do before." In *Logo*, there is pleasure in the focused patience required to draw a desired outcome like a stick figure or a house, but there is also a chaotic pleasure. One retrospective of *Logo*[332] caught my eye with its language:

> "While I sat at my desk one day, two of my classmates figured out how to overwrite the entire screen, which seemed kinda naughty at the time. They giggled, did it again, then giggled some more. From curious children, hackers were born."

What is "naughtiness" in programming? Why does this feel like a delicious transgression for the author? Reading Papert's open-hearted earnest belief in

Logo's ability to lift children above the crushing boredom and wrongness of contemporary mathematical education, it strikes me that his solution still privileges control and patience and thoroughness in programs. Students are compared against each other, and the ones who can look at a program and dispassionately break it into component parts are valorized. These are the skills he wanted the system to teach, surely, and the skills that it needed to teach to be included in an educational setting. But *Logo* also had this emergent power: the idea that screwing up your program wouldn't lead to a blank screen, but something weird and surprising and generative, meant that failure was itself a pleasure.

This interest in generativity can lead to powerful motivation in students, as Papert found with one student who began working on what *sounds* like a rather Tracery-like templated generative poetry maker.

> "But now, as she tried to get the computer to generate poetry, something remarkable happened. She found herself classifying words into categories, not because she had been told she had to but because she needed to. In order to "teach" her computer to make strings of words that would look like English, she had to "teach" it to choose words of an appropriate class. What she learned about grammar from this experience with a machine was anything but mechanical or routine. Her learning was deep and meaningful. Jenny did more than learn definitions for particular grammatical classes. She understood the general idea that words (like things) can be placed in different groups or sets, and that doing so could work for her. She not only "understood" grammar, she changed her relationship to it. It was "hers""

I've heard similar things from Tracery users and teachers, anecdotally. The generativity (and entertaining failure) encourages students to "try it and see", rather than to plan out their approach in advance. This leads to the discovery of the intrinsic hurdles in any problem, such as supplying words that grammatically fill in a template, and motivates the eventual turn to a structured solution to solve it (i.e. lists of verbs and nouns).

Logo was entered, like most programs, as a string of characters typed into a blank page. The *Logo* IDE provided a way to take user input (the code, plus a "run" and "stop running" command) and an engine to interpret, run and display the code. Its coding assistance was limited to replying "I DON'T KNOW HOW TO [FOO]" for commands it didn't know. Even though this seems like not much of a Casual Creator IDE, it has a few properties that we can recognize from our casual creator definition. There is no way to have multiple LOGO files combined into one program, or to import code. There aren't different ways to run a program: a single action interprets, runs and displays the code, and toggles the screen. In this way, this "press GO to run" structure is a power-for-simplicity trade-off which can be seen in more recent systems for non-experts, like Processing[s1802] or Jupyter[s1093].[4]

Interestingly, in 2014 Google released *Code with Anna and Elsa*[s87], an *Hour of Code* exercise of a *Logo*-reminiscent programming environment where Ana and Elsa (of the movie *Frozen*) created ice-skated patterns determined by block-based programs in *Scratch*[s195]. The programmer is led through a series of short exercises (draw a line, draw a circle, draw multiple circles) until they draw a complete snowflake and are placed in a free-play mode. This project is almost certainly inspired by *Logo*. It reinterprets the turtle as an ice-skater, which takes advantage of the same syntonic reasoning of Papert's turtle. Oddly, a princess icon is a 'turtle' that a human should have an even easier time identifying with, but none of the text in the tutorial hints at Papert's "play Turtle" technique of pretending to *be* the turtle or *experience* what the turtle experiences. One other difference is that the language, instead of *Logo*, is *Scratch*, a combination language/platform/UI in which code is embedded in blocks that *snap* together only in ways that work

[2] `https://processing.org/`

[3] `https://jupyter.org/`

[4] Neither Processing nor Jupyter are casual creators. But they share a lot of innovations that ease some issues with traditional terminal programming.

syntactically. We will examine *Scratch*'s interesting syntax-error-free properties later in this chapter.

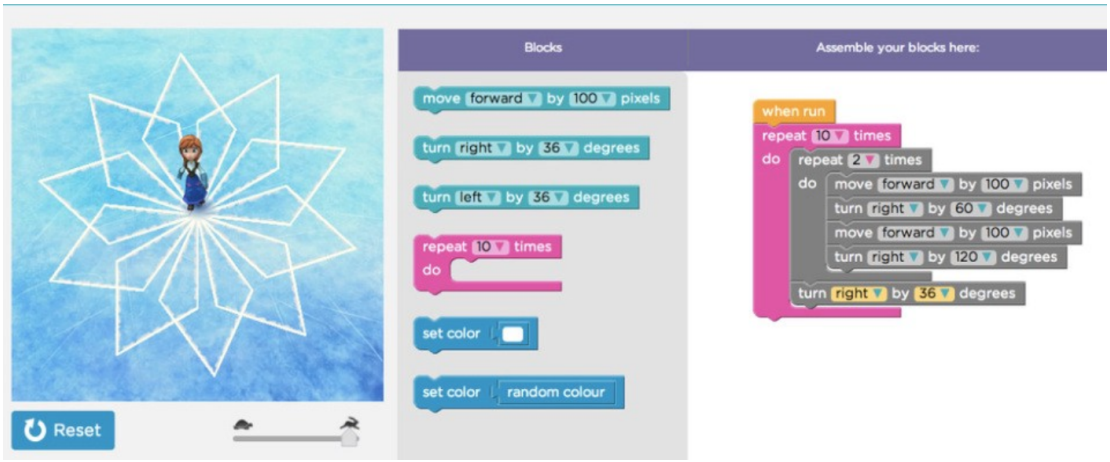


Figure 14.1: *Code with Anna and Elsa*, Google's Frozen-themed *Scratch* programming tutorial inspired by Logo.

Both *Logo* and *Code with Anna and Elsa* draw their paths *over time*: that is, the turtle takes one step per every few milliseconds of actual time, creating an animation of the pattern being "drawn". This approach has some advantages: it allows the user to watch the pattern develop and it shows it in all stages of development (helpful for patterns with lots of overlapping components). One can speed up or slow down the turtle speed in the Google version, though not to instantaneous, and it seems like the LOGO systems Papert demonstrates had only one speed, at least in the versions recorded in video (which would make sense as the machines of the era needed the time to simulate and draw the pixels). What happens if you have a *continuously* redrawing pattern, where the pattern itself is drawn instantly, frame-by-frame, when the program is changed?

Joy.JS

In late 2017[5], Nicky Case released *Joy.JS*[s108], a combination language/platform/UI like *Scratch*. Like *Scratch*, it has a constrained way of inputting code: instead of snapping blocks, each line can do math, initialize variables or perform turtle functions, further constrained by which simulation is running and which part of the program the line is in. Because each line can only have several options, this is a possibility space that can be serialized in a highly compressed way, allowing the entire program to be stored in a link, and then dynamically re-expanded when the link is accessed.[6]

Unlike *Logo* or *Code with Anna and Elsa*, this works *continuously* and *instantly* redraws the pattern. Why should this matter? Recall that the grok loop's speed determines how quickly the user can explore the space, and iteratively improve their cognitive model of the system. Each time a variable is changed in the Google Hour of Code example, the user must wait somewhere between a fraction of a second and several seconds (depending on the drawing's complexity) for the drawing to complete or even get to the step that the user is interested in debugging.

In *Joy.JS*, the user's drawing renders instantly. This immediacy enable each numerical variable (loop count, turn angle, etc) to have a slider attached. When slid, the canvas re-renders the scene with the variable set to each intermediate value. Because this renders so quickly, the user experiences it as an animation, and is encouraged to scrub the slider back and forth to experiment with different outcomes. When I played with Joy.js to make the above illustration, I found myself not actually simulating the code in my head. Instead, I trusted the output of the system and moved through it so fast that I didn't have time, or the need, to

[5] `https://twitter.com/ncasenmare/status/939183550519304192`

[6] A link to a drawing made in *Joy.JS* can be expressed as source code embedded in the link itself.

stop and understand why some numbers caused the loops to overwrite each other or move off-screen. Instead, I would just keep moving along till I got to a place I liked better. As the website claims "No need to second-guess the program, to 'play computer' in your head".[7]

If we look at these three systems (*Logo*, Google's Hour of Code and Joy.JS) though a casual creator lens, we see noticeable differences. All three systems are designed to exchange control for power, enabling though the generativity of the turtle-graphics approach, and so all are *more casual creator* than a language like Python. Hour of Code and *Logo* both have an explicitly educational focus, with Hour of Code executing a set of lessons and tests that explicitly train the user in concepts like loops. They want the user to *understand* what is happening, and be able to logically strategize about how to construct solutions. They want the user to develop an accurate mental model of code.

Joy.JS does not have these educational commitments, so it can encourage the user (through design but also by language use) to play freely and surf the possibility space. I had not anticipated this, but in Joy.JS, the user can build a model of the possibility space *without necessarily building an understanding of the code itself*. I found myself adding lines of code and sliding the numbers around **without reading the code**. The code itself disappeared from my attention as I played, and I saw the "code" as an abstract UI panel that I was toggling and sliding.

14.1.2 Scratch

Scratch[s195] is an online socially-focused IDE (Integrated development environment) and visual programming language targeted at children learning to code,

[7] `https://ncase.me/joy/`

developed at the MIT Media Lab's Lifelong Kindergarten Group, led by Mitch Resnick. As of July 8th 2018, *Scratch* has 37,021,781 projects, 29,225,420 registered users, 161,661,426 comments, 4,656,677 studios (multi-user collaborative rooms) [353], making it one of the largest collaborative coding environments. In comparison, Github [159] in 2017 had 67 million total repositories, 24 million users, and 1.5 million collaborative "organizations".

Like the environments discussed above, *Scratch* prioritizes its users' ability to understand their program and see its results. Like *Logo*, this was done explicitly for the mission of education: "To encourage such self-directed learning, the *Scratch* programming environment was designed to invite scripting, provide immediate feedback for script execution, and make execution and data visible" [256].

Compared to Papert's writings on *Logo*, Resnick expresses a much stronger focus on safety and its necessity in encouraging exploration. Papert's Mindstorms never addresses errors in syntax or syntax difficulties, all of the "errors" listed are errors in planning and logic, as though the children he worked with simply sat down and immediately started writing flawlessly compiling *Logo* syntax. We can guess that this was not the case: when learning any language, we have to learn what the keywords are and how to put them together in a way that the computer will understand. Having taught a bit of early programming, I know that there are also logical jumps to be made in understanding that computers *can* only understand certain syntax. Some students in my past seemed affronted that a computer couldn't understand what they *meant*, just because they reordered commands or left off a semicolon, errors that a human could easily ignore. Worse, the failure of a student to immediately remember and master a language's syntax (coupled with the intimidating screenful of errors that pile up from a mere typo) will often convince a student that they are "bad at programming", even if they grasp the logic

and structure of code. These two issues, **remembering available keywords** and **avoiding syntax errors** are addressed by Scratch's tightly-constrained and visual syntax. In Resnick's introduction to the free Scratch manual, he enumerates the properties of the language and environment that make it less intimidating than a conventional language in a text editor:

> "Scratch is easier to use than most other programming languages for a number of reasons:
>
> - You don't have to remember or type any commands: they're all on screen, so you can just drag and drop them.
> - Commands fit together like jigsaw pieces, so there are strong visual hints about how you can combine them.
> - Error messages are rare. Because Scratch commands lock together, programs always make some kind of sense. It is possible to still write programs with logical errors in, if they don't do what you expected, but Scratch guides you to write things that work, rather than nagging you when they don't.
> - The commands are color-coded and categorized, so you can easily find a command when you need it.
> - The commands in Scratch simplify common activities in games, such as testing whether a missile has hit an alien (collision detection), or rotating a character on screen.
>
> In short, Scratch is designed for your success." [266]

"Scratch is designed for your success" is a notable phrase! They want the user *not to fail*, or at least, not to experience the hard failures of syntax problems and "nagging" error codes. These bullet points address a different kind of failure than Papert's logical failure. "It is possible to still write programs with logical errors in, if they don't do what you expected" handwaves this issue; it is possible (and common) for a program that "makes sense" syntactically to make no sense logically. But to avoid syntax errors and make programs which are readable to new users, Scratch engages with the tradition of visual programming.

Since quite early in their invention (depending on where you want to put the pin), nearly all programming languages are represented as linear streams of characters. Also quite early in the history of programming languages, people were dissatisfied with only linear textual representations of programs and began working on ways to make two-dimensional (or more) *visual programming languages*. These began with early experiments such as the first one by William Sutherland in 1966[280], and 20 years later, Ben Shneiderman expressed hope that many linear-programming needs could be met with more visual solutions[365] and continuing through currently-used programs like Max/MSP/Jitter[237]. Myers defines a visual programming language as a language that is non-linear and actually makes use of its multi-dimensional layout, contrasting it with "programming visualization" which visually shows information about the program state. Intriguingly, one of Myer's contemporaries argues that a physical simulation, Pinball Construction Set [s171][132] constitutes a visual programming language:

> ".. and in fact it is one example of a purely visual language; complete programs are constructed without the use of any conventional code. Of course that facility is achieved because its power is severely limited in that it can only be used to create one kind of computer program - one which plays pinball games."

Is Scratch a visual programming language? It is interpreted losslessly as a linear textual representation, more so than data-flow visual programming languages like Max/MSP/Jitter. However, its shape-based visual style creates affordances, like the interconnecting puzzle-piece nubbins on the top and bottoms of the syntax pieces, that visually communicate to the user which pieces can be connected together. A looming syntax error of holding one piece over a piece it won't syntactically mesh with is clear, as the two pieces would not *physically mesh*. The physical aspect is a UI fiction that communicates the *actual* constraints of the syntax.

14.1.3 Casual languages and the Gulfs of Execution and Evaluation

These systems, *Logo*, *Scratch*, *Joy.JS* and Google's LOGO/Scratch exercise all demonstrate a variety of casual creator principles. As we have seen before, casual creators often benefit from a fast grok loop. The programmer can examine their program, hypothesize about possible actions, type or modify the program, run the program, and see the results. Remember the two "gulfs" of our loop, the gulf of execution and the gulf of evaluation. What solutions do these systems use to bridge those gaps, and tighten their loop? How long does it take to enter in the code, how long does it take to evaluate the code, and what assistance does the system offer the user for both tasks?

Immediate feedback systems like *Joy.JS* allow this loop to run continuously, but they require an input form which is continuous (like a slider) and an output which is both continuously updated *and* continuously visible. This continuity works for a system that can be immediately visually evaluated (like a graphic design) but not for an interactive system (like Scratch games)(chapter 16 shows some possible playful options for these systems). The gulf of evaluation can be either wide or narrow, depending on how fast the program can (or does) compile and run, and how quickly and completely the user can evaluate the outcome. This may be why so many of these casual systems use a drawing metaphor, as we have seen that drawing has a fast evaluation step.

On the other side of the loop, the type of input available to the programmer will affect the gulf of execution. Sliders are also a limited interface for changing code: they work for continuous values, or even two-dimensional values (like a 2D color picker). With a slider, a user can fully model the space of input, from the slider's minimum to the maximum. Additional annotation, like the pixelated

colors of a color picker, will augment the user's ability to model the space. These forms of input are well-suited to casual creators, but the downside is that they may not suffice for other kinds of code, or control over the code. Coding environments with this form of input have very narrow gulfs, but a limited possibility space.

Other input forms have different execution properties. Dropdown menus provide list of *acceptable* inputs, allowing the user to choose among them. Because the choices are discrete, it should be possible to provide assistive visualizations for each option's outcome, though I have never seen this done in practice in a coding environment. Scratch has a menu of code blocks, which can be selected, then placed anywhere that the block fits, as limited syntactically and visually.

The most common and familiar way to enter code is a text field. Text fields allow the user to type *any* code, within a predefined character set. An example of a system with a *single* text field can be seen in coding in a traditional text editor, but systems can also have text fields embedded in more structured input UIs. These inputs offer the flexibility to enter any string. However, a blank text field tells the user nothing about what *should* go there, or what valid syntax should be. In many modern IDEs, autocomplete features will add "visualizations" (in code or in a floating box) of what code *maybe should* appear next. As seen earlier in the dissertation, even professional IDEs like Sublime Text [s223] show casual creator patterns. Expert developers and novices alike benefit from text-completion that creates syntax faster and with fewer errors than hand-typing it. The difference is that a system for experts also needs allow the user to discard the suggestions, while a casual creator system might be implemented to constraint the user to only "known-good" code. Similar shared patterns include syntax highlighting, auto-formatting, and continuous linting (checking code for errors). Both casual and expert users may appreciate the additional power, but expert users **also** need the

control to ignore or disable these safety features.

In summary, programming is a task that has an unusually difficult gulf of execution and gulf of evaluation. Casual creator IDEs can bridge these gulfs with assistive or UI-scaffolded input, and with automated evaluation, syntax annotation, or linting. Are there further casual creator properties of languages themselves? In *Logo* and *Joy.JS*, we see that a language with a graphical output will produce output which can be quickly generated by the machine, and quickly evaluated by the human user. From *Scratch*, we see a language with a set of syntax that is so small and so rigidly structured that one can replicate its constraints as a set of blocks. In the next section, I propose some overarching properties of casual creator languages and IDEs, drawn from casual creator languages like *Logo* and *Scratch*. I also draw from my experience making Tracery (an unexpectedly successful grammar language/library) and my experimentation trying to apply the lessons of Tracery to both Chancery (a finite state machine language/library) and a revised and expanded new version of Tracery.

14.1.4 Features of a casual creator language

From this set of examples, and what we have so far discussed about casual creators, can we encapsulate some principles for Casual Creator languages?

I propose three overall principles for such languages, and patterns for them: **scaffolded**, **unfoldable**, and **little**. Not all casual languages will have each principle, but each principle unlocks new affordances which support a casual creator experience. As usual, these patterns are targeted not only at language for casual users but also languages that want to enable casual creator modes within a professional practice.

Scaffolded

Let us consider a common language/IDE experience:

- the user imagines some code (what it would do, and how to encode it)
- the user enters that code in a text editor
- the code is compiled/interpreted and run
- the program produces and displays output
- user interprets the output, and revises their mental model of their code
- the user revises the code

Remember the grokloop from chapter chapter 8? The act of writing code is *also* modelable as a grokloop in which we iteratively explore the possibility space of our programming language though modifying an artifact (our code). As we have explored in the previous section, IDEs (casual and non) can provide assistance, constraint, or visualizations at any moment along the loop. Such assistance can come in two forms: **augmentation** and **constraint**.

Augmentation is any automation that *produces* something in response to a user's action, in order to ease one of the following steps from the groklook:

- make a hypothesis
- plan the encoding
- encode the change
- display the artifact
- understand the display

- evaluate the artifact
- update the mental model

Displaying the artifact, ie, the output of the code, is one form of augmentation. As we have seen with *Logo* and *Joy.JS*, there are variations within that, such as updating the drawing instantaneously or rerendering it as an animation over time.

Two other steps from the diagram ("modify the model" and "reinterpret the model into an artifact") are usually computation-only steps, and wouldn't make use of additional augmentation to help the user.

Unfoldable

Unfoldability is a common pattern across many casual creators. We have already seen casual creators that have a limited "beginner" mode, or artificially restrict the tools available until the user has "earned" themchapter 10. Unfoldable language have syntax which can be used in its fully-unpacked advanced form, but novice users can use a simplified form, and seamlessly transition to the full form as they advance.

HTML has been a surprisingly effective novice language, in large part because of its very lax approach to syntax[8]. Most badly written forms of HTML will still render in a browser: in practice it has perhaps the most forgiving syntax of any language (despite what the official W3C specifications say). HTML also has the useful property that **plain text** is a valid form of programming it. This creates a smooth escalation of expertise that meets non-programmers where they start. A common path of learning HTML goes something like this:

```
This is my webpage
```

[8]I didn't know that you could close a "<p>" tag for a full 20 years of using HTML.

```
This is <b>my</b> webpage
```

```
This is <b>my</b> <a href='webpage'>webpage</a>
```

```
This is <b>my</b> <a style='color=blue;' href='link'>webpage</a>
```

```
This is <span id='me_1' class='emph'>my</span>
```

```
<a class='big_link' href='webpage'>webpage</a>
```

At the first stage, they have created valid HTML by writing in plain text. At the second, they have heard about tags, and (after a lot of fully-bolded webpages) have learned to close them. In the third, they may have pasted in a link from an example, but can see that it is *more complex* than the bold tag, and has an equal sign and some quotation marks (which are *sometimes necessary* but they can't figure out why). In the fourth, they have discovered that these internal properties can do more than make links, and they change the color (likely by pasting in someone else's cryptic "color=blue;font-size=150

"Syntactic sugar" can also be a part of unfoldability. Sugared syntax can be language features that act as shorthand for common actions[259] or increase readability[214] Sugaring is by definition a mathematically unnecessary part of the language:

> "A construct in a language is called "syntactic sugar" if it can be removed from the language without any effect on what the language can do: functionality and expressive power will remain the same."[3]

For a user who has not fully unfolded the language, using sugared syntax

can provide restrictions that keep them within the possibility space. In a well-sugared language, it should be easier and faster to write safe, common code, while dangerous and weird code can still be written with the full syntax.

Little

> "When I first started to design a survey system to be implemented on a personal computer, I sketched an interactive program. It sounded easy at first: tell me the question, tell me the responses, now to the next question. As I explored further, though, I realized that I was designing large portions of a text editor (I want to change part of question 35. Which part? A response. Which response? 3, I think, but let me see them all. Oops, 4. Change "Smith" to "Smythe", and leave the rest alone...). I finally made progress by abandoning the interactive approach and thinking about the problem as designing a little language to describe surveys (and leaving the editing to the system text editor!)."[30]

Jon Louis Bentley proposed the term "little languages" for simple domain-specific languages that can serve as input for larger systems. A canonical example of a little language is the syntax for Regular Expressions. The domain of what it tries to express (a restrictive kind of pattern-matching) is limited enough that the syntax can be very small yet expressive, while also being broad enough to solve a very common set of needs. Many functions accept regular expressions, and many non-programming tools can also take it as input. It is a rare lingua franca in computer science, a language that shows up *across* language communities (a PHP regular expression question on StackOverflow can be useful to a Python developer years later).

Bentley describes six principles for language design in "good taste":

- "Generality: use an operation for many purposes.
- Parsimony: delete unneeded operations.
- Completeness: can the language describe all objects of interest?
- Similarity: make the language as suggestive as possible.

- Extensibility: make sure the language can grow.
- Openness: let the user "escape" to use related tools."

The last two points have lessons for casual creator languages. While I won't attempt to argue that Regex is a casual creator[9], its spread and usability showcase why "little" languages can catch on in ways that "big" languages can't.

In my zine "Oops, I made an open-source art tool!", I talk about what open-source tools owe to the people who make art with them. One of the strongest principles is the need to not only have code be open-source, but have the data format *open* as well, so that artists have the ability to *leave* with their art, and find another tool that will take it as input. If a data format/language is human-readable, then it can travel on all the channels that normal human communication travels along: text editors, comment sections, tweets, or email can all be used to share (and edit!) human-readable formats.

Additionally using an open human-readable file format for user artworks (like JSON for Tracery grammars) can keep files readable for years to come, even if the main readers go down. If the version of the tool used to interpret/run the art is included *in the source* of the art, this can help future curators revive and emulate the art in the future.

If the format is common, like JSON or XML, the programs/artifacts can also be imported into tools meant for that kind of data. There are no end of automated linters and error-checkers for JSON, as well as JavaScript libraries that can turn any div on a webpage into a fully-featured structure editor, where users can *only* enter valid syntax[119]. Tools like these can provide casual creator safety as a plug-in.

10

[9]..but could you *make* a casual creator for regular expressions?

[10]In the Sims, all object behavior was stored in LUA scripts. This relatively simple human-

Little languages can be the input of one system, but they are also small enough that they can be the output of another. Not many systems can generate useful Javascript: its possibility space is too complex. But it *is* possible to generate useful Regex[165] and Tracery[s32].

Tracery has become a successful language because it is a **little language**, with **unfoldable syntax**, and with **scaffolding**. In the next chapter, I will take a deep-dive on Tracery, its offshoot Chancery, and their complicated status as a casual creators.

readable format allowed non-technical designers to design behavior during development. It *also* made it so that community members could easily mod objects, sparking a huge and long-lived modder community.

Chapter 15

Tracery and Chancery

In this chapter, I will review the history, design, and creative ecosystem of *Tracery* for authoring recursive text expansions. In Tracery, users author "grammars", consisting of several symbols, each of which is a pairing of a key, and some number of rules. Below is an example of a simple grammar in Tracery syntax in JSON. If the symbol "story" is expanded, we might get "Betty the cat went to grad school".

```
{
"animal": ["cat", "lemur", "okapi"],
"name": ["Ahmed", "Betty", "C.J."],
"character": ["#name# the #animal#"],
"place": ["the zoo", "the beach", "grad school"],
"story": ["#character# went to #place#"]
}
```

Tracery is an example of a "little" language (see the previous chapter) that exists as part of a larger ecosystem of creative work (tools, tutorials, ports, and experiments), some of which I built, but a larger part of which was built by others,

as we will see later in this chapter.

Tracery has become one of the more popular generative tool for casual users. The library and language power at least 7000 twitterbots through the hosting platform *CheapBotsDoneQuick*[s31][42], and countless other generative projects, bots, and games. It has also spawned a dozen ports into many different languages, and several spin-off languages. But this "little" language first started as a class assignment. In Michael Mateas's CMPS248 "Interactive Narrative" class, we were assigned to write a pencil-and-paper story grammar. If it were my first time taking this class, I would have just done that, but I had also taken a version of this class in 2005 at Georgia Tech, and the assignment was the only thing that had not changed in the intervening decade of story-game evolution. I has also recently started learning Javascript, so I instead wrote it as an interactive HTML page[1].[66] Javascript's *object* data structure (a dynamic dictionary) practically *wanted* to represent the lookup table of a replacement grammar. As many Tracery implementers since then have discovered, once there is a dynamic dictionary that pairs strings with arrays of strings, then the only additional requirement (to create a base-level Tracery) is to do a bit of light string-parsing to extract the tags.

It is hard to say what makes a language "usable" or easy to learn. Dagstuhl hosted the *Evidence About Programmers for Programming Language Design* retreat, dedicated to identify the nature of language learnability. The results were complex, pointing to the inability to do clearcut evaluations (it takes years to master a language).

> "One idea that emerged is that learnability appears to concern the complexity of abstractions embedded in a language. For example, consider indefinite loops, which require someone to reason about poten-

[1] In the sourcecode original 2012 version, you can still see the influence of traditional planning-based story-generation language, as one part of the expansion is referred to as a "plan".

> tially infinite executions of a block of code. What about this possibility of the infinite is hard and why is it hard? We need methods to discover these hard concepts in languages, so that we can precisely define what is irreducibly difficult about them." "The methods we might use to evaluate language features are many, but the group had a large bias towards positivist methods like randomized controlled trials." "Language designers often make vague arguments about the "simplicity" or "intuitiveness" of language features, without ever saying what they mean by these terms, or testing whether these claims are true."[223]

Tracery has always been a bit of an awkward fit as a "casual creator". It was an intentional rejection of the approach of many of my labmates (and advisor's) conditions-and-properties planning-based approaches.[375][262]. These systems could produce viable narrative plots and motivations, but the labor involved in building up all of the constraints was grueling, and suppressed my ability to be playfully creative. Tracery's name comes from it intention to *not* model narrative causality, but only provide texture and expansion to an existing structure (which depended on the author to write it with causality implicit in the structure): Tracery is the fragile complex bits that provide *interest* and *style* to a Gothic cathedral, but *they do not hold up the cathedral.* That lack of logical dependencies made it possible for Tracery to grow from what I expected to be a "story-generation tool" (with no story-generation assumptions baked in) to something that is only rarely used to make full stories, but creates cocktail recipes, names, insults, emoji art, code, music, legal documents, and all of the strangeness that users can find ways to make. The rest of this chapter will cover the ways in which Tracery is, and *isn't* a casual creator, but it is worth documenting its path, so here is a history of notable events which track its growth, both in users and possibility space.

15.1 Tracery Timeline

2012 and **2013**

- *Feb 15, 2012* Michael Mateas's interactive narrative class assigns a pencil-and-paper assignement, which I do in Javascript. My Roman comedy generator is still online[2].

- *Dec 13, 2013:* A challenge from a friend leads me to update the code and publish the first general purpose Tracery (since lost)

2014

- *Jan 16:* Tracery is play-tested with Noah Wardrip Fruin's Playable Media students (including first-users Jacob Garbe and Aaron Reed)

- *June 17:* I present a poster at the Seventh Intelligent Narrative Technologies Workshop, and release an interactive editor[3][80]

- *Nov 17:* I upload *Eternal Nightvale*, a Tracery fanfic generator[4], to Itch.io for the first ProcJam

- *Nov 24:* Tracery is uploaded to Github[5]. It is practically open-sourced, but doesn't get a license until George Buckenham bugs me for it years later.

2015

- *Jan 19:* Dietrich Squinkifer releases *Interruption Junction*[s103], the first game[6] using Tracery

[2] `https://jsfiddle.net/galaxykate/SECmW/`
[3] `http://brightspiral.com/tracery`
[4] `https://galaxykate.itch.io/eternal-night-vale`
[5] `https://github.com/galaxykate/tracery`
[6] `https://twitter.com/DomPamplemousse/status/557281499787960320`

- *May 13:* I run an IRB evaluation of Tracery in the Interactive Narrative class
- *May 28:* Matthew Balousek releases Twinecery[7], a wrapper to use Tracery in Twine games.
- *May 21:* I lead a Tracery tutorial at UCSC's WritLarge
- *May 27:* George Buckenham announces[8] CheapBotsDoneQuick (CBDQ), a tool made for a Feral Vector workshop, after finding Tracery through *Eternal Nightvale*
- *May 30:* Tracery writes valid Javascript code[9], executed throuh 'eval'
- *May 31:* Ranjit Bhatnagar posts[10] the first SVG art-making grammar[11] with the Tracery editor (not CBDQ)
- *June 18:* Aaron Reed leads a CBDQ bot-making workshop at DHSI2015 (@BotCarrion emerges here)
- *June 22:* Tracery demonstration at FDG (Asilomar)
- *Oct 19:* I make a music generator[12] with Tracery and ABC notation
- *Dec 3:* Tracery (with visualization work by Ben Kybartas) is presented at ICIDS 2015 (Copenhagen)[82]

2016

[7] `https://twitter.com/mrfb/status/604147075043852288`
[8] `https://twitter.com/v21/status/603553833516687360`
[9] `https://twitter.com/GalaxyKate/status/604680265223409665`
[10] `https://twitter.com/ranjit/status/605149881200713728`
[11] `https://gist.github.com/moonmilk/2f11d31f4032c35bbb44`
[12] `https://twitter.com/GalaxyKate/status/656250630277758976`

- *Jan 3:* CheapBotsDoneQuick enables SVG images, the first CheapBots image bot is released
- *Jan 10:* Ranjit Bhatnagar's first SVG bot [13]
- *Jan 18:* Tracery is on the front page of BoingBoing (with a picture of Interruption Junction)
- *Jan 18:* DnD Twitter account (@Wizards_DnD) creates procedural fortunes, by Darius Kazemi and Courtney Stanton (with George Buckenham's Node Tracery)[14]
- *Jan 29:* Tracery zines are printed for the UCSC Global Game Jam kickoff
- *Mar 9:* Allison Parrish releases the first Python port of Tracery
- *Mar 12:* Tracery.io is online
- *Mar 14:* I present Tracery at the GDC Indie Tech Toolbox[71]
- *Mar 28:* I write a Tracery grammar that writes Tracery grammars (Javascript-generated Tracery grammars had happend much earlier)[15]
- *July 19:* Tracery workshop and Techne^{s229} presentation at Nucl.AI (Vienna)
- *Oct 21:* Presented Tracery to Workshop in the Woods (a gathering of generative artists and musicians)
- *Nov 15:* Chris Martens and James Ryan present their own languages at the INT workshop at ICIDs, with ideas that inspire me to start on the syntax of Tracery 2

[13] `https://twitter.com/ranjit/status/686214985266163712`
[14] `https://twitter.com/tinysubversions/status/689143845259378688`
[15] `https://twitter.com/GalaxyKate/status/714530128018034688`

2017

- *April 2016-Feb 2017:* I work building conversational prototypes for Google's "Doodles & Delight" team, which is working on the Google Assistant. Some of this work becomes Chancery.
- *Apr 28:* "Bot poetics" presented at Interrupt 4
- *Oct 30:* Bottery/Chancery is open-sourced, after much back and forth with Google's open-source group(and internal Google work by Calvin Ashmore). It briefly goes to #1 on *Hacker News*.
- *Dec 3:* Tracery and Chancery languages are presented at the Google games summit

2018

- *June 3:* I attend the "Convening of Open-Source Tools for the Arts", as a representative of the Tracery language.
- *Jun 6:* Twitter suspends CheapBotsDoneQuick (it is brought back several days later)
- *Aug 20:* Joel McCoy (@BooDooPerson) clones CheapBotsDoneQuick as CheapBotsTootSweet.com to make bots on Mastodon
- *Sept 6:* First Tracery Quine (a program that can write itself)[16], by @EliDupree
- *Oct 25:* New Tracery site with Tracery2, in prerelease ([69])
- *Oct 26:* Puzzle Poesis Tracery tutorial (to assembled comparative literature scholars)

[16] `https://twitter.com/EliDupree/status/1037811562915024896`

- *Oct 30:* Tracery/Chancery are selected for funding by UCSC's Center for Research in Open-source Software

In Tracery's first year, it was used only locally to friends and colleagues at UCSC. 2015 was the year that Tracery took off an gained recognition and use outside of UCSC, with George Buckenham's CheapBotsDoneQuick. Writing CBDQ bots gave new users a way to create work that could quickly go spread on Twitter, and when it did, the new audiences also became aware of Tracery and CBDQ.

15.2 Tracery language features

Tracery is a recursive text expansion language. Each Tracery programs is a "grammar", which is a list of "symbols", and each symbol is a key and the set of possible rules (a "ruleset") that it can expand into. When a key is "expanded", one of its rules is randomly selected, and that rule is broken apart into plain-text sections and sections that need to be recursively expanded, like tags that contain the key of another symbol. These special sections are expanded out recursively, and when finished, they are concatenated together to create the finished text of that symbol's expansion.

This process provides a generative pipeline that turns a grammar and a starting symbol into a space of possible finished text strings. Depending on how the grammar is authored, this space can contain many surprising artifacts. Running the generator over and over can be a pleasurable way to discover new artifacts and explore the space, even when done *non-interactively* through a Twitter bot. This pipeline can be used in a variety of different systems (some casual creators, some games or other experimental forms), as we will see in the end of this chapter. Tracery does not contain a world model or other constraints[17] and restricts its

[17]Tracery 2 enables some of world-state tracking.

expressive range more than other story-generators, but in doing so, it allows a simpler authoring format. It does offer a small set of modifiers that are common in English-language authoring (like pluralization or "a/an" prefixes) that would be hard to add correctly using grammar-only logic, which add power without the total control of user-written code.

The data structure of the grammar (usually a dictionary of string/array-of-strings pairs) can be easily implemented in other languages, especially languages that have a built-in dictionary structure like Javascript and JSON, which may have led to the large number of languages it has been ported to.

15.2.1 Domain model

Tracery uses a context-free(ish)[18] grammar to represent a generative possibility space. A context-free grammar, is at its heart, a list of symbols, each with some number of possible ways to expand it. If the symbol cannot be further expanded, it is a "terminal"[222].

The "program" that a Tracery author writes is simply the list of symbols and their expansions. This symbol grammar is the only data that needs to be saved or modified, and comprises the whole of the Tracery grammar[19]. This grammar may be represented as a graph of which symbols have rules that connect to other symbols, and seen as a graph, there may be loops (rules that self-reference) and unreachable symbols. Some grammars may be used to expand only from one symbol ("origin"), or used to expand multiple symbols that share sub-grammars within the grammar (ie: a *last-name* grammar and a *town-name* grammar that both contain references to name components).

[18]Tracery is not actually context-free, due to the push rules, but "context free grammar" is close enough to get people to understand what's going on.

[19]Except in Tracery 2, where it can also contain metadata.

The pleasure in creating such a story grammar does not lie in admiring the dictionary of rules, but in seeing the variety of outputs, so the "artifact" of a Tracery grammar is not the grammar, but the possibility space that it creates.

A dictionary of Symbols

The user authors a dictionary of symbols, each with a unique symbolic name (the *key*). Each symbol also has a non-zero number of possible expansion rules. Unusually for a grammar, each dictionary entry for a symbol is not just an array of possibilities, but a stack of such arrays. This array can be pushed-to or popped-from by actions in replacement rules, allowing the dictionary to change over the course of generation[20]. This feature (along with the Modification rules) means that Tracery is not a strict "context-free" grammar, but the increased authorial affordances make it well worth the loss of strict mathematical formality of a true CFG. We will see uses of this push/pop functionality in the "Expansion examples" section.

Rule syntax

A rule is a way of expanding out some single symbol into either a sequence of tags (containing a symbol's key and optionally, modifiers), text, actions to perform, or some combination of these. In Tracery, a rule is a plain-text string, which is then parsed by the Tracery system as an array of some number of the following elements.

- **Text:** no further expansion is necessary. These are the terminal symbols for Tracery.

[20]In Tracery 2, this changes so the this array editing occurs inside an object representing the context of the current expansion, so that multiple expansions can operate on a single grammar without overwriting each other.

- **Tag:** look up the current options for this key in the dictionary. Pick one, and iteratively expand it until it consists only of a tree of finished texts.[21]

- **Tag with modifiers:** expand out this key's symbol, but then apply some number of modifications to the subtree of texts (almost always a function on the finished text of the subtree) . This is useful for solving common natural language generation issues like capitalization, "a/an" discrepancies, pluralizing or conjugating the returned text.

- **Push/Pop actions:** of the format `[_key_:_ruleset_]` or `[_key_:POP]`.

- **Advanced actions:** A Tracery2 feature, these actions are in braces `[]`, but may contain list comprehensions `['_rule_' for _id_ in _ruleset_]` or functions, i.e.: `[functionID(_ruleset_,_ruleset_)]`.

Modification functions

Tracery provides a fixed set of functions that can transform text trees in ways that might be useful to an author, such as providing capitalization. The casual author is not expected to write these functions themselves, as they are actual code, and could be arbitrarily complex manipulations of the text tree, but they should perform a single, easily comprehended transformation, like capitalization, so that the casual author knows when to call for them.

Expansion examples

Imagine a rule's text representation as this line:

```
"story": ‘‘Once #name# went on an adventure to
```

[21] In Tracery 2, the "key" may also be a ruleset created by an advanced action, such a list comprehension or a function that pulls a list of strings from an external source, or a *path* key, which pulls information from a *world object* in the expansion's *context object*.

```
the #placeAdj.capitalize# #place.capitalize#."
```

Here is how Tracery would parse, and then expand this rule:

- The string is split into an array of text and non-text sections as delimited by the `#` characters: `Once`, `went on an  adventure`, `. Seeing such a`, etc, for the texts, and mainCharacter, placeAdj.capitalize, destination.capitalize, etc, for the symbols.
- The texts are terminal symbols, so are added to the text tree without further modification.
- These symbols are recursively expanded into text trees of their own, and the ones with modifications (`capitalize`) have the text trees modified by those modification functions.
- The finished text tree is viewable as a human-readable text string, but can also be passed back to whatever generated it as an additionally-modifiable text tree (creating recursive structure).

To see how push/pop rules work, imagine the same symbol with this rule instead:

```
[myCharName:#name#][myPlaceAdj:#placeAdj#][myPlace:#place#]

Once #myCharName# went on an adventure to

the #myPlaceAdj.capitalize# #myPlace.capitalize#.

Seeing such a #myPlaceAdj# #myPlace# made #myCharName# #emotion#''
```

Here is how Tracery would parse, and then expand this rule:

- The string is split into three action sections (e.g. `[myPlaceAdj:#placeAdj#]`), and alternating text and tag sections.
- Each section is expanded in order. When the action sections are expanded, they expand their inner rulesets first, becoming the single-rule rulesets "Jamal the Iguana", "scenic" and "wasteland". These actions are *push commands* and so their resulting values are pushed onto the ruleset's stack for "myCharName", "myPlaceAdj", and "myPlace", (which don't exist yet, so they are added to the symbol dictionary).
- The other sections are expanded. Since some of these symbols have been overridden by the push command, they expand out into whatever had been generated by the action sections. The resulting story can refer to the symbol myCharName as many times as it needs to, while being sure that it will always be "Jamal the Iguana" until another push command is used to overwrite that.

No *pop* actions appear in this rule. So after it has finished expanding, the new rulesets are still on those symbols' stacks. If we wanted to create a Scheherezade style story-within-a-story, we could use *pop* actions to remove those overlays to recover the previous (if any values). For example, if we added pop actions to the story rule above, then we could expand this rule:

```
[myCharName:#name#]#myCharName# opened a book.
The book read: #story#. #myCharName# closed the book.
'What a good story', #myCharName# thought.
```

"Cynthia the octopus"(push: myCharName) could read a story about "Jamal the Iguana"(push: myCharName). Then Jamal's name would be popped from

from stack, and Cynthia would close the book. This can be applied to whatever depth is necessary (bounded by the computer's stack limits). In the version of Tracery on CheapBotsDoneQuick, a push rule can also occur *inside* a tag to be auto-popped after expansion.[22]

Push/pop syntax can also be used to push *groups* of values at the same time[23]. In this example, I want to have a few matching groups of words saved, like a set of pronouns, and an occupation and its equipment:

```
{
"setGender":["[they:she][them:her][their:her][theirs:hers]",
"[they:he][them:him][their:his][theirs:his]"],
"setOccupation": ["[occupation:baker][tool:spatula]",
"[occupation:warrior][tool:sword]"],
"story": ["#setGender##setOccupation##they.capitalize# was
#occupation.a#, and carried a large #tool# on #their# belt."]
}
```

This would output "She was a warrior, and carried a large sword on her belt." and "He was a baker, and carried a large spatula on his belt." This allows the user a bit more control over the generation, but without changing the syntax itself. However, there are still parts of the possibility space that are left off here, such as being able to give the user they/them pronouns and have "They *were* a warrior" correctly conjugated. Giving the user more control, but not changing the base level syntax which so many user are comfortable with, is the challenge that I am trying to solve with *Tracery 2*.

[22]This was a bad idea; its added complexity was not justified by its use.

[23]I explained this at length in the Twitter thread here: `https://twitter.com/GalaxyKate/status/1011667909951668224`.

15.2.2 Tracery 2 Syntax

I have been working on the design for Tracery 2 since being inspired by Chris Martens' and James Ryan's language syntax tutorials at ICIDS 2016[260]. Tracery 2 makes several jumps forward from the original Tracery(s)[24].

- *Rulesets* as a first class concept (many parts of syntax now take or produce *rulesets*)
- Rulesets specified in the grammar can be one string, an array of strings, or complex structures with case-matching, conditionals, custom weightings and distributions, repeat prevention or randomization paradigms (shuffled cards vs dice roll) specified
- Simple string or string array rulesets can also be created on the fly during expansion with *rule generators*, such as list comprehensions "`#story#` for x in animals" (creates an array ruleset by expanding the rule `#story#` with x set to each rule in the current ruleset at the symbol `animals`).
- Expansion tags now take a ruleset instead of just a key (but a key represents the current ruleset at that key's symbol, so it is backwards compatible)
- Expansion is controlled by a *context* object, rather than writing to the grammar itself. This context object contains all the modifiers and functions, an optional link to a world object, and any overlays or other information stored during expansion.
- Most places that use a key to refer to a source of rules (in Tracery 1, always a symbol in the grammar or its overlay) now use an *address*, which

[24] I have rewritten Tracery dozens of times, and the syntax has shifted slightly over its early history.

can be a key in the grammar, *or* a path to a stored ruleset in the world object. Keys can also be *dynamic* and have tag syntax embedded in them: `#weapon{myCharClass}#`.

- Modifiers (and functions) can have parameters
- Expressions can embed mathematical notation inside of parameters or rule generators.

Tracery 2 will add some of the control that my power-users have been requesting[25], while still keeping the Tracery 1 syntax as the "base" experience. This will keep the language itself at a usable level; however the added parsing and expansion complexity will make Tracery 2 less likely to be ported to other languages. Many ports did not support the full modifiers and push/pop functionality of Tracery 1. Part of Tracery's little-language status is based on its ability to exist on multiple platforms, I want to make sure that people know what level of Tracery they want to port. To that end, this chapter now specifies *levels* of Tracery, so that even the process of language-porting can be *unfoldable*.

15.3 Tracery specification levels

Tracery has been long overdue for a specification, so here are the levels of Tracery syntax, for those creating their own versions of Tracery. Tracery 0, for instance, is a popular parsing-and-recursion exercise suitable for beginning programmers.

Tracery 0: Tag and text only. Tags only contain a single *key* to expand. No *actions* or *modifiers*. The grammar is made out of key-ruleset pairs. A *ruleset* can be a single string, or an array of strings.

[25]The most common request is a way to keep rules from repeating.

Tracery 0.5: Tracery 0, but with modifiers. Modifiers are stored in the grammar.

Tracery 1: Adds push/pop actions to Tracery 1.5. This requires that the *state* of the grammar is recorded, with an *overlay* dictionary composed of stacks of rulesets for each symbol. This can be done in the grammar itself[26] or as a separate *context* object that records an expansion.

Tracery 2: Featureful Tracery that is currently in development. Python style list comprehensions can create and modify rulesets, which can then be used in push actions. Actions can contain functions, and modifiers can contain parameters (which are rulesets). *Context* objects record the expansion and any state it needs (shuffled draw-deck rules, push/pop stacks per symbol). Modifiers and functions are stored in the context object, not the grammar. The grammar is an immutable object. Not for the faint-of-heart to port.

15.4 Tracery as a little language

Remember that a casual creator:

- privileges enjoyment of the creative process above productivity
- produces artifacts within a limited-yet-meaningful domain space, enabling automation and support, both passive (encoded into the domain model and system constraints) and active (responding to user actions)
- supports a state of creative flow by restricting choice and preventing hard failures while allowing rapid iteration
- results in the user's feeling of pride and ownership toward the produced artifact, and sense of pride in their own creativity.

As we have seen in the previous chapter, these can't apply to a language itself, but *can* apply to how a language's properties allow systems to be built around

[26] In the 2016 branch of Tracery and earlier.

it (IDEs, visualizations, sharing tools) which *are* casual creators. Casual creator language environments (the language plus its surrounding support structures) have three properties, **scaffolded**, **unfoldable**, and **little** (see chapter 14. Let's examine how these properties reflect Tracery's programming ecosystem.

15.4.1 Scaffolded

In the previous chapter, I proposed that two categories of language scaffolding, **augmentation** and **constraint** can help guide the user to writing good programs. Tracery is often written in an augmented environment. In my own original IDE for it[], typing anything in the JSON editor causes a number of behind the scenes processes to happen:

- the system attempts to parse the JSON, and if it fails, borders the screen in red so the user can fix the error
- if the JSON parses, it outputs some number of expansions in the right-hand panel
- if Color Mode is on, it colorizes the symbols and rules in the grammar, displays the expansion as a nested tree of HTML elements which are each colored according to the symbol that they came from. I have made a number of other Tracery editors[70][69], as have others[176], and the common factor is *always* the creation of multiple expansions, and the ability to rapidly reroll them. Tracery is a language of authoring *a possibility space*, so for users, the primary way to explore that space is to see a number of examples of it.

Tracery is also a very *constrained* language. To restate a casual-creator theme, Tracery *knows* what the user is trying to do (expand text) and so even if there are errors, such as referring to a symbol that has not been defined, it can continue

the expansion *and* be helpful about it. When this happens, instead of halting the program and dropping an error (like we would expect from professional programming), it pastes in the key in double brackets, (`[[animal]]`). This formatting lets the user notice this omission, but also lets them *ignore it* until they are ready to fix it. Casual creator tools and languages should adhere to my proposed **Doctrine of Partial Specification**[27]:

> Creation is by its nature a piecemeal and non-linear task. Any creativity tool must accept and support **partial**, **incomplete**, or even **incorrect** input to the best of its ability, until such time as the creator chooses to fill in any holes.

We recognize that creative flow is a phenomena in writing, and if we made traditional authors complete each sentence before moving on to the next one (with no "dialog here", "Todo:motivation scene" annotations), no-one would ever finish any novels. Programming is no exception, especially creative programming. So this feature enables Tracery authors to *stub in* symbols that they will define later, and encourages them to name them in ways that flow naturally with the language of the sentence, i.e. `#personName# went to the #place# and did #aFunThingToDo#`.

This constraint also enables a variety of syntax decomposition techniques (seen on the bottom right of Figure 15.2). It also allowed me to make a visual *structure editor*. This visual mode allows users to click to create new symbols and rules. While I had *not* expected this to be of any importance (I'd added it to justify adding Tracery to a Casual Creator dissertation), it became a critical feature. Many users (including CBDQ's most prolific user, Nora Reed) did not identify as coders, and felt uncomfortable and unwelcome in the JSON editor. I had thought that my inclusion of the structure editor was a perversion of what casual creatorness *ought to be*, but the **constraint** of making JSON errors impossible was a necessary safety support for a large percentage of my users.

[27] Inspired by the Doctrine of Partial Performance in law.

15.4.2 Unfoldable

Tracery has been very successful in is ability to *unfold* as a language through use. Like HTML, it consists of **little chunks of syntax embedded in natural language**, at least, once you are looking at a single rule (which is inside of quoted strings in JSON, which still presents a hurdle). This allows users to begin authoring in natural language. The use of hashtags has also been beneficial, since users can leverage expectations of hashtags in systems like Twitter and Instagram. A hashtag on Twitter is a special section with only one word in it (Twitter has trained people to not expect special characters or spaces in hashtags) that the computer will do something automated with, and this is the same in Tracery. Users don't seem to mind the small addition of a second `#` to close the tag.

Most users stop at this point. Others then go towards deeper recursions, self-referencing symbols, and use of the modifiers. Relatively few users ever use the push/pop actions. From anecdotal experience, push/pop is the hardest concept to grasp for users. It is *like* a variable (for users with programming experience), and rule-selection can be used to set multiple "variables" at once[28], but this is non-intuitive and leads to some bafflingly unnecessary workarounds, even for subject experts[424].

In the syntax for Tracery 2, I have attempted to maintain this property, by keeping almost all of the original syntax, but allowing it to *unfold* at places where that syntax had previously made assumptions. For example, the key in tags (`#animal#`) previously meant "go to the symbol `animal` and select a rule from the ruleset there". Now, there can be several different ways to create rulesets, which can be used in tag selectors `#[ruleset]#`, push actions `[myValue:[ruleset]]`, or functions `[join("", [ruleset])]`:

[28] `https://twitter.com/GalaxyKate/status/1011667909951668224`

```
[animal]
[animal\\{biome.capitalize\\}]
[/myPlayer/pet/animal]
['#animal# hat' for x in animal where rhymesWith(x, 'at')]
```

The first one simply returns the current ruleset for the symbol "animal". The second one expands the part in curly brackets as if it were a tag. Thus this it returns the current ruleset for "animalWetlands" or "animalDesert" depending on what "biome" generates. The third ruleset requires a *world object* in the expansion context. The world object can be a dictionary of dictionaries, or anything that implements the methods "getAtPath" and "setAtPath". This syntax gets the data at that path location, which is hopefully a string or array of strings, and attempts to construct a ruleset from it.

The final syntax is a very fancy list comprehension, with syntax borrowed from Python, which:

- gets the current ruleset at 'animal'
- filters that ruleset with a function (with parameters)
- for each rule that passes, pushes it temporarily to the "x" symbol, and expands the rule `#animal# hat` to get a final value
- constructs a new rule set from each of those final values, e.g.: "cat hat", "bat hat", "rat hat"

As we can see, these all chain together nicely, though the syntax gets very difficult to read. At the bottom right of Figure 15.2, you can browse some of the visualizations that I've used to automatically parse and visualize particularly complex rules (and spot errors)

15.4.3 Little

Tracery is far from the first template-based text expansion language. From conversations with indie developers (including Kingdom of Loathing creator Zack Johnson), many of us have created our own templating language over then years, and from comparing notes at the Roguelike Celebration in 2017, many have remarkably similar features (a few even had "a/an" modifiers).

About the same time as Tracery was published, Orteil (a notable indie game developer) released a generative language named RandomGen[293]. Users could write RandomGen generators and host them on his site. We can see some similarities to Tracery, as well as some features that Tracery lacks (like the ability to attach subproperties to choices):

```
$animal
cat {action:meows}
dog {action:barks}
fish {action:swims}
$phrase
I have a [animal,#pet]. My [#pet] is [#pet,as action].
```

Why did Tracery take off, and RandomGen did not? I think one major reason was that Tracery is *portable*. Users of RandomGen wrote programs that would only run on a single website, controlled by someone else. They could only be *seen* by people who visited that site. Tracery was initially designed to be embeddable as a low-dependency Javascript library, which made it usable in games and apps. Crucially, it also was able to be ported into a Node library by George Buckenham, who created CheapBotsDoneQuick to host Tracery grammars as Twitterbots (more on CBDQ later). Its simple syntax was also was ported (with varying levels

of fidelity) into many other languages, allowing its spread to a variety of domains (like Unity and Twine games). Tracery's grammar syntax is also JSON, a format that is immensely common and well-supported. This made it safe and simple for CheapBotsDoneQuick to import and run arbitrary input from anonymous users. JSON also has wide range of linters, validators, and structure editors available, for free. In short, Tracery was little enough to live *across* a creative ecosystem, not just in a single app supported by a single person. While I created Tracery, its development has been the result of a community of development. More than any other clearly-a-casual-creator app in this dissertation, Tracery exemplifies the chaotic, community-led, networked nature of casual creators.

15.5 Community ecosystem

Tracery was originally designed and implemented by me. But other people have used its syntax in a variety of ways, either by porting the Tracery parser-and-expansion to new languages and platforms, building new tools to output Tracery grammars, or embedding Tracery generation in different contexts. This section lists many of these adaptations and adoptions.

15.5.1 Tracery Ports

Common Lisp: Steve Losh [246]

R: DL Miller [274]

Rust: Paul Woolcock [440]

C# Unity: Max Kreminski [227], 5ive Bullet Games [5], George Buckenham [41]

Ruby: Eli Brody [38]

Haxe: Ciro Duran [131]

Choba: Jurie Horneman [198]

Twine: Matthew Balousek [21] "There are lots of cool sources of data out there. For example, dariusk/corpora. If you create a passage and tag it corpus, twinecery will do some importing for you."

Twine 2: Michael Thomét [409]

Lua: Milan Gruner[174]

JVM: Almas Baimagambetov [19]

Python: Allison Parrish [300][302]

Swift: Benzi Ahamed[8] Also adds functions(modifiers with parameters) and calls (modifiers that don't change the returned value), which then was ported *back* to Javascript as Ephemeral by Martin Pichlmair [308]

Node: George Buckenham [43]

15.5.2 Languages Inspired by Tracery:

These languages grew out of, or were inspired from, Tracery

Bracery: Ian Holmes [233]

Expressionist: James Ryan [336]

Tracery's portability also meant that it could be embedded in a range of different systems. Several games use it, including *It is as if you were playing chess*[23], *Quiet as a Stone*[433], *Panoramical*[s161], *Monstr (A Dating Website for Monsters)*[415] and a number of Dietrich Squinkier's experimental games (*Most Sincere Greetings, Esteemed One*[390], *Interruption Junction*[388], *Fitzwilliam Darcy's Dance Challenge*[389]). In each of these games, it was used for flavor text (*It is as if you were playing chess* and *Quiet as a Stone* in game, *Panoramical* as a Twitter-post caption for screenshots) or dialog (*Monstr (A Dating Website*

for Monsters), *Interruption Junction*, *Fitzwilliam Darcy's Dance Challenge*), or assigned tasks (*Most Sincere Greetings, Esteemed One*). I also used it in many of my own games, including the unreleased *HipChef* and *Revisions*[29].

These two games are both idle games where users curate generative content (recipes or research projects) to optimize for changing conditions. The Tracery-generated text creates texture and interest that turns a pattern-matching game into a fun fantasy[30]. These games also both contain casual creator elements, as the user is slowly exploring the generative space defined by the built-in grammar, and seeking out and collecting their favorite artifacts from the space.

By far the greatest driver of Tracery interest has been CheapBotsDoneQuick. George Buckenham released this site in 2015, whereupon it began to drive an absolute revolution in creative bot-making. Large numbers on non-coding users began to create Twitterbots, and continuously expanded the possiblity space. They created small humorous bots (@hipstercocktail) and participatory art bots (@autoflaneur) and discovered that Tracery could make emoji patterns (@infinitedeserts). Users started making more complex emoji art[37]. Ranjit Bhatnagar showed my original Tracery editor creating SVG graphics, so Buckenham revamped CBDQ to also pre-render and upload SVG artworks(@softlandscapes). Despite the safety of JSON, user-created SVG can have a variety of horrifying inclusions, such as running arbitrary Javascript, and loading external files[9].

All of this was made possible by the hosting, but also the visibility that these artists had to each other on Twitter. Many of these techniques moved virally through the community, usually on Twitter, but sometimes through small workshops[31] or online tutorials:

[29] `http://www.galaxykate.com/revisions/`

[30] ...if you enjoy the fantasies of running a restaurant or graduate AI lab

[31] In one tutorial, an attendee learned the repercussions of using a bot to spam other attendees `https://twitter.com/GET_TUDA_CHOPPA/status/755343399562121216`

> "it shows a handful of useful patterns for incorporating large amounts of data into your Tracery grammars that would be impractical or inconvenient to do with a Tracery generator on its own."[303]

Users spoke openly about their bot-making process, and shared experiences of tuning generative spaces over time:

> "At the top level, the list above decides which kind of phrases can go with others within the same tweet. For example, fragments about bad weather should be kept away from fragments about play or joviality. Some of these categories ended up a lot broader than is suggested by their old names. You might also have noticed a number of repetitions in the list above. The randomness in CBDQ is simple; one entry = one chance. This makes it quite easy to make an event rarer by adding more references to the phrases which should appear more regularly. I found this out early on when I introduced the concept of meeting other voyagers on different boats. It was just happening too often so I had to make some of the more desolate environmental imagery have a much greater chance of appearing. I would watch the feed for a day or two and then make adjustments like this."[26]

Many tutorials and tools were created for Tracery. Several alternative editors and creators have sprung up. The *gulf of evaluation* seems to be solved only with the single approach of showing lots of generated examples, but these tools take different attacks on the *gulf of execution*. One solves it by creating new bots with the push of a button, as "[a] tracery-based bot that makes SVG artbots." [172]. Others attempt to crack it by supporting the user in generating by proposing new suites of functions for this task[370], such as importing from Darius Kazemi's *Corpora* (a collection of user-sourced lists) and Martin O'Leary's *Ketchum*.[289][32] *Cheap Markovs Traced Quick*[33] by Serin Delaunay lets users generate full grammars from sample texts, puckishly remarking, "Tracery grammars allow too much

[32]"This is code for generating lists of similar words, using word vector similarities from the fasttext data, released by Facebook. It's useful for building collections of words to slot into generative grammars, such as Kate Compton's tracery."

[33]`https://github.com/serin-delaunay/cheapmarkovstracedquick`

human creativity (and take too long), so I'm making a tool that makes Markov chain Tracery grammars #PROCJAM"[34].

Others tilt at the problem of sharing grammars. While CheapBotsDoneQuick allows users to opt-in to sharing their code, this can be hard to find, and makes it hard for users to share small grammars that *don't* have an attached Twitter handle. Beau Gunderson and Gregory Weir both made online Tracery editors where the user's grammar is saved as part of the URL, allowing the grammar to travel in all of the ways that URLs can (tweets, pasted, link shorteners).

A few users use bots themselves as a creativity tool. As we will see in chapter 17, creating a prompt can be an important part of enabling casual creativity. These bots serve as improv partners[74] thanks to CheapBotsDoneQuick's response feature, where a bot which is directly spoken via Twitter can respond with a generative Tracery response.Figure 15.4 Less directly, several bots are operating as creative tools and inspiration (@artideabot, @artassignbot, @InstrumentBot) on Twitter, as well as one Tracery art-improv bot *not* on Twitter but installed in a small receipt printer physically hung on a wall in Nicole He's artwork *The Best Art.*[186]

Finally, some Tracery projects playfully decorate other generative pipelines. *Every Rat in NYC (That People Complained About in 2016)*[s69] combines a live feed of rat reports from New York City's department of sanitation with Flickr photos and Tracery biographies of the rats. My own *Imaginary City*[35] was a prototype of how to turn a bland Voronoi graph into an interesting space using Kevin Lynch's *Image of the City* framework. Dan Hett piped Tracery into a music and sound visualization language to DJ a club.[36] Allison Parrish, maker of many Tracery tools, finally brought Tracery full circle to its roots, by using it to decorate

[34] `https://twitter.com/SerinDelaunay/status/927000672091951104`
[35] `https://github.com/galaxykate/imaginarycity`
[36] `https://twitter.com/danhett/status/794226517765226496`

and narrate a planning-based story outline, a "minimally opinionated rough-draft JavaScript framework for producing narratives through simulation"[301][37].

While many of these are not interactive, in them we can see Tracery's use in a generative pipeline, which could lead to new *interactive* expansions on these systems.

15.6 Chancery

Chancery[161][65] is a quiet footnote to this chapter. It is a Tracery-like language that was developed at Google, open-sourced, and then provided me several happy years of hard labor attempting to wrestle it into a casual creator. Chancery maps, like Tracery grammars, are JSON objects. Instead of a single grammar, though, these maps describe a state-machine which represents a conversation. This map contains:

- A grammar for use in generating Tracery(2) text
- A list of states (like Tracery's grammar, in a dictionary of IDs matched to syntax objects)
- At least one Pointer, which represents movement through the state space
- A world object, which can be read or written by the pointers movement through the space.

Like the first Tracery editor, it also comes with a simulator and visualization suite. The user can type into a virtual chat app, and the bot can respond, in text or synthesized speech, after the text is expanded with Tracery. They can watch

[37]This is not the first combination of Tracery with traditional story-planning: at the very beginning of Tracery I had coupled it to output from Mexica[80][442]

the pointer move from state to state in the conversation. For our "casual creator IDE" checklist, it's got great augmentation! But for every problem Tracery has in the *gulf of execution*, Chancery has a much worse version. The concept of a map is not hard to get users to imagine (I describe it as a board game). But the syntax is much more complex than Tracery. It will assuredly need a structure editor (as Tracery did), but more urgently. Unfortunately, each attempt so far has spiraled out of control into tangled complexity.

It will also need a CheapBotsDoneQuick platform. If hosted, it could engage with Twitter, but also the Google Home and Amazon Alexa. If users could publish their bots, would others learn from them? Is Chancery too complex and too heavy to have the nimble adaptation that Tracery succeeded with? These questions, at the time of this dissertation, are still unanswered. Hopefully when this is complete, I will have time to return to Chancery and see just how far a "casual programming language" can stretch.

Tracery and Chancery show the limitations and affordances of casual creators for programming. While the core software is itself has some properties of a casual creator (error tolerance, generative surprise), the original systems are surrounded by other tools and interactive interfaces that apply different social and technological solutions to the problems of their slow grokloop, and bring them into a social environment (CBDQ) where users can self-promote their generative creativity.

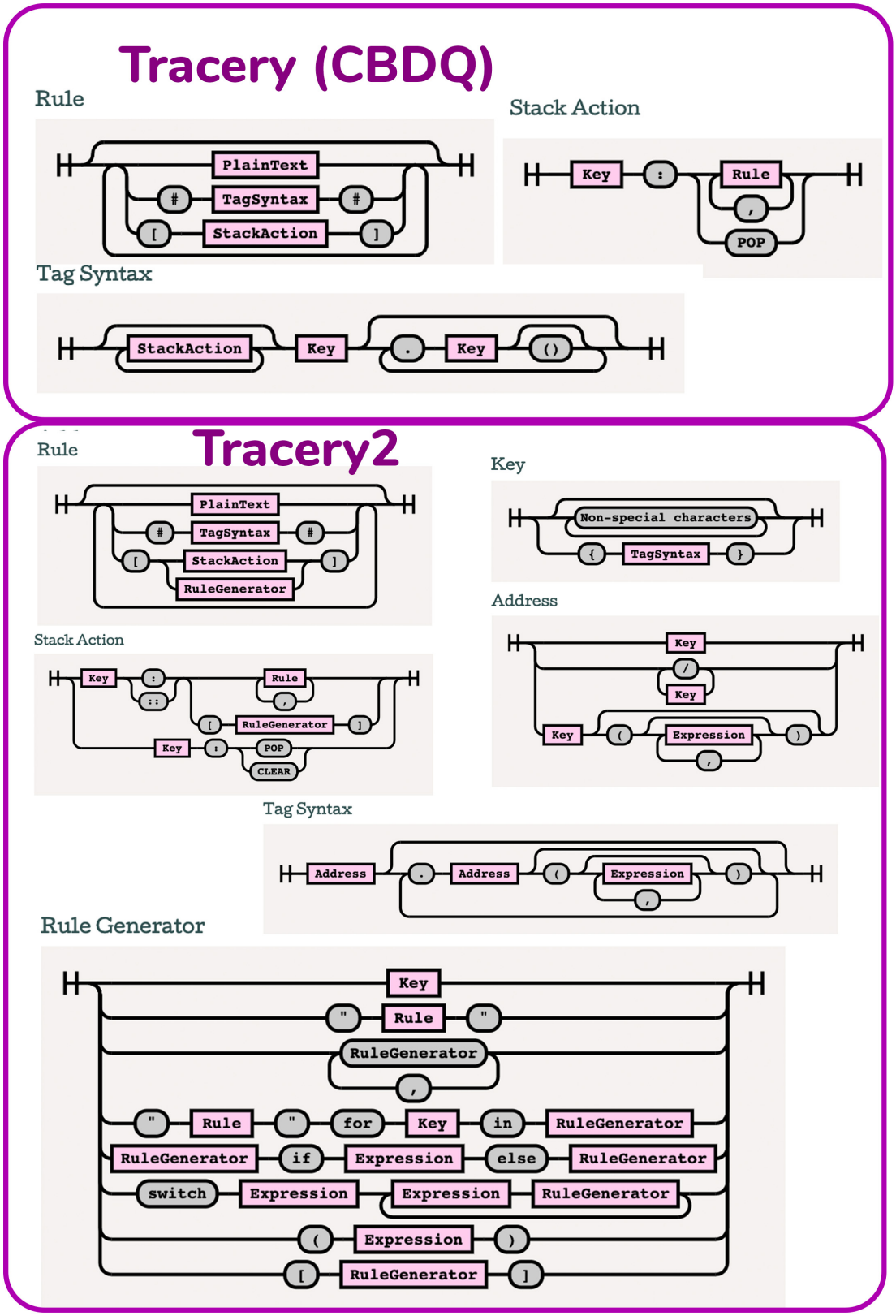


Figure 15.1: Railroad diagrams for Tracery 1 (CBDQ edition) and Tracery 2



Figure 15.2: (left)A variety of visualizations of Tracery grammars with syntax coloring, graphs, colocation diagrams (by Ben Kybartas), frequencies of leaf-node selection, and automated Tracery error-checking. None of these have been as successful enough to add to the editors, except hierarchical decomposition of output (top right two) and rules (bottom right two), which is especially important as Tracery 2 syntax gained complexity.

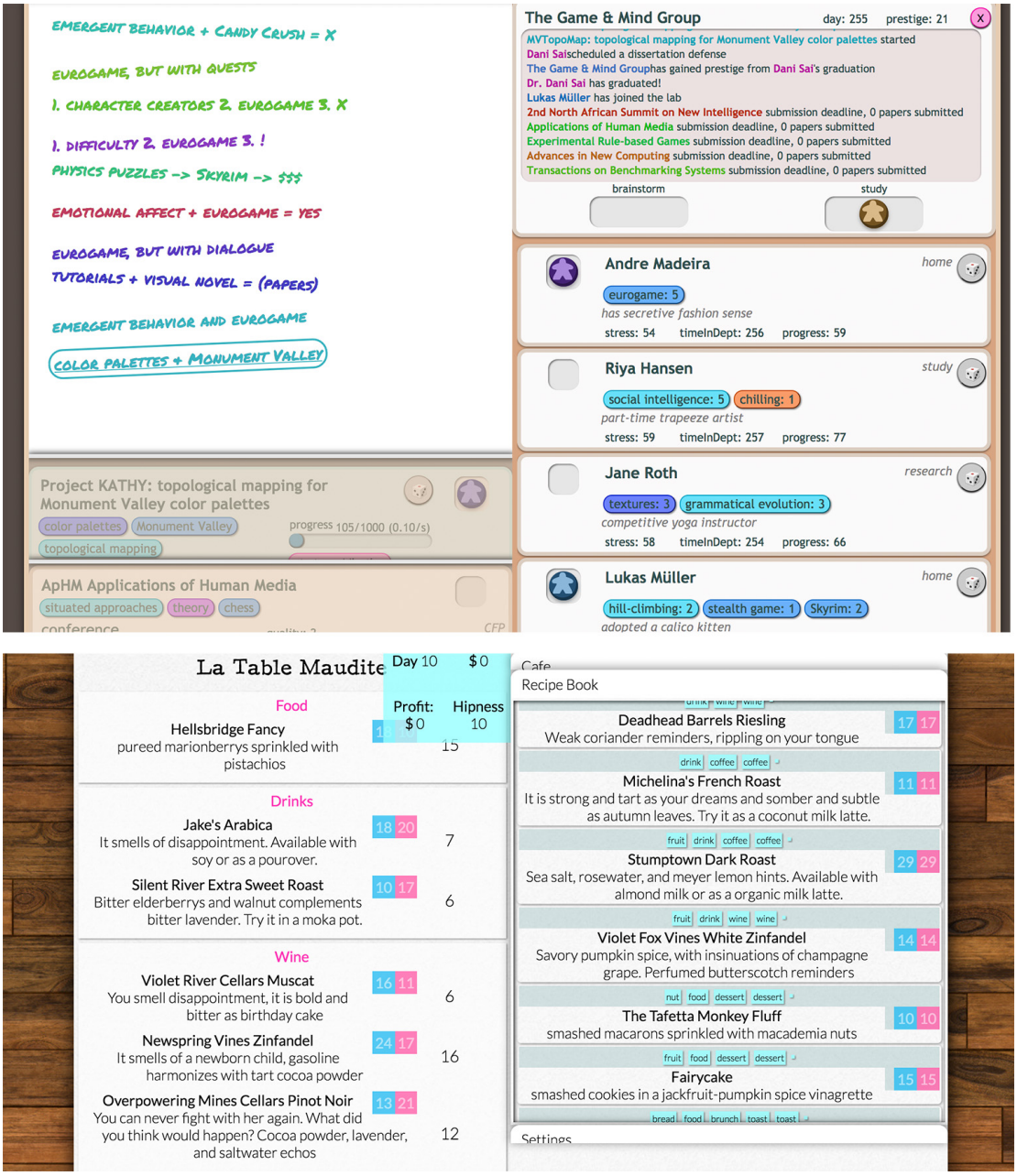


Figure 15.3: *HipChef* and *Revisions*, two of my own games with embedded Tracery as a core feature of the experience.

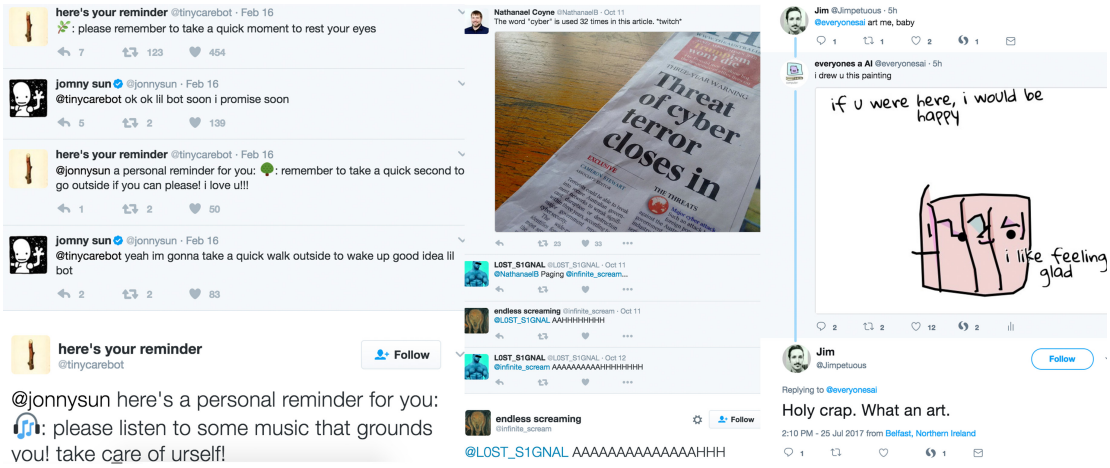


Figure 15.4: Users playfully creating scenes with bots. A few "honeypot" bots trick ill-tempered users into repetitive conversations with them, but most bot conversations are knowingly using the bot as the straight-man partner in an improv conversation.

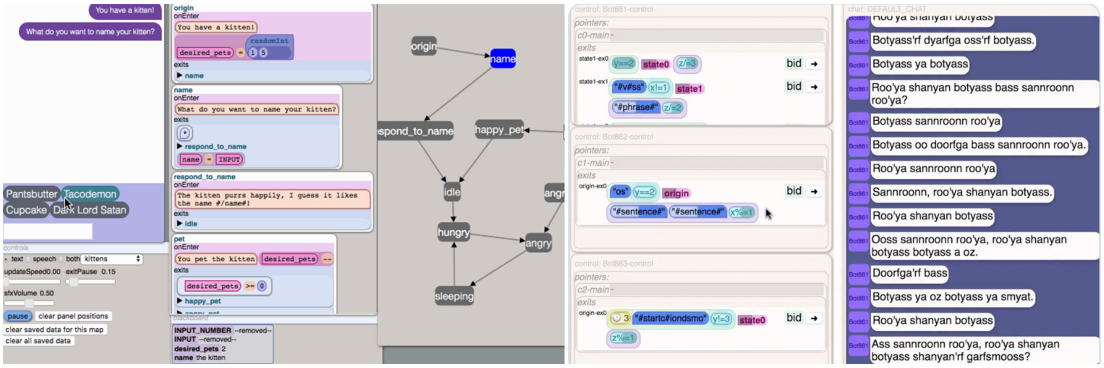


Figure 15.5: The original open-sourced Google code as "Bottery" (left), and my recent experiments at making a multi-bot casual-creator platform for writing bots in Chancery (formerly "Bottery"). (right)

Chapter 16

Pattern - Entertaining Evaluations

> "Only a machine can appreciate a sonnet written by another machine."
> — Alan Turing[428]

So far we have established that successfully bridging the *gulf of evaluation* is one half of creating a fast grokloop. We have seen casual creators in which the gulf is narrow, and the grokloop can become a continuous cycle, allowing for the user to gain expertise and confidence in the strangest of interactions. We have also seen *slow* creators, in domains that requires seconds or hours of the user's attention to evaluate an artifact. In both of these perspectives, the bridging of the gulf is the means to an end. In this chapter, we will explore an interesting phenomena, casual creators where the act of evaluation is *part* of the autotelic experience, **entertaining evaluations**.

16.1 Instant partial feedback

In chapter 13 we saw slow creators for which there *were* forms of (near) instantaneous feedback, like the visualizations of *Sentient Sketchbook* or the histograms of expresssive range analysis. These evaluations were **partial** evaluations of the actual experience of the artifact being created. They did not tell a full, or necessarily accurate, picture of the artifact, but they gave the user the confidence to keep rapidly exploring the space. We can look at other creators and editors and see many different forms of partial evaluation: the *Spore Creature Creator*[s216] has a complexity meter, which serves a reminder to users of how computer-intensive their creature is.[1] Twitter has a character-count meter in the corner of the tweet-authoring panel; it can't tell a user how good or thoughtful their tweet is, but sure can tell them when they are close to 280 characters. Several text tools (Google's *Perspective API*[s166], Text.IO[s231], and online bias checkers) promise data-backed immediate feedback as part of a learning loop, often referring explicitly to a loop that sounds a lot like a grokloop:

> "The API uses machine learning models to score the perceived impact a comment might have on a conversation. Developers and publishers can use this score to give realtime feedback to commenters" `https://www.perspectiveapi.com/#/`

> "Say hello to the loop. Augmented writing is fueled by massive quantities of data, contributed by companies across industries and around the world. Textio's predictive engine uses this data to uncover meaningful patterns in language, guiding you to stronger communication and better business outcomes. Those results add even more data, more fuel for the engine. Together, the companies using augmented writing are building a learning loop that delivers the collective insights of the whole directly to your fingertips. [..] Imagine if you could see in advance how people will respond to what you say. That's augmented writing: the power to see the future that your words will create. Now, the art of writing is also science." `https://textio.com/`

[1] During development, the marker on this meter was a picture of the famously-angry lead engineer's face, getting redder and redder.

These evaluations are often couched in the language of science, sophisticated machine-learning, and proven fact, but so far they are only very partial, inaccurate, and brittle approximations: "They saw that changing "You are great" to "You are fucking great", made the score jump from a totally safe 0.03 to a fairly toxic 0.82."[267]. A more playful tool, *I Write Like* performs a literary text analysis with similarly inaccurate results: this chapter is calculated to be written like Arthur C. Clarke.

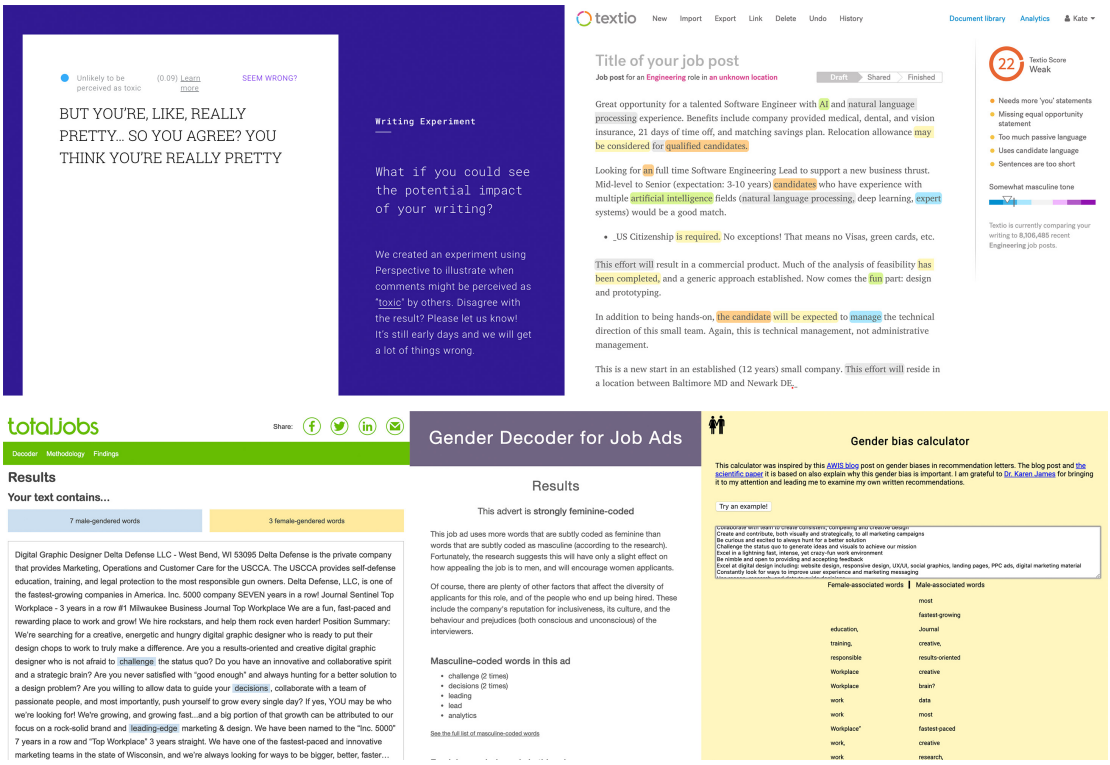


Figure 16.1: (top) Google's Perspective API and Text.IO sell companies a promise of "scientific" confidence in writing.(bottom). Three gender-bias tools return diverse values for the same sample text: the TotalJobs Gender Bias Decoder (`https://www.totaljobs.com/insidejob/gender-bias-decoder`), Kat Matfield's Gender Decoder (`http://gender-decoder.katmatfield.com/`), and Tom Forth's Gender Bias (`https://www.tomforth.co.uk/genderbias/`).

These tools are designed to sell a false sense of confidence about the decidedly non-autotelic domain of business writing. We can adopt these methods (or use

them as patterns) for experiences where the inaccuracy of the evaluation creates a fun story (like *I Write Like*) rather than a bad business choice. In an autotelic creative experience, we don't mind if the user's sense of confidence is false or based on a misleading evaluation (as it is in games like *Sim City*[s204][224]) so we have more latitude to explore a *range* of evaluation approaches.

Imagine we have some casual creator that produces a 4 minute pop song with generated lyrics, melody and backing score (beats, harmony, musical filler). What could we show the interactor to help them understand their current song? There's a nearly unlimited supply of things we *could* show them:

- give them a play button for the full song, but then it would take 4 minutes to evaluate each artifact. But since pop songs are repetitive, we could loop a 10 second clip of the main melody and beat.

- give the interactor the waveform of the generated audio, or the sheet music. Experienced musicians might be able to read these at a glance, but if our audience is casual interactors, they might not be able to read anything meaningful from these visualizations. We could give them a printout of the lyrics. This may be more helpful, as people can quickly read pop-song length lyrics, so they could quickly evaluate whether the lyrics seem promising.

- evaluate the music along various axes that are meaningful to listeners (range, speed, intensity, major/minor).

- use the interactors' cultural knowledge as a waypoint, by displaying some evaluated distance between the current song and "Uptown Funk", "Happy", "Beat it", "We Are Never Ever Getting Back Together" and "Single Ladies" (as calculated by a machine-learning algorithm).

- make use of **peripheral attention** by putting all of these on the screen together, and let the interactor glance over and listen for whatever stands out to them about the current music (a sort of musical "situation room"). After some amount of playing around with such a complex environment, would the interactor begin to intuitively "read" the data? And would this lead them to interacting more confidently?

The exercise above was an imaginary situation, but it should be clear that there are many ways to *partially evaluate* otherwise difficult-to-evaluate artifacts. Some of these evaluations are strongly representative of the artifact, but take time or expertise to evaluate (playing the song, reading the sheet music). Others are immediate, but have less information, are less accurate, or only represent one aspect of the artifact. As we have seen above, the accurate representation of the artifact is of only secondary importance (at least in a casual creator). Instead, we optimize for the user's ability to understand or feel confidence in the evaluation (and their knowledge of the possibility space). This is why evaluations which are visually-displayed or link to a known social context (Taylor Swift, Arthur C. Clark) have such a powerful draw.

What, then, if we flat-out *lie* to the interactor via the interface? In the machine-learned proximity example above, the designers of the system may rightly have limited confidence in the algorithm's ability to measure some "objective" distance to Taylor Swift, but if we show it to the user as a graph, it becomes perceived as objective mathematical truth (in the way that simulations with rigged rules often become their own proof[36]). If we replaced that metric with a bar labeled "quality" that went up and down with different generated songs, would the interactor believe it? Would it be useful to them, or help them make better songs? Would they enjoy it or feel more confident if their song scored high on

a completely fictional quality metric? What happens when we go from a lossy metric to a fictional or mendacious one?

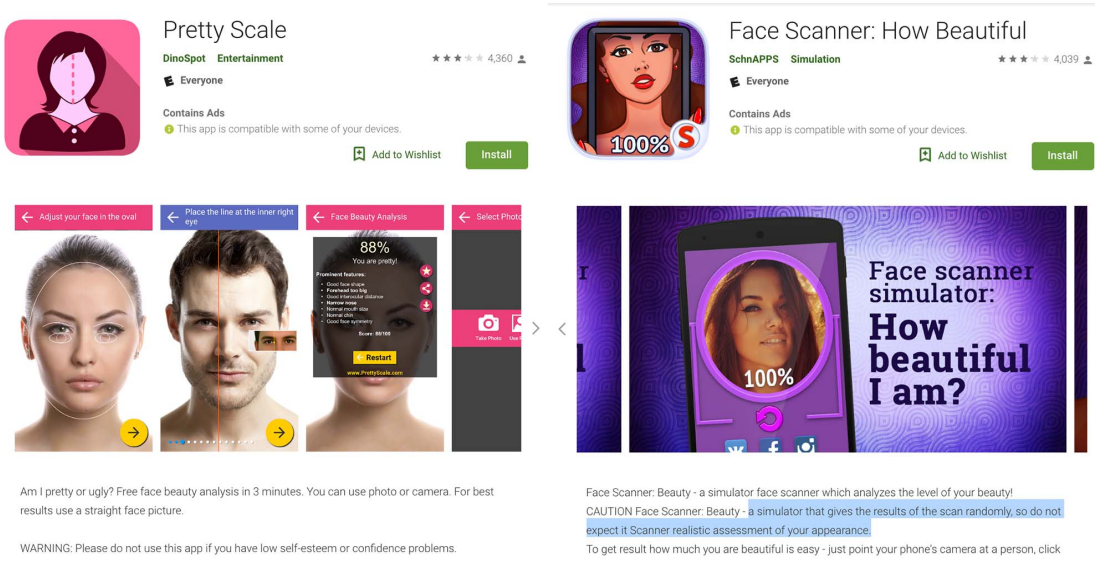


Figure 16.2: A few of the "beauty scanners" on the Android store. App notes include: "WARNING: Please do not use this app if you have low self-esteem or confidence problems." and "gives the results of the scan randomly, so do not expect it Scanner realistic assessment of your appearance." These *may* be examples of a "dark pattern" for "entertaining" evaluations. (`play.google.com/store/apps/details?id=com.prettyscale.PrettyScale`), (`play.google.com/store/apps/details?id=com.schnapps.face.scanner.beauty`).

16.2 Fictional feedback

There are many kinds of ways that creativity software can lie to the user, from flattering fictions about information that doesn't exist in the system ("that looks great!") to fictions that encourage the user to see intelligence or independence in the system that may not be there ("Clementine will remember that"[2]). It

[2]In Telltale CEO Kevin Bruner's keynote at ICIDs 2016, he claimed that while many user interactions were recorded by Telltale's interactive narrative systems for algorithmic use, the famous "X will remember that" callouts were not coupled to that recording system, and could

is possible to use these to convince users to buy tools or spend time with the system in a manipulative way: in *DIY Fashion Star*[s52], users are encouraged to buy hair or makeup upgrades (with in-game currency bought with real money) so that virtual agents will upvote their virtual-Instagram selfies. Many fictions are just theatrical illusions: we choose to believe them, because it makes the virtual environment more intersting, or, for casual creators, provides the magic feather we need to believe in ourselves creatively. This section will explore several ways that casual creators have used fictional feedback to give users a sense of mastery and creative safety.

"Yes and" is often cited as one of the primary tenets of improvisational theater. It is shorthand for the need for improv performers to acknowledge and accept the narrative "offers" from their stagemates in order to move on with the scene[210]. If an actor say "Here were are, bold pirates", and the other says "...lost in the depths of space", then they have yes-and'ed the offer and built on it. If instead they said "No, no, we're cowboys", then they have rejected the offer. Yes-and creates a space of emotional and creative safety, where performers can collaboratively build a scene and make large creative jumps, without fearing rejection or reprisal.

Non-human systems can also follow this improv principle (becoming better improv partners). A UI can acknowledge, accept, and build-on a user's input by playing a "click" sound on a button-press (*acknowledge*), dimming the button or screen (*accept*), and carrying out some computational action (*build-on*). This lets a user know that they clicked correctly, that the UI "heard" that button press and didn't freeze up or reject it (the dreaded "bonk" sound), and then presents them with a new digital state as a reward. UIs are carefully designed to make us feel acknowledged, accepted and responded to.

be independently added by writers if they wanted the tell the player that a particular choice was an important one, regardless of system implementation.

That pattern is important in any piece of software, but we can be more playful with it in a casual creator. The *Spore Creature Creator*[s216] was designed by Chaim Gingold, an amateur improv theater practitioner. He consciously embedded many of the principles of improv into the system, including yes-and. When a part is added or removed from a creature by the user, it springs to life, looking at that new part and nodding gleefully at it, or rarely, shaking its head in disapproval. Its approval or disapproval is based on absolutely no calculations. But it acknowledges and responds to the user's *particular* actions. The virtual people of *The Sims*[s206] will similarly walk up to just-placed objects, expressing curiosity and approval or disapproval, though they have more world-model knowledge to make a "reasoned" evaluation.

Often, it is the *absurdity* of a virtual being evaluating a human that provides the enjoyment in an experience. We know that an artificial intelligence is alien to us. Sims are hilarious because of their simplified rational-yet-irrational logic (eating in the bathroom, leaving babies on the stove). The more irrational the behavior is that we are trying to model, the more fun it is to make a computational caricature of it. For this reason, it is particularly common to model *art critics* as digital systems. For many people, modern art criticism is a baffling word-cloud of misappropriated terminology, ridiculous statements, and black-box heuristics. What makes a good painting these days? Why not just ask a robot?

And so we see AI projects where a physical robot (dressed as an "19th century aesthete") is an art reviewer[297] rolling around a gallery and commenting on art, and a virtual "artist" (a generative pipeline with a webcam) generates fictional rationales based on current events[62].

BECOME A GREAT ARTIST IN JUST 10 SECONDS and Art Sqool

This same genre of humor has given rise to at least two please-the-art-critic casual creators[3]. In the first, *BECOME A GREAT ARTIST IN JUST 10 SECONDS*[s90], the humor of the ridiculous premise is made obvious in the title. As the game opens, the user chooses from several "lessons" that seem familiar from traditional art training: "Getting started: the blank page", "Still-life drawing", "How to draw a landscape", etc. Choosing one opens a panel with a canvas of smudged color and a sample image on the right (a bowl of tomatoes), as well as an evaluation at the top of "6.69% similar". Mashing keyboard keys spams a number of generative/transformational glitch art operations: sliding rows of pixels, dropping rectangles and squiggles of color, or even playing chopped-up recordings of art-speak words like "dichotomy". The player's impossible task is to use these unpredictable tools to make something that the computer recognizes (through whatever logic it uses) as a bowl of tomatoes.

In *Art Sqool*[s74], the users plays as Froshmin, a new student at ART SQOOL. Their professor is Professor QWERTZ, who is a spoof of both art professors and the deep-learning models trained on classical art that have become such a meme in the years since *GREAT ARTIST* was released in 2013: "I am an advanced NEURAL NETWORK, trained on over 100,000 terraflops of the greatest art in the world. I will be your faculty advisor this semester."

The player is given a randomly selected assignment to draw ("100 degrees" "a wiggly thing" "the back of your head" , "the ugliest thing on campus", "the winners and runners-up of a pie eating contest"), and can walk around a surreal campus collecting tools, colors, and brushes (reflect back on the chapter 10 on gameplay-linked tool unlocks). When ready, they can draw a picture on the *MS*

[3]There are several more that I have found, but they are less well-developed

[4]Released **just in time** for this dissertation, in Feb 2019

Paint[s141] style canvas, and jump off the terrain to reappear in from QWERTZ to be graded on color, composition, linework, and approach, an evaluation that seems to be gleefully random. The designer, Julian Glander, echos some of the properties of creativity we have covered so far in this dissertation:

> "I wanted to design it so that it's hard not to make art," says Glander. "You have prompts to get you over the blank canvas-phobia that a lot of people have. It's silly and it's simple."[163]

The computer's inability to judge good art from bad levels the playing field, as does the clunky drawing interface and silly tools. Every user will probably make scribbly art of the same level, and prompts like "go really cubist with this one" leave room for choice while making it clear that there is no *right* answer to the prompt (even if QWERTZ could detect one). Both *BECOME A GREAT ARTIST IN JUST 10 SECONDS* and *Art Sqool* can also be seen as spoofs or critiques of serious art-making, built around casual creators:

> "I think one way of thinking about Art Sqool is as an insightful critique of the arts education. It's a cynical critique that sees art schools as a big private grift that ask you to do a bunch of arbitrary assignments in a classroom setting that are graded purely by magic."[229]

Deep Improv

We can see examples of AI-partner improvisation whenever users challenge each other to let their phone's predictive text complete "I was born" to write their biography[5][263]. Half improv and half fortune-telling, these memes rely on the limited possibility space (chosing one of three words, or only choosing one side or the other) to explore the possibility space that the phone's predictive AI has generated from our conversations. Botnik[6s19], famous for their predictive text

[5] `https://twitter.com/therealbradg/status/926260526270652416`
[6] `https://botnik.org/content/harry-potter.html`

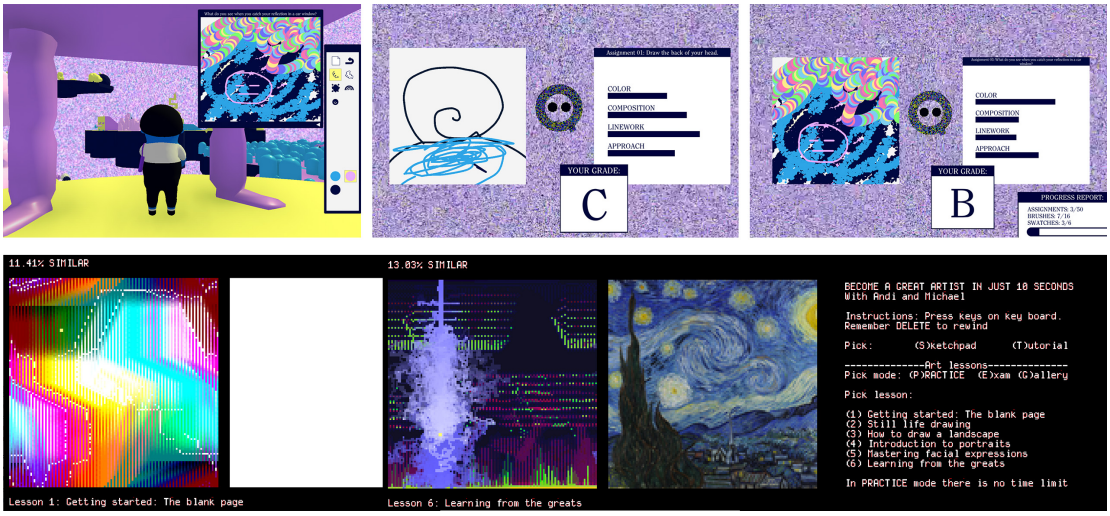


Figure 16.3: *BECOME A GREAT ARTIST IN JUST 10 SECONDS* and *Art Sqool*

spoofs of Harry Potter, band names, and other famous media, released predictive keyboads (trained on different corpora) as a casual creator app for anyone to play with. Like the predictive text phone game, but with more specific signifiers, these let users quickly explore the Markov Chain possibility space in their favorite media.

Deep learning can also be used to create entertaining evaluations. In *The Laughing Room*[s1197], an installation at MIT, an inviting living room space was constructed and strewn with books on humor. Participants could sit down on the couches and talk to each other, but as they did so, a microphone would record their speech, parse it into text, and feed that text to a network trained on humor to identify the statement's proximity to comedy. If it found a high humor value, it would trigger an audience laugh-track, which laughed maximally to such one-liners as "oh wow fruit", "i would eat tater tots" and "fish"[8]. In this project, the system played a sitcom audience, laughing or not laughing at users' attempts at

[7] `https://metalabharvard.github.io/projects/laughingroom/`
[8] `https://twitter.com/ahandvanish/status/1064227326987300864`

comedy. While the project's description was a viral success with lots of coverage, and made its argument about surveillance, it was a mixed bag from an experience perspective. From watching the stream, users often did not feel acknowledged and would keep talking without a response from the system, while sometimes pausing awkwardly to wait to see if the system had anything to say.

In my system, *Human-Free Photobooth (how not to be seen)*[s94] I also use deep-learning as a tool to subvert expectations of "artificial intelligence"'s intelligence. This system (based on a concept by me, Gretchen McCulloch, Janelle Shane, and Jonny Sun) is themed as a photobooth run by an AI who hates humans. It consists of a webcam's live feed, which is piped into a python program and shown on screen, so the users can use it as an augmented mirror, and a stack of five machine-learning algorithms:

- face detection: if faces are detected, the bounding boxes are added to the video so that users can tell they have been spotted (and the screen displays the dialog "Ew humans")
- body detection: if no faces are detected, it attempts to find bodies. If found, these bounding boxes are also drawn to the screen
- text-to-speech: if neither bodies nor faces are detected, the AI asks in a computerized voice: "Are there any humans there?"
- speech-to-text: processes what the humans say in response to that. If they answer "no", then it takes a picture
- image-labeling: the picture is sent through an image-labeling network to determine what non-human objects the AI "thinks" it sees. These labels are added back to the image, which is then revealed on screen.

The goal is for the humans in the photobooth to playfully trick the AI into seeing no humans to discover what humorous things it *does* see, a fun pattern in playful AI[358]. Along the way, it also trains them how to disguise their faces and body-shapes until they no longer appear human, which trains them on the possibility space that the AI can recognize (improving awareness of AI's current ability in this field, and their own ability to outwit it). Like *The Laughing Room*, this work uses machine learning algorithms that can be slow or imperceptive, so I added a lot of improv acknowledgements and acceptances, like drawing the boxes to the screen, showing "ew humans", noting when no humans were found, asking for permission to take the photo (by asking them to lie to the AI), and then yes-and'in the photo with the AI's improvisational labels. In early tests, the experience seems to invoke a lot of creative behavior and use of local props.



Figure 16.4: *Human-Free Photobooth* tested in the EIS lab and at a party. While the AI wanted to describe the local furniture it could be certain about, it was also willing to guess at the identity of the human objects. Humans especially like lying to the AI, by yelling "no!" when it asked if there were any humans there.

Fictional social feedback

Sometimes the AI does not represent an art critic or a partner, but instead plays the role of a whole field, from Csikzentmihalyi's environmental creativity framework[111]. *The Sims*[s206] originally began as a casual architecture simulator called *Home Tactics: The Experimental Domestic Simulator* for "giving grown-ups some tools to design a dollhouse"[58]. Sims would come into the house to give it a score.[209] Users became attached to the Sims, shifting their role from playful judges to the star of the show, but in the finished *The Sims*, they still wander around judging objects. In *Rollercoaster Tycoon*[s192], users build creative roller coasters in a casual-creator style editor. They can virtually ride the coaster, but mostly its value is based on its automatically calculated Excitement, Intensity, and Nausea scores, which can very involved calculations that players attempt to optimize[9]. Most of the game after that consists of watching the tiny park-goers evaluate the park, attempting to path-find, buy food, and avoid the splatter of nausea-inducing rides. Both *The Sims* and *Rollercoaster Tycoon* officially endorsed trying to please your tiny judges, but many users enjoyed the counterplay of optimizing for misery instead[200]. Any quality meter presented to a user has both a maximum *and* a minimum, and users can always be relied on to explore both extremes, especially in a system that encourages creativity.

Digital Clockwork Muse and *Techne*

One can also build systems that explicitly model Csikzentmihalyi's environmental creativity framework (and general models of creativity as an environmental structure). Rob Saunders and Kajon Grace propose a framework for games that can be played in a "DIFI" (Domain, Individual, and Field interaction) environ-

[9] `https://www.reddit.com/r/rct/comments/79v8e2/a_comprehensive_guide_to_the_excitement_rating_on/`

ment.

> "According to this framework, a creative system has three interactive subsystems: domain, individual and field. A domain is an organized body of knowledge, including specialized languages, rules, and technologies. An individual is the generator of new works in a creative system, based on their knowledge of the domain. A field contains all individuals who can affect the content of a domain, e.g., creators, audiences, critics, educators, etc. The interactions between individuals, fields and domains [...]form the basis of the creative process in the DIFI framework: individuals acquire knowledge from domains and propose new knowledge evaluated by the field; if the field accepts a proposed addition, it becomes part of the domain and available for use by other individuals." (Towards a Computation Model of Creative Cultures)

They identify a few language games like "guessing games" in which agents attempt to communicate about an object until their language synchronizes, "imitation games" in which agents try to match each others recognition pattens (not unlike Generative Adversarial Networks), and "generation games", where a "client" makes a request of a "designer" agent.

> "the expressed requirements contain a set of features that a design must contain but the specific form of the design will depend on the generative capabilities of the design agent. To succeed, the design agent must generate a design that satisfies the requested feature set by exploring the space of possible designs for a design that satisfies the design requirements as it understands them."

We can see echoes of these "games" of interconnected creativity in Saunder's earlier work *The Digital Clockwork Muse*[343]^{s43}(*DCM*). This system consisted of a population of multiple art-producing art-appreciating agents. Their art production "minds" were simple function trees that produced generative artwork. Their art-appreciation "minds" were self-organizing maps (SOM). The agents were then simulated in a virtual world where they could see each others artwork and evaluated it with their SOMs, and decide how they felt about it in via the hedonic

function (see chapter 4), where easily-classified works were dull, impossible-to-classify ones were very bad indeed, and somewhere in the middle were deliciously novel yet recognizable works.

This work is fascinating and inspiring, and seems to simulate many interesting artistic and creative phenomena. As a person making many art bots, and having written the language behind a wildly-popular artbot platform (see chapter 15), I wanted to turn this work into one where any user could create an artbot and set it free in the simulator, to see it mingle with, be evaluated by, and learn to hate other bots artwork.

This idea became the *Techne* project[296][s229], a bot-making platform. While the whole system never came together, it had several subsystems that were of interest:

Heterogeneous media: All of the bots in *DCM* used the same art production and criticism modules, but with different tuning. *Techne* ran with four kinds of "media" that bots could know: generative poetry, generative art, importing Shakespeare and taking credit for it, and mashups, where text was pasted on generative art.

Self-opinion: in *DCM*, each bot had a decoupled production and criticism module. In *Techne*, bots could have these modules, but they could also be for different media, so a Shakespeare plagiarism bot could have opinions on art, and vice versa. This allowed bots to play artist or critic for different media, as well as being able to critique their own work. Our critic modules were too simple to be of much interest, but they let us see the beginnings of dynamic behavior.

Language adoption: All of the bots, with the exception of the Shakespeare plagiarists were running on Tracery grammars. The bot-generator for these bots constructed a new Tracery grammar for each new bot. This grammar had some

limited set of expansion rules selected from a metagrammar, such as only being able to use blue, green, stars, and spiral compositions, or to only write poems with farm nouns, scary adjectives, and the pronoun "Thou". When bots found other bots whose work they enjoyed (and that they had a compatible module for), they could *adopt/steal* each other's tools, just like the human creativity that spawned a book with the title "Steal Like an Artist".

The ability to see and hate one's own artwork made for entertainingly cathartic moments when playing *Techne*, as did introducing bots to mentors with whom they could improve their work, and watching the little artists evolve from despair to mild self-satisfaction.

16.3 Human social feedback

Not all entertaining evaluations are performed by artificial intelligences. Humans quite enjoy evaluating each other's creative work. The card games *Apples to Apples*[s6] and *Cards Against Humanity*[s25] both use a very simple "generator" of juxtaposing two (sometimes more) cards to create interesting cognitive dissonance or storytelling. This juxtaposition is "scored" by a critic-player selecting the best pair from those provided by the creating-players. Players take turns being the critic, but every player and social group has their own social expectations and beliefs about what makes a combination clever or funny. In board games with a creativity element, this is by far the most common mechanic to "fairly" rate creativity, and is often colloquially referred to as "an *Apples to Apples* mechanic" to explain gameplay of a new game of this type[10].

Fewer digital systems use this mechanic, instead they often use massive crowds to reach a consensus. No-one can reach massive crowds like Google, so it used

[10] `https://boardgamegeek.com/boardgamefamily/46175/judging-games`

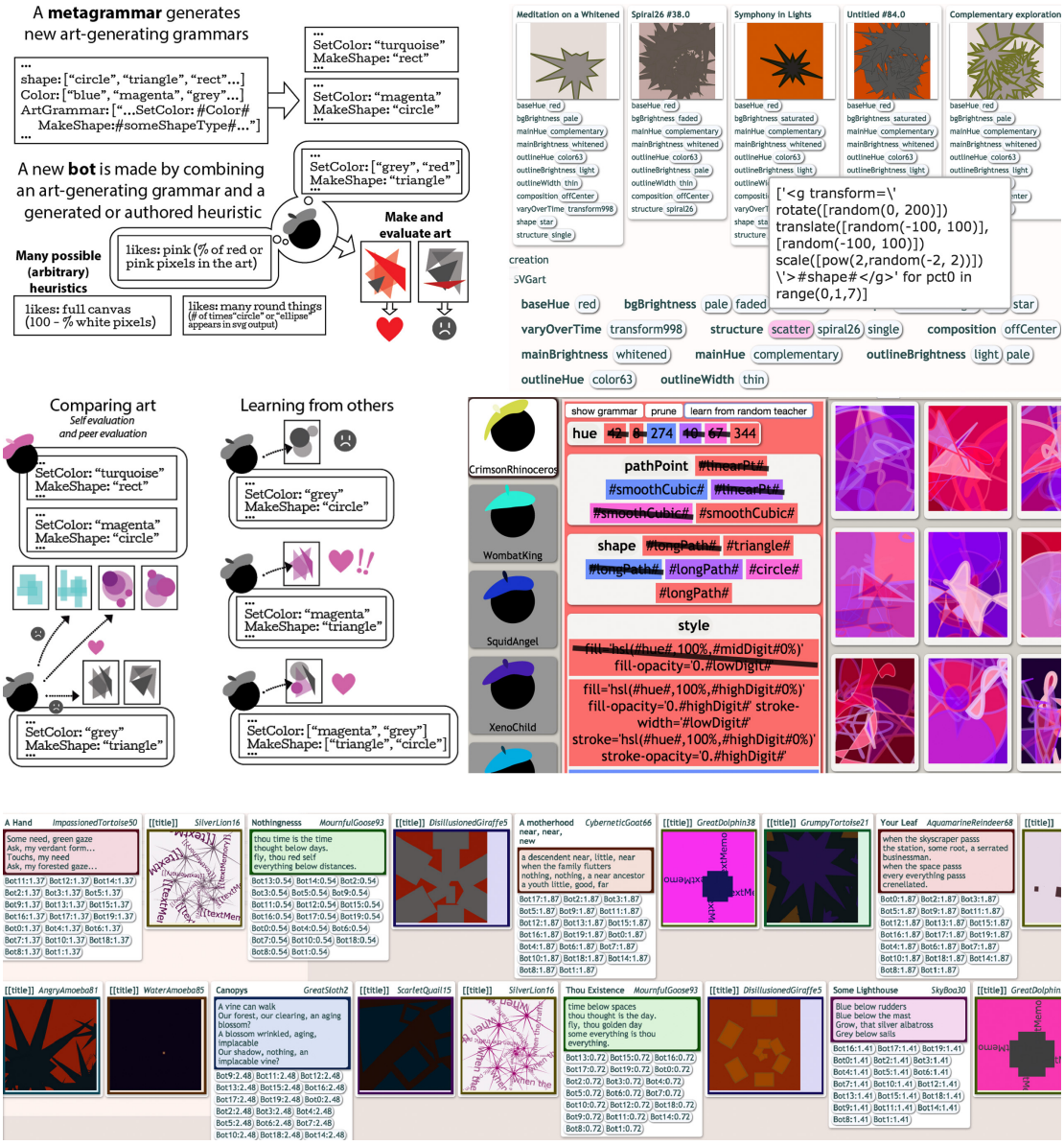


Figure 16.5: Bots in Techne. In the top row, the metagrammar generates a bot with some subset of tools (grammar expansion rules) for a given domain (i.e., only red, muted colors, spiky shapes, and spiral layouts). In the middle row, bots compare their art to the art of their creative community. If they find one they like better, they will frenetically try out their idol's tools, hoping to "improve" their art. In the bottom row, many bots were released onto a virtual platform to try to "talk" to other bots of different domains (mostly, the mixed-media bots stole lines from the poets to use in their posters).

crowd-sourced evaluation to evaluate user-provided image labels in a game[11s98]. Users could submit any label, ridiculous or serious, but would only get points for typing a label *also* guessed by the other user. A heavily mediated version of this can be found in Google's *Quick, Draw!*[s182] game, where users drew pictures to be guessed by a trained classification algorithm. The human users provided feedback on the computer's guesses, but their feedback and drawings were also training the next generation of the network, so humans were indirectly guiding the guesses as well.

Many casual creativity systems have social sharing built in: online portals where users can share and like and comment on each others work (see chapter 20). That is certainly a powerful form of human feedback for creativity systems. There are also ways of digitally or algorithmically mediating that feedback as well, or using it in a generative pipeline. The recent game *Occupy White Walls*[s152] is "a PC sandbox-building, AI-driven MMO where people play with Art, developed by people who really love architecture and abstract characters"[12]. In it, users construct galleries with a plethora of design elements (brutalist, rococo) and fill the walls with collected art.

This game/MMO/gallery-simulator uses a few different *entertaining evaluation* mechanics. The system promises to bring the user new works that they like, with the magic of machine-learning:

> "Clever this, you don't actually have to do anything for DAISY to work her magic - just look at art that interests you, don't look at art that doesn't and voila! You will soon start seeing lots of awesome art - selected just for you, not by some 'art world' type who knows better, but by your new art BFF who just happens to be a sentient machine. It's not just you, every player in OWW helps train the AI to make better recommendations for everyone else."[13]

[11] `https://crowdsource.google.com/imagelabeler/welcome`
[12] `https://www.oww.io/`
[13] `https://www.oww.io/`

Readers who have followed this chapter closely may wonder how *accurate* this AI is, given the scientistic language around it, but it also doesn't seem to provide any feedback about why it brings pieces of art (no visualizations or partial evaluations to build trust), nor is the user acknowledged when their gaze has reached the threshold to register interest in a particular piece of art. This AI leaves something to be desired as an improv partner. Aside from the user's gaze providing evaluation for the AI's art taste, the user's gallery is evaluated by the presence of other users, who may teleport into new galleries and walk around. The longer they spend there, the more money the player gains to buy artworks.[108]

There are many ways to evaluate the artifacts of casual creators. Evaluation can be done to bridge the gulf of evaluation and speed up the grokloop, but it can also be done to create an entertaining experience.

Chapter 17

Pattern - No Blank Slate

"... the empty brush hung poised, heavy with destiny, irresolute in the air. My hand seemed arrested by a silent veto. So very gingerly I mixed a little blue paint on the palette with a very small brush, and then with infinite precaution made a mark about as big as a bean upon the affronted snow-white shield. It was a challenge, a deliberate challenge; but so subdued, so halting, indeed so cataleptic, that it deserved no response. At that moment the loud approaching sound of a motor-car was heard in the drive. From this chariot there stepped swiftly and lightly none other than the gifted wife of Sir John Lavery. "Painting! But what are you hesitating about? Let me have a brush - the big one." Splash into the turpentine, wallop into the blue and the white, frantic flourish on the palette - clean no longer - and then several large, fierce strokes and slashes of blue on the absolutely cowering canvas. Anyone could see that it could not hit back. No evil fate avenged the jaunty violence. The canvas grinned in helplessness before me. The spell was broken. The sickly inhibitions rolled away. I seized the largest brush and fell upon my victim with Berserk fury. I have never felt any awe of a canvas since."- Winston Churchill[60]

"Stravinsky once declared that the most frightening moment in musical composition was being faced with a blank sheet of manuscript paper. It is a moment when all options are open, and virtually all constraints are absent. It is rather like being on a boat in the centre of the Pacific - you can, in theory, go everywhere, but why should you want to go anywhere?" - JL Alty[11]

"Writing is easy: All you do is sit staring at a blank sheet of paper until drops of blood form on your forehead." - Gene Fowler

Any person who has been creative (writing a dissertation, for example) knows the terror of the blank slate. The moment of a blank canvas is the time of the most potential possibilities, good and bad, but as we saw in the section on Flow in chapter 8, too many choices can cause paralysis, not creativity. The quotes above, from creatives in painting, music, and writing, all reflect this problem.

Its not only an intuition, though, it can also be measured. One study tested Chinese children learning English through storyboarding[277]. They provided more or less constraint, with a finished strip with empty dialog balloons, or with the panels removed at the middle or the end for the children to fill in themselves. Overall they found that "(t)he familiar comic format contributes structure and comfort, thereby increasing participation" when compared to letting children just make up stories without any panels, but based on familiar toys: "Even with these aids, our results confirmed the claims made in the literature that the open-ended nature of SB (storyboarding) left children largely bewildered and frustrated". They also confirmed that increased structure also increased participation, **up to a point** when the children felt trapped by the structure: "when we constrained the CB (comicboarding) more, in the case of Dialogue blanks (fill-in-the-blank balloons), children not only failed to generate any ideas but filled in the blank with illogical, trivial phrases so that they could move on." The *right* level of constraint for a casual creator, then, is something that inspires and creates direction, but does not feel *binding*.

I did not find any studies where creativity was measured for different levels of constraint (for whatever way one could measure creativity). While the previously-mentioned study didn't measure "creativity" of the ideas, it did measure the children's participation, which, assuming that the children participated because they enjoyed the activity, could serve as a stand-in for creative experience in a

casual creator. Remember, in a casual creator, we are concerned less about the brilliance of the output than the experience of creating it.

17.1 Examples of blank-slate solutions

17.1.1 Starter artifacts

How can a casual creator prevent a blank-slate situation? One way is by starting the user with a dull or unfinished artifact. This has several advantages. As seen in the grokloop, we know that users can theorize about what would improve a concrete example (even if they could not invent a wholly new one).

Giving user a starting artifact limits the number of choices from the infinity of a blank slate to a reasonable three:

- Modify the artifact, in any of the limited ways provided by the creator
- Conceptualize a way to subvert the existing artifact, making a big creative leap
- Reject the artifact, at which point they are at the blank slate stage, but by their own choice

For example, here are two books which purport to help casual artists, by proving prompts. For the one on the left, its assumed that the artist will fill in pinup details. But the artist has instead reconceptualized the poses into action hero fight sequences. For the The Heartless Machine Guide to Drawing, the makers of the book intend the user to come up with unusual or creative reimaginings of the prompt. In both of these cases, the user could ignore the lines altogether (though it might be easier to buy a blank piece of paper)



Figure 17.1: Left: The *Doodle Pad*, a mid-century book of "sexy lady" templates to fill in, reinterpreted as action heroes by David Jablow [140], Right: The *Heartless Machine Guide to Drawing* (`https://www.kickstarter.com/projects/494572698/the-heartless-machine-guide-to-drawing-fill-in-the`). Note the caption: "Starting with a blank page is terrifying! This book alleviates that fear by starting you with a partial drawing. You just fill it in."

In a digital creator, we have more flexibility of what to provide. For some creativity software, like *Photoshop*[s168], it would be bizarre to open it up to see a photo of a strange family, ready for retouching. A game like *The Sims*[s206], though, famously provides a "starter family" of Mortimer and Bella Goth. The player of *The Sims* is encourage to practice the UI controls with this family, and whenever they want, move to the blank slate of a freshly created family and an empty lot.

Sometimes, a blank slate isn't even possible in a casual creator. For generators that are parametric, not additive like a grammar or tile-based generator, there is no such thing as an "empty" artifact. In the *Create-a-Sim*[s206] mode, there is no "empty sim": the Sim is either a pre-authored configuration, or a randomly

selected artifact from the space. This is especially easy (and common) for the all-parametric space chapter 11: just pick a random vector in the cube!

17.1.2 Inspiration

A more subtle form of breaking the blank slate is by providing some form of inspiration or direction. This direction isn't codified in the artifact itself, but can be provided outside the generative system, even outside the software. This property makes it possible for communities to self-organize and organically break the blank slate. Examples of this include Twitter art jams (#mermay, #myartistdesk, #drawthisinyourstyle), constrained-writing jams like "Six Word Story" challenges, the Facebook group *Quilt-a-Day* (which provides an image and color palette for a quilt), and "Create-a-Sim" challenges.

While several of these could use any tool (including pen-and-paper), inspiration jams and challenges like these can be created for any tool. Each one creates a prompt, like the starting artifact, that provides the user a limited set of choices:

- Move *with* the inspiration, making small changes and operating *within* assumptions of the prompt (taking advantage of genre expectation, for example)
- Reconceptualize the inspiration, finding a creating reinvention of it (mermaids in space, dinosaurs in frilly dresses)
- Reject the prompt. Just don't participate in this jam

As in the artifact-based prompts, the user feels the freedom of choice to go with the flow, push against it, or opt out entirely. This reduces the number of immediate choices from an overwhelming number ("do anything!") to three options, which is a better number to create flow, as observed in Csikszentmihalyi's

flow theory[110]. This seems to prevent the paralysis seen in the storyboarding study, where students felt uncomfortably bound by the constraints, while letting the user choose how much guidance they are in the mood for.



Figure 17.2: Pictures for quilting inspiration and a finished quilt:Quilt inspiration and finished quilt by Anne Sullivan.



Figure 17.3: Create-a-Sim challenges:Youtubers and Twitch streamers seem particularly fond of Create-a-Sim challenges, performing difficult creative tasks in front of an audience.

AstroSquatch

> "LEAVE US ALONE" effectively ended SETI.

ezacharyk

> We thought they were stars. Wrong.

DJScience

> Free doomsday machine. Only used once.

Figure 17.4: 6 word short stories:Microfiction stories from IO9's Six Word Scifi story challenge[275].

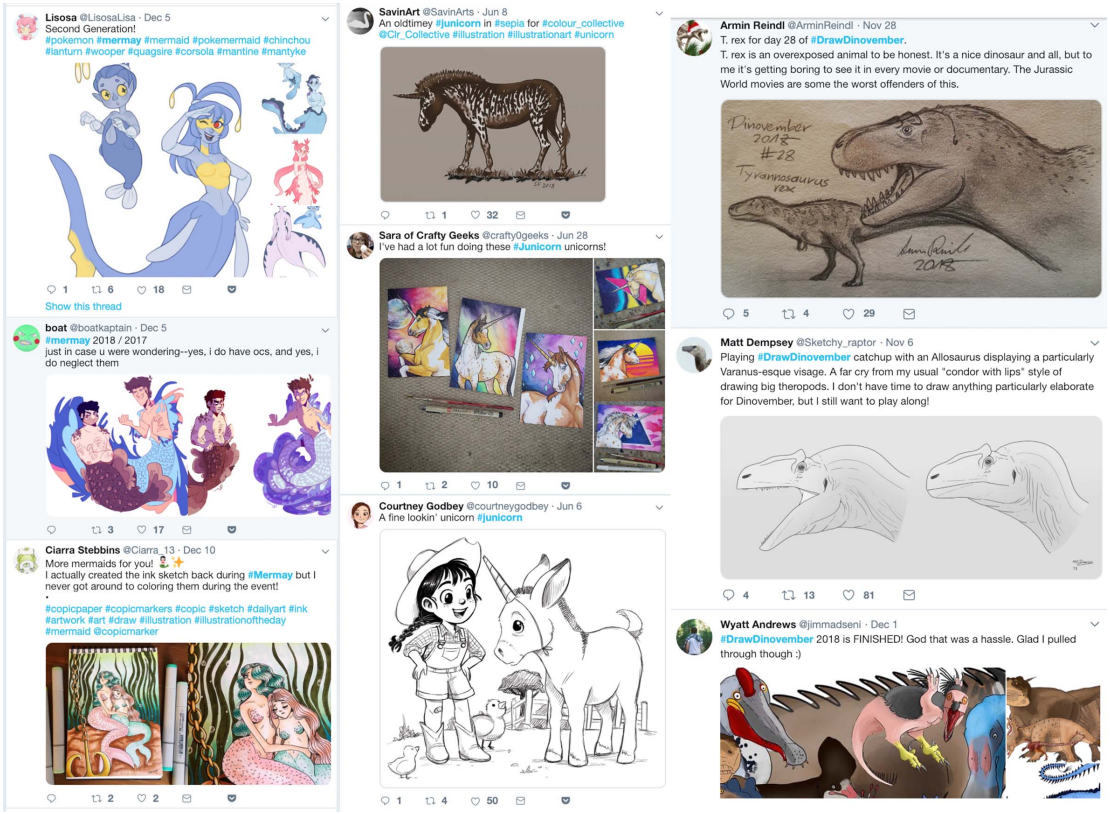


Figure 17.5: Art memes of Twitter. With #Dinovember, #Junicorn, and #Mermay, there is a drawing challenge month for every taste.

17.1.3 Structure and Templates

Not every user is particular gifted at using the tools available. Their artifacts aren't aesthetically pleasing, or they can't draw well (especially common in creators that have more flexibility). Yet we want to provide a chance for them to show off their creativity as well. In this case, memes develop where the creativity lies in making a clever yet parsimonious re-imagining of a very-limited template. A great example of this was the "Howdy I'm the Sheriff" meme on Twitter, where users made very rudimentary "sheriff" figures out of emoji, signifying different kinds of sheriffs based on the emoji selected for the body. These "skill-free" or "low-participation" challenges allow more users to participate. Other forms of this

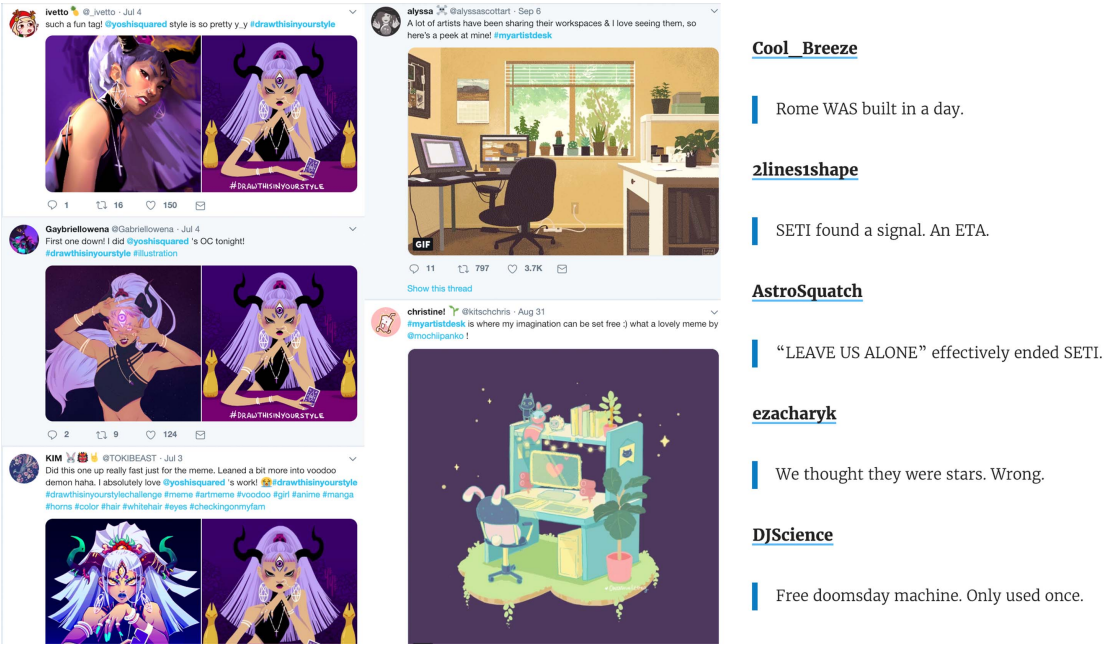


Figure 17.6: (left) Multiple drawings of the same character in different styles:Yoshi Yoshitani's original drawing as reinterpreted by dozens of Twitter artists for the #drawinmystyle challenge, where one user posts a character, and everyone reinterprets it. (center)Artists draw their surroundings in #myartist-desk. (right) Not *every* challenge meme is for visual art: the scifi site IO9 hosted a "six-word story" challenge for its readers[275].

include the "Change a letter"/"remove a word"-style challenges, where users have to come up with a movie title that, when a word is removed or a letter is added, becomes silly (e.g. "Shaving Private Ryan").[1]

Why do these work? Like the prompts above, it is all about reducing the number of options, so that even an uncreative user can brute-force their way to creativity by trying out a lot of the available options. It also levels the playing field: while many users would look at a beautiful mermaid drawing and expect that they couldn't match it, most users would expect that they could play the sheriff or word games as well as the best creators of the genre.

Because games like these hang on achieving a big semantic shift from relatively

[1]The *vast* majority of the posts of this type make a sex pun or dirty word from the title. Accessible creativity has its drawbacks.

small changes in a fixed structure, this kind of prompt will only work in creators that have very semantically-heavy components, like emoji. The teapot editing software probably will not achieve prompts like this. But when they are possible, they open up all of the strong emotional properties of socially-played games to users who are less skilled.



Figure 17.7: Templated creativity activities only require the user to make one or two choices (instead of generate whole new ideas). "#ChangeA..." memes let users figure out which word in a movie or book title to change for humorous effect. The "emoji sheriff" meme creates sheriffs a range of colorful expressive purposes, while only changing the emoji used.

17.2 Case Study: Spore

The *Spore Creature Creator*[s216] is a rare example of a vertical-slice of all the approaches here, from the in-system starter artifacts to community-developed creativity challenges.

The *Creature Creator* shipped with a fixed number of "starter bodies", tube-like shapes that were the minimal possible body. The creature bodies were parametric spinal column sizes and positions, with a tree of body parts hung off of any given spinal point. It *would* have been possible to create a random creature

in the space, but it would have looked dreadful, and would have been a poor introduction to the aesthetic possibility space. Instead, the starter bodies were bare, and were carefully designed by the team artists to suggest known shapes: dog-like mammals, giraffes/ostrich, bulky T-Rexes, or top-heavy monsters. These worked as prompts, which the users could continue to improvise on, subvert, or reject by shaking out and resizing the spine.

The game itself could have provided additional direction, encouraging the users to create "environmentally suitable" creatures that would survive. But during development, it was reasoned that this would limit the kinds of creativity we were seeing early on, and might limit the wilder forms of creativity. So while the game encouraged or even mandated certain survival traits (creatures had to have a mouth), we left this form of constraint mostly dormant.

The *Spore*[s215] community-management team created a variety of challenges (led by Theresa Duringer), from part-use challenges to challenges that showcased the recontextualization the players were already performing in game. The building editor was the most flexible of the tools, and players began using it to make sculptures and dioramas. So the Holiday challenges were to make things like Christmas scenes and fruitcakes[130]. Other challenges gave less-skilled users a chance at victory, such as winning by just uploading any creation.

Not many structured templates arose, because the system had so few intensely-semantic pieces (no emoji), but we did see the rise of a "mailbox" meme, where users passed around a mailbox template and decorated it to have a designate place to keep their direct messages (more on this in the chapter 20 section).

Creativity often needs a *spark* to ignite it. Sometimes people think of that spark as coming from inside the head of some great genius, but often it is an external source that they are reinterpreting, riffing on, responding to, or push-



Figure 17.8: Christmas buildings:a holiday contest for scenes made in the building editor[404].

ing against. When used well, generative systems can make use of these "starting points". Perhaps the inspiration is provided by humans in a social context, in themes and challenges like #mermay or #nanogenmo[s145], and the casual creator provides a way for unskilled users to express that theme. Other times, the casual creator's job is to provide ridiculous and unexpected suggestions, as one advantage of a system is that it may not *know* about expectations. *Instrumentbot*[s102], a Twitter account that challenges its sound-artist owner to make instruments from

strange tools, and *The Best Art*[s11], a physical printer that prints out art assignments[2]

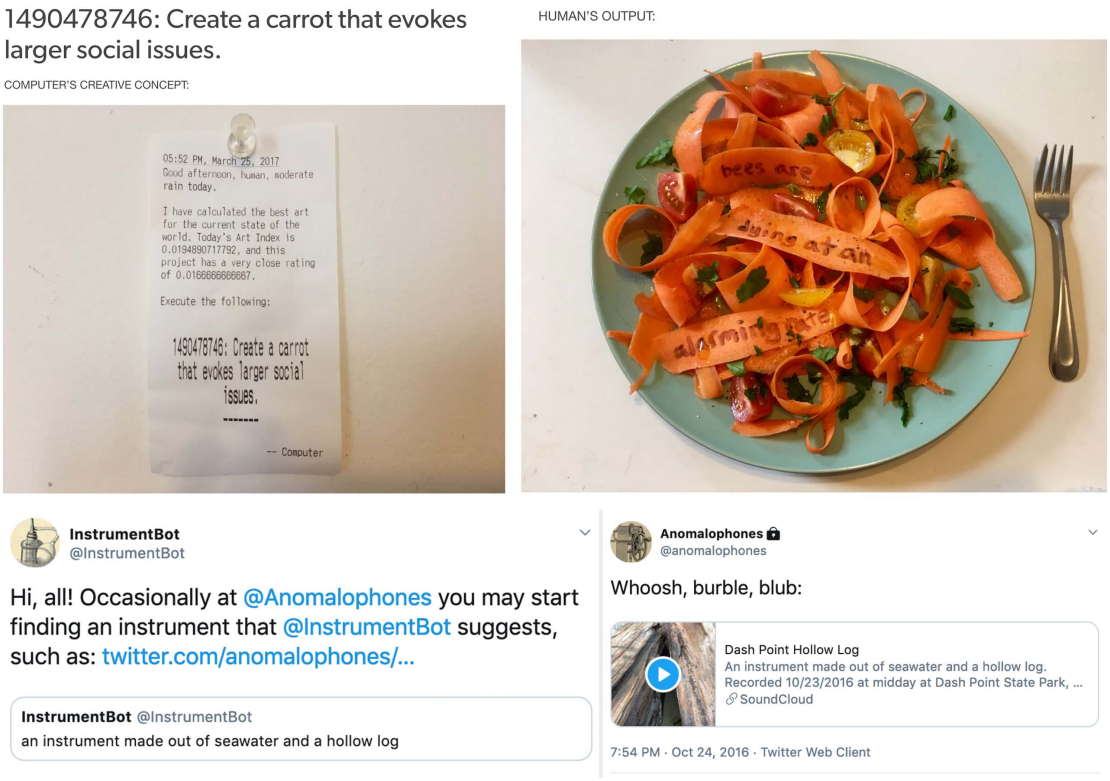


Figure 17.9: Challenges and creations from *@instrumentbot* and *The Best Art*, as fulfilled by their human partners. (`https://the-best-art.computer/`).

[2] *The Best Art*'s assignments are based on the "Art Index", an intentionally surreal generative pipeline using the position of the space station, the weather, Kickstarter pitches, and gallery art descriptions as input.

Chapter 18

Pattern - Safety

"We don't make mistakes. We just have happy accidents." - Bob Ross

Safety seems to be a key foundation of creativity, at least for the experiential causal creativity that this dissertation is concerned with. We can expect, then, that casual creators must provide safety to the user. But what is "safety" in creativity? Mid-century psychologist Carl Rogers identifies creativity as a phenomena that exists only in set of conditions[325][1]. He then defines those "conditions of psychological safety and freedom" that "maximize the likelihood of an emergence of constructive creativity":

- "Accepting the individual as of unconditional worth"
- "Providing a climate in which external evaluation is absent"
- "Understanding empathetically" ("see you and what your are feeling and doing from your point of view[...]and still accept you")
- "Psychological freedom" (permissiveness)

[1] "The farmer cannot make the germ develop and sprout from the seed; he can only supply the nuturing conditions which will permit the seed to develop its own potentialities."

Can a computational system perform these tasks? It is perhaps too much to ask a virtual pottery wheel to implement simulated empathy, but we can understand these as *metaphorical* goals. In chapter 16, we saw ways that systems could be good improv partners by *acknowledging* and *accepting* user's actions, even when all that meant was playing a sound or animation.

We can imagine systems that are "accepting" of users by

- using UI acknowledgments in response to user actions
- celebrating content (putting a frame around it, lighting it nicely, literally placing it on a pedestal)
- complimenting it, or appreciating it in some way, as seen in the variety of evaluations in chapter 16

Quick, Draw![s182] was a machine-learning project from Google, structured as a game of *Pictionary*[s170] in which the user had only 20 seconds to draw an object in the way that an AI could guess what it was. As the user drew, the AI would shout guesses in a synthesized voice, which acknowledged the user's new additions to their drawing, and let them know if the AI was getting closer with each additional line. The game was wildly successful, thanks to the lively and affirming behavior of the AI, and Google acquired millions of hand-drawn representations of objects like "cat", "belt", or "The Mona Lisa". Google open-sourced that data, leading designers to make use of the drawing data to create new AI-powered drawing tools, like *Magic Sketchpad*. *Magic Sketchpad*[s1292] is an interactive drawing experiment with a collaborative AI partner trained on the *Quick, Draw!* data in which the user, in an *MS Paint*- style editor, draws line drawings. When the network can predict the next strokes for the selected style (e.g. "cat"), it "completes" the

[2] `https://magic-sketchpad.glitch.me/`

drawing with eyes and whiskers, or whatever strokes it statistically thinks come next. While its not the smartest algorithm, it can add legs to a flamingo and a tail to a scorpion (sometimes). Even in its confusing strokes, when its unclear what it's trying to render, it feels playful. When it doesn't add anything at all, it feels like it is *no longer alive*. These two systems have the same improvisational loop: the user draws, and the system "yes-and"s it, like in improv[210]. In both systems, the AI response can be wrong without disrupting the illusion of life, but *Quick, Draw!* feels more alive than *Magic Sketchpad* for the reason that it never *stops* acknowledging each action by the player.

Many casual creators "celebrate" the user's productions: *Petalz*[s167] displays the user's flowers in nice pots (which are collectible and upgradeable), *Pixel.ly 3D*[s174] replaces the gray-scale menu images of its coloring-book patterns with the colorful ones that the user has finished. *Tap Tap Fish - AbyssRium*[s1] is an idle game mostly about gaining fish for a coral reef, but it suggests that the user buy new lighting, backgrounds, and even costumes for the reef. A very large number of creators encourage the user to share their creations on Twitter, as this is a way to let the user express pride while also subtly advertising the system. Word-search game *Alphabear*[s4] has a tiny casual creator at the end of each level, where users can refresh a template that is filled in with the words they found during the game. It is well designed to celebrate the user's skill at the game (posting a template with a long or specialty word tells your friends that you found it in game), while also practicing other forms of safety, like letting the user refresh for a new filled-template as many times as they want, and letting them skip it altogether.

Rogers' requirement of "absence of external evaluation" is a complex premise in casual creators. We have seen that entertaining AI evaluations, and even social

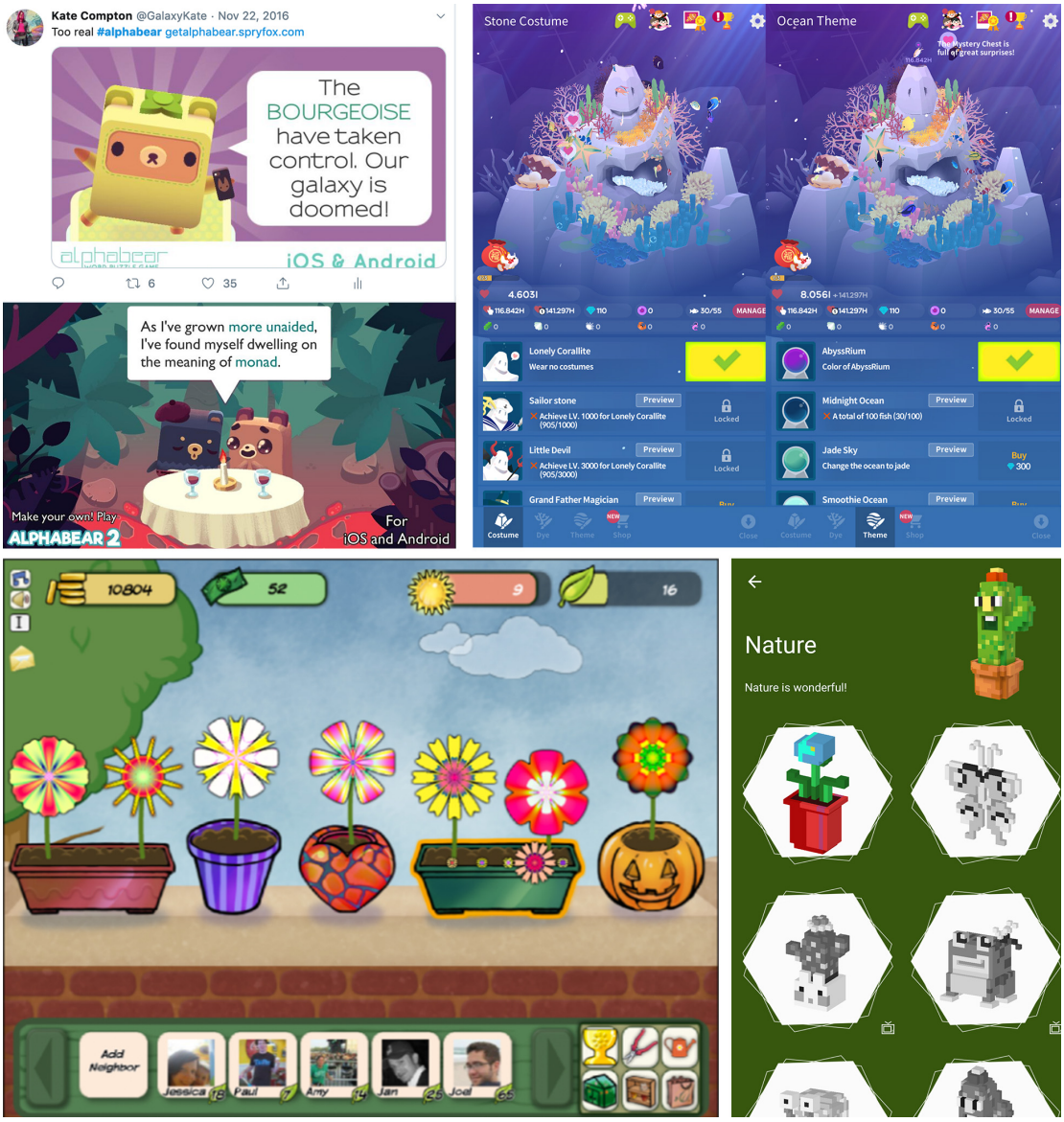


Figure 18.1: The casual creator embedded in the word-search game *Alphabear* lets users fill in templates with the words they found, and encourages them to post them to Twitter. Also shown: *AbyssRium* dress-up, which also has Twitter snapshot sharing, *Petalz* collectible pots[321], *Pixel.ly 3D* menu replacement

evaluations, can be part of a casual creator experience, with positive effects. It *may* be that what this concept means in casual creators is *absence of being judged in a serious way*. It may not feel bad to be judged as a bad artist in a domain that is intentionally impossible to make bad art in, or where the blame is shared.

Pictionary[s170] is a board game where players compete by drawing. This is already somewhat constrained to level the playing field (players aren't judged on quality, but ability to convey hard-to-draw concepts), but there are still haves and have-nots in Pictionary. Several games inspired by *Pictionary* further level the playing field by making the exercise *sillier* and *more impossible to win through skill* (an "impossible task" pattern that can be applied to many casual creators). *Cranium*[s38], a 1998 board game, has "creative" challenges where players must draw but with their eyes closed, or use clay to sculpt their answer. In *Tee K. O.*[s230], a creativity game in the *Jackbox* party game collection, players draw graphics for t-shirts on their phones, then write captions. Then the graphics and captions are distributed to the players, who much construct good pairings. An exceptional drawing is less desirable than one that perfectly juxtaposes with a caption, so all players have an equal chance to be used in a winning design, and the credit (or blame) is shared with the one who wrote the caption and the one who picked the pairing.

Another example of diffusion of blame in a stressful situation is Dietrich Squinkifer's *Coffee: a Misunderstanding*[s35], a computer-guided group improv performance. As a piece about the terrible social discomfort of two people meeting, most people would dread having to perform incredibly awkward improv in front of an audience. But like *Tee K. O.*, the creative risk is diffused among several players: one pair who select the next lines for the couple, one pair who perform the lines (but have no choice in what they are saying), and an audience who can goad the selectors into picking the most embarrassing possible content. Creative awkwardness happens, but no one person bears the full weight of creative risk. Just as *Mario Party*[286] is a series of party games where player skill is uncorrelated with victory, so too are there casual creators that provide a space where

players can win (or lose) creatively without feeling meaningful judgment[3].

How we judge a user seems to be more important than *whether* they are judged. This dissertation wouldn't be complete without a reflection on one of the most repeated anecdotes about creativity, from the popular book *Art and Fear*.

> The ceramics teacher announced on opening day that he was dividing the class into two groups. All those on the left side of the studio, he said, would be graded solely on the quantity of work they produced, all those on the right solely on its quality. His procedure was simple: on the final day of class he would bring in his bathroom scales and weigh the work of the "quantity" group: fifty pound of pots rated an "A", forty pounds a "B", and so on. Those being graded on "quality", however, needed to produce only one pot – albeit a perfect one – to get an "A". Well, came grading time and a curious fact emerged: the works of highest quality were all produced by the group being graded for quantity. It seems that while the "quantity" group was busily churning out piles of work – and learning from their mistakes – the "quality" group had sat theorizing about perfection, and in the end had little more to show for their efforts than grandiose theories and a pile of dead clay[426].

Will Wright frequently used this anecdote (without attribution) in his presentations about *Spore*[315], as an explanation of why rapidly iterating on creatures (with a fast grokloop) leads to confident capability in the *Spore Creature Creator*[s216]. It is unclear from the book whether this anecdote is a real study that happened, something that was observed, or just a parable. However, it is repeated because it makes *intuitive* sense. We can picture the students of the first group, clenching over their perfect pots, and the students of the second, rapidly making terrible pots, freed from their socially-manufactured allegiance to perfection by an orthogonal judgment mechanic.

Casual creators have an easier-than-usual task for implementing "psychological freedom". As long as the app is not live-broadcasting everything that the user is

[3] *Cards Against Humanity*[missing system:cah], a "party game for horrible people" enables both creative and ethical diffusion by allowing players to blame their limited options and random draw for making socially-unacceptable combinations.



Figure 18.2: Creative risk-diffusion in *Tee K. O.* and *Coffee: a Misunderstanding*'s puppets/puppeteers/director/audience setup (not shown: awkward laughter).

doing to a critical audience[4], then they are in a space free of meaningful judgment, in which all of the controls are *meant* to be used, and all of the possibility space is *meant* to be explored. Casual creators are a *constructed* environment; users know this, and so know that all the UI was intended for their use and safety in the same way that a child on a playground knows that play activity that can be performed on that structure should be "fair game". It is up to the playground designer, and the casual creator designer, to ensure that any activity in that space should be accessible, fun, and non-harmful.

This dissertation has talked to no end about what kind of artifacts possibility spaces can contain (chapter 9). Possibility spaces can have many different distributions of artifact-quality: some spaces have only blandly acceptable content, others have mostly bad or broken content with a few gems. Chaim Gingold, in his 2007 GDC talk on the *Spore*[s215s216] editor, and his concept of "Magic Crayons"[158], drew the possibility space of the *Spore* vehicle editor. Some parts of the space were good, bad, okay, or desirable-but unreachable. The goal, he said, was for the designer to tune the space so that the possibility space overlapped as much as possible with the desirable ring, while still having enough focus to be able to

[4]Twitch streaming, e.g., but there will *eventually* be a casual creator that tries to make this work, I guarantee it.

tune out bad content.

"If you're making a screwdriver editor, if you only know that your players are going to be making screwdrivers in the editor you can probably make and editor that you can *only* make good screwdrivers in. If you expand that domain to be something bigger, you're trading off breadth for quality. So if you say 'I'm gonna make the tool that lets you make the screwdriver and a boxcutter tool', then you're going to end up with less quality stuff. And that's the trade-off between breadth and depth"

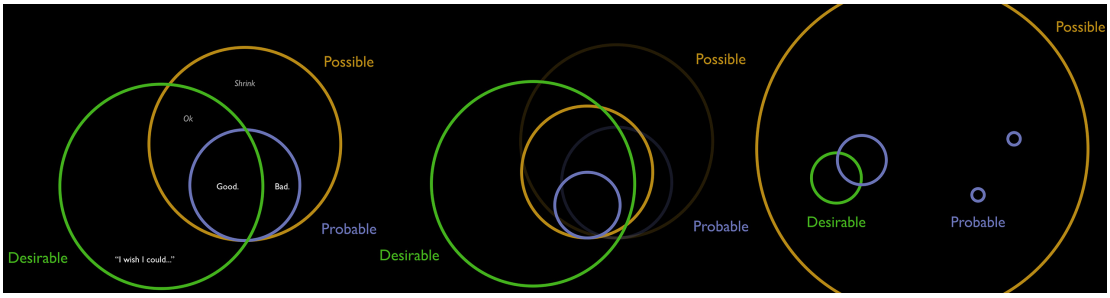


Figure 18.3: Gingold's illustration of the *Spore* vehicle editor's space during development (left), how they tried to shift it (center), and comparing it to the space of a tool like *Maya*[s132] (right).

This is *one* way to design a *safe* possibility space: to optimize the amount (and coverage) of good content, while minimizing bad content. But there are also patterns that we can use to support an environment of creative safety in systems that still have bad artifacts, either accidentally or in exchange for a larger range in order to have richer artifacts in the possibility space.

- shopping/lookahead: let the user browse content faster, by showing them many options in the local space, or landmarks in far away parts of the space that would be too dangerous (rife with low-quality artifacts) to jump to if they didn't have a preview.

- space annotation: annotate and augment the space, so the user can orient

themselves to regions or landmarks

- create warnings: if there are parts of the space that are high-risk, let the user know. Examples of this are "green-yellow-red" danger meters (like the *Spore Creature Creator*[s216]) or sliders that become resistant or harder to pull as they get to dangerous extremes.

- the ability to undo, bookmark content, save multiple options, and reroll artifacts without penalty

The last one seems obvious, but in games with generative content, the new content is only generated very rarely, and with no user control. This can be as a way to hide the possibility space from the player (dealing out only one artifact at a time can make them imagine a larger space than is there), or use randomness as a gameplay limitation, preventing them from rerolling or editing a random artifact for better gameplay advantage. If the only way to explore the generator's possibility space is through playing the game, the logic goes, the player will be more motivated to play the game to gain the new peeks at the space. This is a reasonable conclusion (I spent years addicted to the randomly-generated weapons of *Diablo II*[s48]). But it can be a high-risk environment for creativity. If the user has to earn the right to edit or undo, they will be very conservative in their creative moves, as each one *has to count*. I found myself falling into this play style when I was playing through many of the trashier casual creators on the app store (spammy apps marketed to children). They either had gratingly slow grind-mechanics to pad gameplay time, or 30-sec long required ad-viewing (often both), before allowing a few edits to the artifact. Changing modes or tools (even undo!) would often trigger another round of unskippable ads.

The ability to "undo" is a fundamental piece of casual creator safety. Its absence can be part of the "impossible-task" safety pattern above, or as encour-

agement to get users to keep moving forward and not fuss (like the ceramics anecdote) but this is a risky move and not generally to be recommended. "Undo" itself can mean many different things in an interface[162][6] and be implemented in many technical ways. Most commonly, it is a stack of operations, and a single "undo" command to pop the last operation and "re-render" the stack. Sometimes the stack is an array, and a "redo" command can move forward in the array of operations, unless it has been overwritten with new operations. This is a computer-science definition of "undo", but it may or may not be what the user needs for safety. One of the most invisible forms of "undo" is a slider. A slider is a wonderful bundle of safety features. It contains upper and lower limits, and implicitly tells the user that anything in between is quite alright[5]! A slider also has a short-term "undo"/"redo" built-in: unless the slider is very small or fine-grained, the user remembers about where their hand was a moment ago, and can navigate back to a previous position without having to change mental context to a different piece of UI to undo. 2D sliders (e.g. color-pickers) can have this same property.

The single history of undo/redo can be expanded in various ways[133], allowing the user to bookmark multiple pasts and futures, although this can become arbitrarily complex, like *Photoshop*'s[s168] recent ability to control history with a layer mask, or impossible to understand, like Git branching. A casual creator should steer away from anything that requires the user to perform a multi-branch Git merge, but there are certainly ways to play with new "undo" formulations.

[5]I remember hearing from a Sims designer that players complained about bad behavior from the Sims, because their sliders were set to the extreme low ends of 'cleanliness', because the users didn't perceive the bottom of the slider to be an extreme value, but a default.

Chapter 19

Pattern - Possibility Space Annotation

As we have seen from chapters chapter 9 and chapter 11, understanding and navigating the possibility space is a key component of a casual creator, but not a well-studied one. In "Search-based procedural content generation: A taxonomy and survey"[411], Togelius et al dissect and document generators which use some form of *search algorithms* to find good artifacts in a possibility space.

They describe content as being:

- created online or offline ("offline" here meaning that it is fixed as the game is run, not generated on the fly)
- necessary vs optional (note that this doesn't include "decorative" content which they rule out as not "PCG")
- generated from random seeds vs parameter vectors
- stochastic vs deterministic generation (whether, absent a seed value, a piece of content generates the same each time)

- constructive vs generate-and-test

In this paper, they are primarily concerned with "generate-and-test", which they define as an algorithm where, "after a candidate content instance is generated, it is tested according to some criteria". Some forms of search that they recognize are "exhaustive search", "evolutionary algorithms" and "all forms of heuristic and stochastic search/optimization algorithms". For these forms of search, they recognize that "in order to search the space effectively, the vector should have the right dimensionality", that is, it must create a possibility space that is large enough to hold interesting artifacts, but small enough to be meaningfully searchable. Also necessary to many forms of hill-climbing search is "locality", the idea that small changes in the artifact's model (or genotype) should produce correspondingly small changes in the artifact or the experience of the artifact (phenotype)[1]. They also list three kinds of heuristic functions that can direct an optimization's movement through the space:

- direct evaluation functions: properties calculated from the artifact/phenotype itself
- simulation-based evaluation functions: some simulation of the artifact's experiential properties are evaluated (for example, a simulated play-test in a game, or a physics simulation for antenna signal strength)
- interactive evaluation functions: in which users (or players) implicitly (through attention or interaction) or explicitly (through survey) approve or disapprove of content

Eventually they ask "Can we combine interactive and theory-driven evaluation functions?" The answer is yes. As shown in the chapter 16 chapter, users can be

[1]See the chapter on Parametric Spaces for translations between the biological language of genotype/phenotype and the model/artifact/experience language I use in this dissertation.

presented with a range of evaluations for an artifact, and seem to hold them all manageably (even for fictional or entertaining evaluations).

Not addressed in this paper is **how** a user moves through the space, and indeed, from all the examples, the users have no way of moving through the space at all. Rather, the common pattern among their cited generators is that users are simply presented with new content on restarting or replaying a level or game, with only the very slow and indirect "control" of possibly providing their implicit choice data as a feedback mechanism for the next play-through. For example, in Galactic Arms Race, the players choose and use evolved weapons, which feeds back into a global ranking system to spawn new weapons. If they like and use a weapon, that preference will filter though this global system, to the next player in the world getting a weapon descended from it, and then re-propagating *their* preference. It might be weeks before the world changes noticeably, through the actions of thousands of players, which is a far cry from Direct Manipulation[2] or instantaneous feedback grokloop!

Games where players can't directly modify the generative artifacts may *seem* like not much of a casual creator ideal. But if any given generative space is fixed (by the limited range of any algorithms used to define it), then "creating" and "finding" artifacts in it are two sides of the same coin, or two search methods to achieve the same aim. If **we imagine the user is a search method** then they operate like the methods that Togelius et al are examining. You can have humans-as-search as:

- exhaustive search: incrementally searching a space seed-by-seed
- random search: randomly selecting a seed (or using seeds with low locality)

[2]chapter 8

- hill-climbing search: looking at nearby artifacts, and using that to choose which direction to explore

- look-ahead search: the human uses some advanced knowledge to guess at which model parameters to tweak to get to a better artifact

The first two can be done with zero-locality seed-only random generators (i.e., where the seed is expanded to more complex model structures automatically), the third requires good locality for seed-to-model translation, and the fourth requires direct user control over the model. Unfortunately, nearly *all* generative content in games has very low locality and provides no control, except over seeds. Even that level of control is only granted in a few generators.

However, we can see *many* communities that spawn around generative content in particular games. Why do these communities form? What reward do these users get from mapping and exploring a generative space as human search engines?

We have established that generative spaces are one critical feature of a casual creator. Now we can analyze this phenomena to see the way that users can experience a generative space by exploration alone, *orthogonally* to how they control it.

19.1 Automatically mapping generators

When we build generators, it is often helpful to build tools that allow us (as programmers) to see, analyze, and track changes in the generator's output. Gillian Smith's *expressive range* analysis[383] and Michael Cook's *Danesh*[s41][105] tool are two examples of stochastically generating *many* artifacts, and then visualizing the output to create 2D graphs, mapping which combinations of properties appear in the generator an which are absent.

> "Danesh is a tool to help people work more easily with procedural generators. With just a few lines of code you can connect your generator to Danesh and get a better understanding of what it can do, what it can't do, and what it might do with a few tweaks. (...) Danesh is designed to help people learning procedural generation for the first time, as well as experts who want to get the most out of the code they write." [103]

Rather than simply graphing density along two axes, the *Petalz* project[s167][323] created a clustering algorithm to sort its immense space of potential generative artifacts into 80 categories, based on their phenotypic (or artifact/experience) properties.

In *Petalz*, users create flowers from a compositional pattern producing network, an algorithm that combines functions using a hidden network of weights (comprising the "DNA" of the flower). The users perform hill-climbing search (*Petalz* has one of the few high-locality algorithms in this chapter) by breeding together different flowers. However, the game needed goals or quests to be more interesting to potential users, so they generated the 80 flower species such that any given flower in the space would correspond to one species of flower. This partitioned the space into regions, and gave the users a challenge and a target for their hill-climbing attempts.



Figure 19.1: The 80 machine-learned categories of Petalz flowers (from [323]).

When this is insufficient (many valuable properties cannot be machine-calculated, including aesthetic properties), sometimes other forms of mapping can be used. One example is the automated cameras set up by the No Mans Sky team, "to fly to each of the game's 18 quintillion worlds, take short videos and document its interstellar travels as a series of animated GIFs"[400]. These virtual trap cameras could give the developers a trackable, random, and hopefully representative view of what the current iteration of the generators would produce. Like *Danesh* and expressive range analysis, this was not a tool for users to map the space, but for the designers to understand the current generative space **as an artifact of a greater generative space that they were moving through by tweaking their generator**. Casual creator patterns can also be applied to navigating *this* space as well.

Some fully-automated algorithms can be used to map generative spaces. However, it is also possible to combine human heuristics with algorithmic annotations to drive space-mapping. In Stanford's *Dryad* project[s56][401], parametric trees are generated with a vector-shaped model (some number tuning parameters) and an Lindenmayer-system branching structure generator. They created an editor that allows users to slide these parameters to explore new trees and save good ones. Of all the *many* generators analyzed in this dissertation, this was perhaps the only one to explicitly declare that there *were* non-model-based structures in the generative space. Not only that, but the users could identify good artifacts, and that this could be used to **map** the space:

> "The first key idea behind our system is that we learn properties of the space of good designs from the collection of specific models selected by users. Our conjecture is that this space has structure, and that good designs are not just random collections of parameters. We learn this underlying structure using density estimation, a technique common in machine learning and computational statistics. (...) The second key idea is that we can greatly improve the modeling process itself by

leveraging the structure learned from the community."[401]

In fact, they were able to map the space, according to the user input, and could use that to visualize what *random* trees looked like, compared to trees in *high-density* areas of the map, thus using the output of the mapping algorithm to guide generation.

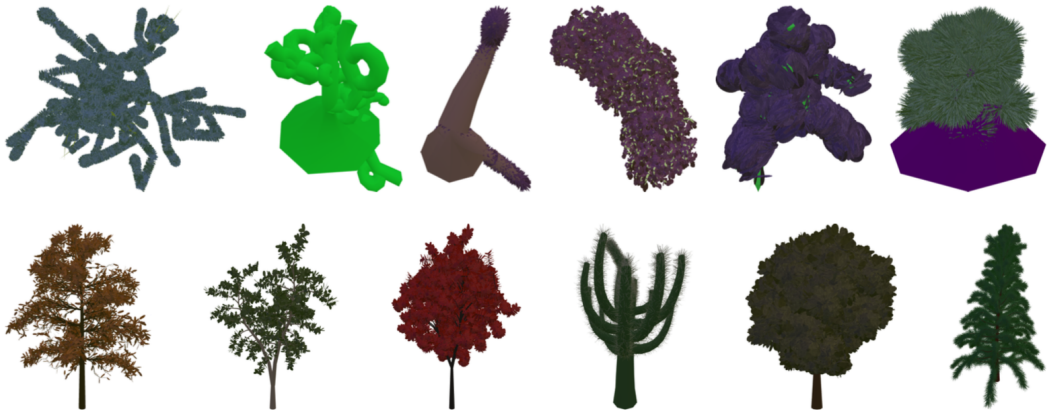


Figure 19.2: On the top row, artifacts randomly selected from the whole tree space. On the bottom, artifacts randomly selected from *high density* tree space, which are more reflective of real-world trees (from [401]).

19.2 Social cataloging

In all these cases, the algorithm (including human-driven algorithms) performing the mapping is used in service to the people building the generative system. However, I also want to explore the many ways that building this map **is an experience in itself**.

Moreover, in every instance of this phenomena that I've found, map-building is also a profoundly *social* activity[3].

[3]This may be a side effect of the visible nature of social play vs individual play. People making very personal, individual ontologies may not post them online, so we only see the multi-person social efforts

Games, and generative spaces, often have many emergent properties, properties which may not be visible in the explicit text of the rules themselves. Common themes, common dynamics, and overarching dynamic structures may only be detected by groups of players coming together to discuss (and play) games. Folk languages spring up around these practices to give players ways of communicating with each other, and more common concepts will be calcified into terms, like "turtling" (a term common in strategy games: building up defensive ability, at the expense of mobility or power). Games with lots of rules and large established communities will create entire folk ontologies, such as these examples of metagame knowledge around *Magic: The Gathering*[s142][2]

> "**Creatureball:** A creature with an X in its cost determining how big it's going to be. Mostly green. Named in analog to Fireball."
>
> "**Engine:** A card or combination of cards which produce a powerful, often repeatable, effect, which does not win the game outright on its own, but is effective in "powering" other strategies. An engine often converts one resource into another. For example, Channel converts life points to mana, Mind Over Matter converts cards in hand to untaps of permanents, Skullclamp converts small creatures into cards in hand, and so on. Engines often form the heart of combos and are often restricted in tournaments due to being too effective."

We can deduce from the existence of these terms that "an X in its cost determining how big it's going to be" is a common property of these cards, common enough to have developed its own term, as well as "Fireball" being a particularly well-known or early example. Likewise, we know that it is useful to consider "engines" as a gameplay mechanic: "powerful, often repeatable, effect(s)" are central to the game, cards will have value based on whether they fit into engines, and there are "engines" that are well-known enough to have names and known properties.

Magic:The Gathering is somewhat generative, in the way that any random hand-drawing game can be. But a recently released game, *KeyForge*[s1124] is inten-

[4] `https://www.keyforgegame.com/`

tionally more generative. Each deck has a generated name and a generated set of cards that are guaranteed to be playable out of the box:

> 'Every Archon Deck contains a full play experience with a deck that cannot be altered, meaning it's ready to play right out of the box. Not only does this remove the need for deckbuilding or boosters, it also creates a new form of gameplay with innovative mechanics that challenges you to use every card in your deck to find the strongest and most cunning combinations. It is not the cards themselves that are powerful, but rather the interactions between them—interactions that can only be found in your deck. Your ability to make tough tactical decisions will determine your success as you and your opponent trade blows in clashes that can shift in an instant!'[153]

The generative names are interesting and memorable, such as "Axiomatically Muscled Ken" "Justworth, the Jailer of Nerds", "The Doctor Who Replicates Owls"[5], and sometimes offensive to the point of being banned from play[427].

Given a *Magic: the Gathering* style game with *entirely new properties*, players immediately began mapping and cataloging the cards[6], as well as the deck constructions: "Took the info from multiple online spreadsheets of card and deck lists. Here is a breakdown of card rarity and card types per deck and house."[7]

This is not unexpected, considering what we've seen with the history of *Magic: the Gathering*. But the publishers of *KeyForge* also require users to register their decks on the official site before allowing them to play in tournaments. The tournaments will then be used to map the "winningness" of each deck live on the webpage: "See how your Archon Decks fit into the meta at large and follow the exploits of the most powerful and cunning Archons of the Crucible."[8]

Other games with generative artifacts lack developer-given or automatically

[5] https://boardgamegeek.com/thread/2098870/what-your-best-deck-name

[6] https://docs.google.com/spreadsheets/d/1p81uugi4N4cQGU8QRFQXiZ-hU39zO8ZfcSCeVqwXzIY/edit#gid=0

[7] https://www.reddit.com/r/KeyforgeGame/comments/970a7d/some_analysis_based_on_46_complete_decklists/

[8] https://www.keyforgegame.com/explore

				Deck Name							
				Benegoblin, the "Alien" of Knowledge							
Set	Num	Name	Rarity	Type	House	Traits	Æ	Power	Armor	Abilities	Flavor
House Logos											
CotA	112	Interdimensional Graft	U	Action	Logos		1			Play: If an opponent forges a key on their next turn, they must giveyou their remaning [Æ].	
CotA	114	Labwork	C	Action	Logos		1			Play: Archive a card.	Attention to detail is the key to all progress.
CotA	115	Library Access	C	Action	Logos					Play: For the remainder of the turn, each time you play another card, draw a card.	
CotA	116	Neuro Syphon	U	Action	Logos		1			Play: If your opponent has more [Æ] than you, steal 1[Æ] and draw a card.	
CotA	123	Sloppy Labwork	U	Action	Logos		1			Play: Archive a card. Discard a card	Don't fret the details. Progress happens when you least expect it.
CotA	129	Library of Babble	C	Artifact	Logos	Location				Action: Draw a card	When you make an archive of all knowledge in the galaxy, don't be suprised when you understand almost none of it.
CotA	137	Brain Eater	U	Creature	Logos	Cyborg * Beast		6	~	After a creature is destroyed fighting Brain Eater, draw a card.	
CotA	139	Doc Bookton	C	Creature	Logos	Human * Scientist		5	~	Reap: Draw a card.	"Don't worry, Momo. We'll have this quantum death ray installed in no time."
CotA	140	Dr. Escotera	C	Creature	Logos	Cyborg * Scientist		4	~	Play: Gain 1[Æ] for each forged key your opponent has.	"Interesting reaction, but what does it mean?"
CotA	144	Quixo the "Adventurer"	C	Creature	Logos	Human * Scientist		3	~	Skirmish. (When you use this creature to fight, it is dealt no damage in return.) Fight: Draw a card.	"...I'll leave this part out of the memoir."
CotA	156	Veylan Analyst	U	Creature	Logos	Cyborg * Scientist		2	~	Each time you use an artifact, gain 1 [Æ].	Logos are divided into two camps: Theorists and Mechanists. Each believes themselves to be superior to the other.

Figure 19.3: Keyforge deck composition.

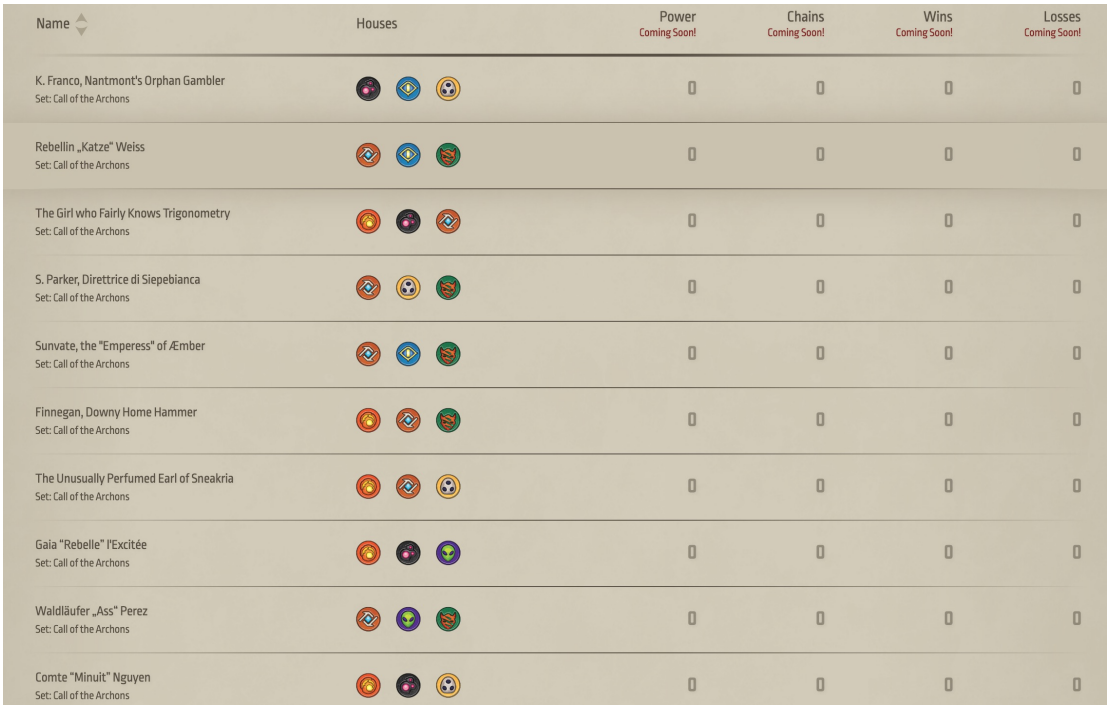


Figure 19.4

detectable properties. For these games, the folk language and folk ontologies are necessary for even beginning to describe the different regions of the generative

possibility space and how it affects gameplay. *Galactic Arms Race*[185], by the same research group who made *Petalz*, using a related algorithm, uses compositional pattern producing networks to generate bullet patterns for a massively multiplayer shoot-em-up game. How can players talk about strategy in a space where there is a massive, smooth, possibility space of different weapons? Like *Petalz*, these weapons cluster into a number of species with internal variation, but unlike *Petalz*, these clusters were developed collaboratively by a community of competitive players. One player advises new community members, using the custom folk-language developed the community to define the weapon categories:

> "But here is a list of Isotopes I recommend all newbies: Triple "V"-Pattern isotope: 3 projectiles fire in a "V" pattern, allowing them to hit more enemies at the same time. still a great and defensive weapon. Wallmaker isotope: Wallmaker isotopes are projectiles which stop after a random distance and create a wall of projectiles, able to block enemy projectiles or dealing damage if enemies rush into them. Recommend for defensive players. One big projectile: 3 Projectiles in one single. This one is not much more different to the starter gun, but still good to kill space blobs or xindo elite.Crazy moving projectiles: Random pattern projectiles which fly around like crazy. Defensive + offensive gun, recommend for close combat."[144]

None of these weapons were named by the developers, as the weapons didn't exist until they were discovered by players. But players need a shared language to talk about their games, so they had to create their own based on their understanding of the possibility space.

Seed catalogs

Perhaps the most common form of collaborative mapping is in what I'm calling "Seed Catalogs". In this form of community exercise, users of a generative system that takes a seed-value as input (like players of a game with generative levels) post seed values that they have used in the generator as well as the properties they've

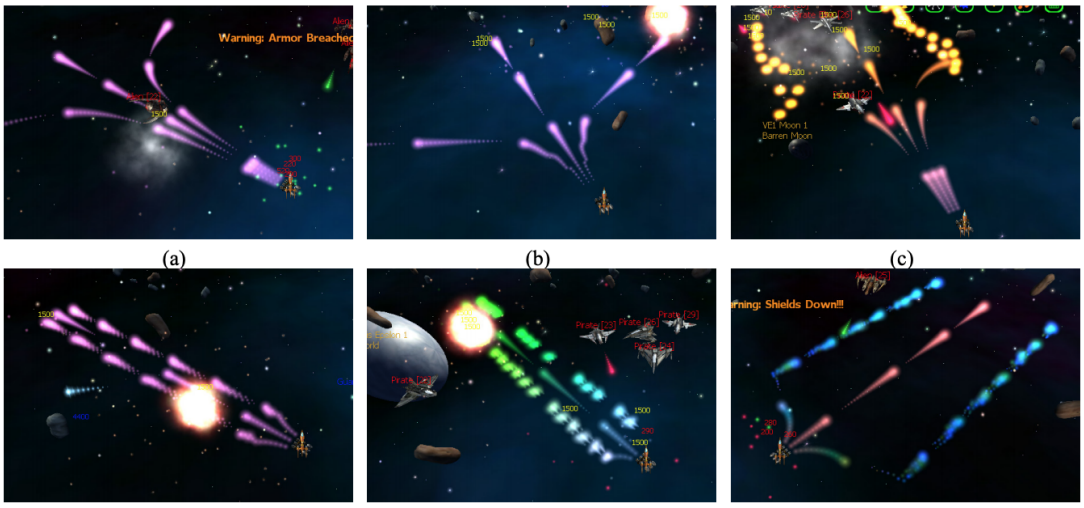


Figure 19.5: A variety of *Galactic Arms Race* bullets (from [185]).

documented for the resulting artifact. A fascinating early version of this occurred with the free Microsoft Windows game *Freecell*[s75]. Unusually for a card game, this game let you input seed values for the shuffling of the deck and resulting layout. This feature, and its limited input values (32,000, much smaller than the actual number of shuffled decks in the real world), made possible an *exhaustive* search of the *entire* possibility space:

> "In August 1994, Ring solicited volunteers on rec.puzzles, sci.math, and elsewhere (eventually getting more than 100 people involved), and coordinated the volunteers in an effort to solve all 32,000 of the deals in the Microsoft versions. He assigned each volunteer a set of 100 consecutive deals, and the volunteers would report back after they had solved (or tried to solve) all 100, when they would be assigned another set if interested."[218]

Eventually, through coordinated effort, they mapped the entire space, and determined that all but one of the seeds were solvable.

Freecell had a limited number of properties that players were interested (mostly solvability, as well as the number of solutions as a proxy for difficulty). Other games have a variety of different gameplay needs, so as games become more

open-ended, the information needed as documentation of a seed grows. In the space colony simulator *RimWorld*[s189], players must gather resources, farm, explore strange areas, and survive attack. So, the forum where seed values are collected and requested has a variety of differently structured ways of talking about levels[98]. Some are seeking particular strategic advantages, or themes, while others want a particular experience:

> "[Request] Looking for a rain forest mountain seed that's easily defendable without any caves" "looking for a map that throws strategy out the window in the name of cool features." "[REQUEST] Looking for a nice Cold Bog seed to do a tribal run"

Players have also developed a rich folk ontology for documenting explored seeds:

> "[1.0][FLAT][250x250][seed: worD][25.23S, 15.95W][50% coverage][50/60 growing]Ancient Road/Coast/Large River Marble/Sandstone/Granite" "[1.0][MTN][DEF][275x275][seed:bowler hat][13.04N, 42.96W][30% rain] Massive, beautiful protected valley" "[1.0] [Huge river, coastal mountains, boreal forest] [Seed: pumpkin] [54.06 N, 32.74W] [275x275] [20/60 grow season]"

Note that the seeds are not random numbers, but words. This provides a *massive* possible seed space (no exhaustive search here), but makes it easier for users to talk about different seeds.

Minecraft[s136], known for its open world and range of user experiences (from survival to architecture to exploration) has spawned lists of seeds that read more like travelogues from a travel writer encouraging you to visit:

> "One for the Minecraft mountaineer, this seed will spawn you at cloud level on this amazing mountain structure. It goes much higher into the sky than the clouds, and its flatter sections are home to some savanna biome-style trees. The whole mountain makes an island, with some really cool geography to explore." "This seed sells itself on an interesting premise: two small islands, one with a wolf on it and the

other with a sheep. From that point on it's up to you how this survival story unfolds – do you harvest your only mammalian friends for resources and meat or do you do everything possible to ensure that your unconventional companions are safe?"[9]

In the space exploration game *Starbound*[s218], players visit a range of planets. While they don't enter individual seeds, each planet has a coordinate that can be used to locate it in game, serving the same purpose. There is a lively catalog of visitable planets[257][10]. Any individual planet can be upvoted by users, letting them bubble-up more interesting or feature-rich planets, and can add notes and walkthroughs for those planets. *Elite:Dangerous*[s63], another space-trading sim, has a comprehensive online database, where users document solar systems, space stations, factions, commodities, and emergent economic and trade properties of the generated systems[189]

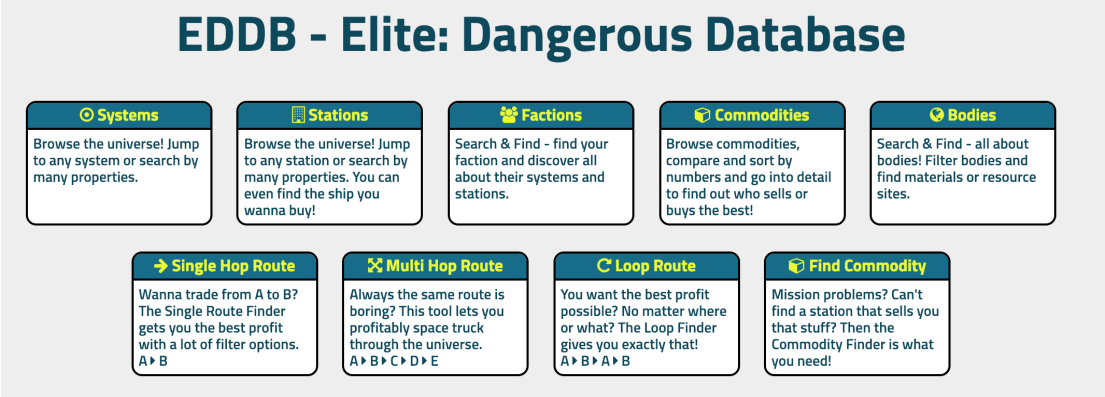


Figure 19.6: The Elite Dangerous Database catalogs planets and content, but also metagaming patterns that emerge from them, like arbitrage loops (from (`https://eddb.io/`)).

Some games cannot be mapped by seeds, especially if they don't use seeds, or have a space that is simply too big to be mapped. Orteil's *Nested*[11s147] is a single-

[9] `https://www.pcgamesn.com/minecraft/30-best-minecraft-seeds`

[10] Description on the site by the maintainer, "I'm developing this site because I love Starbound and I wanted to give something back to, the wonderful community surrounding this game."

[11] `http://orteil.dashnet.org/nested`

seed universe with recursively nested galaxies, planets, villages, minds, thoughts, subatomic particles, and the universes *inside* the subatomic particles. Because it is infinitely nestable and grows exponentially with each layer that is expanded, addresses are not very useful, nor is individual documentation of items practical. Instead, users document the *structure* of the recursive components[292], listing ones like "Lasagnaverse" and "black hole" ("In the tag is the archetype called inside the black hole which inside that archetype has a white hole. It also has a 0.5% chance to have an 'end of universe note' and a 0.2% chance to contain a crustacean.")



Figure 19.7: A medieval villager's thoughts in a galaxy somewhere in Orteil's *Nested* (`http://orteil.dashnet.org/nested`), and the villager specification, as documented in the Wikia for Nested[292].

19.3 In-game mapping

Many games allow users to catalog found objects in game, but these are usually for gathering pre-authored collectibles (*Petalz* is a rare variant of this that is

also generative). In No Man's Sky, all players share a single universe, but each player can name planets and creatures that they come across, and these names can be shared with the wider multiverse. As any significantly large universe will spontaneously generate recreational armchair bureaucrats, there are at least two recommendations of how players can standardize the naming scheme of their creatures in order to better help other players map the generative space. One is traditionally Linnaean[339], the other one is specifically based on the algorithms used to generate the creatures and intended to create a smooth (high locality) mapping between name and artifact:

> "This method of species classification depends on the creature's physical appearance and not on the randomized name given by the game. Fauna that have all the same body parts may still have different skin textures and colors. Color and texture is not accounted for in this method so it does not affect family name. Ears are not currently used with this method, however may be considered accessories.
>
> Rules of Lamb's Method
>
> - Use the family, genus, then species ordering.
> - Family must be named using Lamb's Method.
> - Genus must be named using the games pre-made genera.
> - Species may be any unique word, name, or phrase.
> - To get the family name, use the specified letters given to various body parts of that genus. These can be found on the page of the genus.
> - To get the family name, use the order: Body - Head - Arms&Legs - Tails - Accessories.
> - If an animal has different back and front legs, use the letter for the front legs first and then the letter for the back legs.
> - If an animal has more than one accessory, place the letters for those accessories in alphabetical order within the name."[97]

Oddly, No Man's Sky has all of this information, but there is no centralized place to see it. One player-run server (now defunct)https[12], at least collected some

[12] `url`

of the player-uploaded creature naming, allowing players to see who had found and named more creatures. Additional leaderboards are also user-hosted for maintaining size records[13]. As we will see in the next chapter, allowing users to self-promote or gain fame through interacting with a generative space, or casual creator, can be immensely motivating. The next chapterchapter 20 will demonstrate some patterns that can be implemented in-game and out of game, to celebrate user creativity while also promoting the generative system that enables it.

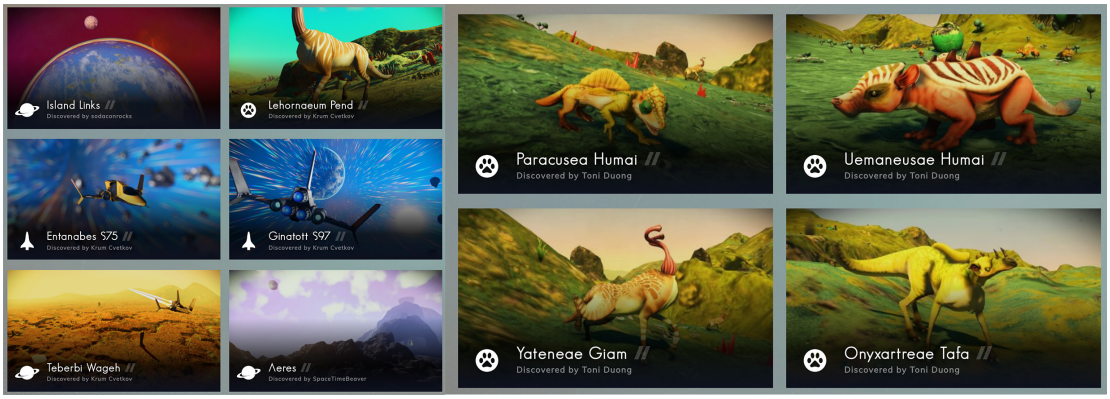


Figure 19.8: Images from the now defunct Athena DataBase of named *No Man's Sky* entities.

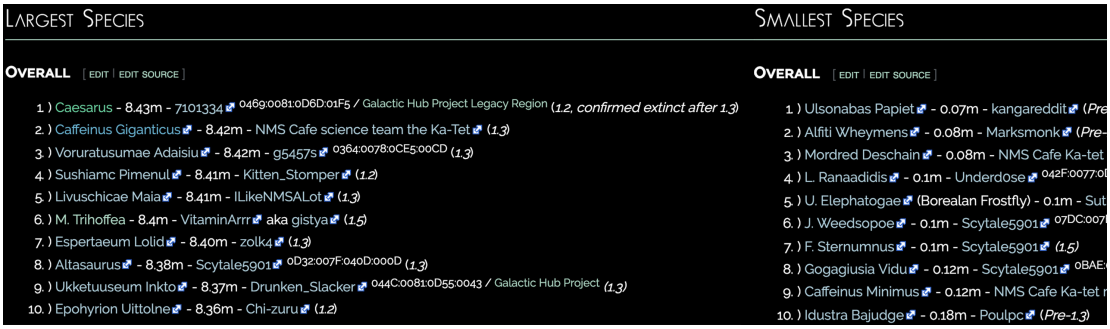


Figure 19.9: Smallest and largest creatures found by users.

The Lindenmayer-system flower-collecting game *Mendel* also encourages the player to collect and name generative content, in a single-player context. Players

[13] `https://nomanssky.gamepedia.com/Fauna_Hall_of_Fame`

can also breed plants together in a rudimentary genetic algorithm (as seen in chapter 11). *Mendel* tracks the genetic distance to the closest known relative, which it can do thanks to its parametric mapping. However, it does not have any way of tracking progression through the space, with either synthetic landmarks like *Petalz* or the dimension-reduction maps of *Dryad*, or even a force-directed graph showing the relationships of my current specimens. While I enjoyed the meditative experience, I found myself wishing I had a way to better orient myself in the generative space.

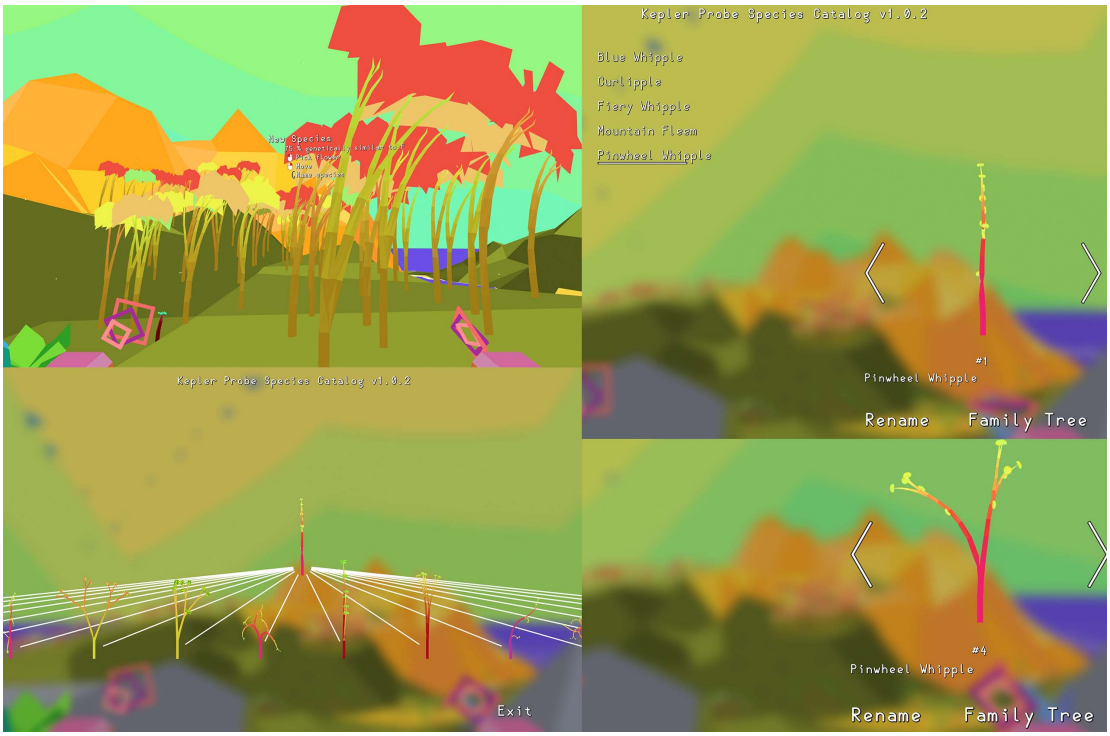


Figure 19.10: Collecting and cataloging plants in Mendel. You can see the family tree of any plants you have bred.

In generative systems, there is often a continuous spectrum between *discovering* a piece of generated content and *creating* it with an editor. Users may have no input in the generator except for hitting "reroll" (or performing whatever in-game action to get new content), and for systems with so little interaction, new

content is *discovered* in some latent space. Perhaps the user can have some input in the way they search (like in *Nested* or *Freecell*) or the amount of searching they perform (*No Man's Sky*). What is the difference between "creating" a flower by breeding it in *Petalz* or "discovering" it growing in *Mendel*? Annotating the space gives users a way to regain a feeling of "ownership" or skill in an interactive generative system where creative input is limited. In the next chapter, we will see more ways that users can self-promote their creative (or exploratory, or curatorial) skill through casual creators.

Chapter 20

Pattern - Self-promotion

> "Like any habit, creativity can either be encouraged or discouraged. The main things that promote the habit are (a) opportunities to engage in it, (b) encouragement when people avail themselves of these opportunities, and (c) rewards when people respond to such encouragement and think and behave creatively. One needs all three. Take away the opportunities, encouragement, or rewards, and one will take away the creativity. In this respect, creativity is no different from any other habit, good or bad." - Robert Sternberg [393]

While attending the "Convening for Open Source Tools for the Arts" in summer 2018, hosted by Golan Levin, several open-source creativity tool designers and I discussed the role of supercreators and community organizers in our projects. We often think of creativity tools (including casual creators) as being built by teams of programmers and designers. However, after that convening, I realized that users were not just passive consumers of fully-constructed systems, but were active contributors to the system by expanding and documenting the knowledge of the possibility space[79].

In the previous chapter 19, we saw how users could collectively map a possibility space. While many users enjoy becoming part of a collective project, many also enjoy gaining personal recognition through their work. Users may excel (or

contribute) in many different aspects of a casual creator or generative space:

- making high-quality artifacts
- making creative artifacts in response to prompts or memes
- making technically difficult artifacts
- performing the creation of artifacts on a stream
- creating tutorials or guides to creating artifacts
- providing templates or starting points for other artifacts

Users may also gain local fame for doing any of the above in sub-communities of interest (e.g. Spanish speakers, Harry Potter fans, furries). In performing any of these actions, the users gain the appreciation of peers or the community, and also contribute to the success of the casual creator or generative space. It is also important to remember that "creating" vs "discovering" is a spectrum, and any of these roles could be relabeled as "discovering" rather than "making" for systems (e.g. *Petalz*, *No Mans Sky*) where the framing and interaction of the system make that a more apt term. However, the social implications remain the same.

What *value* do the users gain from contributing? Depending on how the community and the systems are established, users can gain many different kinds of value from a contribution.

- Monetary value (asset stores, commissions)
- In-system bonuses (gameplay benefits, tool unlocks)
- Fame and social standing (followers, fans), in- and out-of-system
- Social connection (developing and investing in existing relationships)

- Competitive victory (leaderboards, contests)
- Compersion (seeing someone else happy, serving under-served groups)
- Satisfaction of a difficult job well done (technical difficulty or precision)
- Creativity (feeling that one has created a particularly novel solution)

By enabling these kinds of value-generation for users, systems can incentivize users to contribute in the ways above. Some forms must be explicitly enable inside the system, such as asset stores and followers, but others can exist either in a system, or via outside channels. For example, a Twitch streamer making custom *Minecraft*[s136] mods may receive payment via *Minecraft*'s asset store, or via Twitch donations, via Paypal, or by amassing a number of followers and using their channel for advertising.

In the forums for the *Petalz*[s167] game (described in the previous chapter)[s167], one can read the development of features over time, as the developers add features to support different values, such as unlockables and in-game rewards and "social-grooming" features (mutual care-taking to reinforce social bonds):

> "New features, new pots, new backgrounds! Come check it out!" (April 16, 2012)
>
> "Soon you will be able to see who contributed to your flowers and visit their balconies. Stay tuned!"

A new update for Petalz is out with lots of new features! (July 30, 2012)

Eventually Petalz developed more tool unlocks and in-game currency, as well as competitive leader boards and out-of-game social promotion tools (auto-posting pictures to the Facebook feed):

> "Now you can:
>
> - Post a screenshot of your whole balcony

- Buy cool x-mas pots!
- Boost your instagrow and coin rewards
- See how good you score on a global leaderboard
- And LOTS of other things!" (December 3, 2012)

By the time the game had run its course, it even had an in-game seed-sharing economy.

> "Adding to the fun, because Petalz is a social game you can share your creations with your Facebook friends. Simply add them as a neighbor and they can visit your balcony (where you can arrange your flowers however you like) and even water your flowers for you. Of course, you can visit your friends' balconies as well. You can even sell your unique seeds to other players through a marketplace that everyone playing the game can see, allowing new lineages to spread throughout the world."[135]

I use *Petalz* as a leading example, because it is a very small game with relatively few users, that as a limited-time academic project still managed a first-pass at nearly all the tiers of social features. In the rest of this chapter, I will go into more detail on properties of other casual creators that specialized in only a few of these modes of value-creation.

20.1 Social organization

Any time users have a persistent identity, that identity becomes the site of self-presentation, self promotion, reputation, and social identity. Any site with a username and avatar icon will soon discover the many ways that humans like to self represent in text and image (fandoms, identities, allegiances, and creative expression). Creative reputation can build up to the point where casual creators start developing "celebrity culture". On Spore, we could see which members of

the community were held in highest esteem, and we once brought our "celebrities" on an expenses-paid trip to the studio to test a new game[403].

When users have persistent creative identities, such as a fixed user name, they can begin forming larger creative structures. As seen in Chapterchapter 4, creativity is often an *ecosystem* which develops common structures like:

- friendships: persistent reciprocal communication between two users
- fans/followers: non-reciprocal one-way relationships where many users "follow" or are fans of a skilled user
- fame and reputation: the accrual of community acclaim, through follower count, likes or any form of reputation (prizes, awards, contest-wins, ranks)
- cliques: groups with organized boundaries, that regulate belonging to those groups
- movements: non-organized groups without fixed boundaries. Often these are joined informally, through adding tags to work or profiles

Not every platform has explicit support for these features. However, platforms *without* formal support will often see users hacking their way to that support, repurposing features and, when necessary, using text fields when code solutions are not available (such as adding formatted tags to plain-text string descriptions). In Spore, we gave users the ability to discuss specific creations: each creation had a threaded conversation that other users could comment on. However, we did not give users a way to directly message each other, so they quickly invented "mailboxes", creations that would serve as a general-purpose inbox for that user.



Figure 20.1: Mailboxes on the Sporepedia.

20.2 Famous explorers: from "merely-mapping" to "creativity"

We have seen that users can discover artifacts in the possibility space (chapter:spaceannotation), and that they can map and document those artifacts as

a form of service to a community (e.g the *Freecell* database from the previous chapter). When does that "discovery" become "creativity"? One way that this happens is with a persistent creative identity of the discoverer. As above, if a discovery is linked **to a persistent creative identity**, it becomes woven into the complex web of the creative ecosystem. If a Spore creation is built from a loaded creation from another user, that "lineage" is shown in its Sporepedia card. Likewise, when creature or planets are discovered in No Man's Sky, they are labeled with the identity of the discoverer, and the name that discoverer gave them. Users take advantage of that feature to express some humor and creativity that will then be perpetually linked to their creative identity[1]. "Naming rights" are a form of ownership, pride, and fame (not only has one discovered this, but named it well) and a motivation that can be seen in the plaques of natural history museums across the world (e.g. "Smithson's cat, collected by Lord Smithson").

"Interestingness mining" is another pattern, in which the raw possibility space is *so massive* that finding and recognizing an interesting artifact is in itself a creative endeavor. In these systems, creativity can be expressed in the "refinement" of raw "ore" of generative output into an interesting object to share with the community. Nowhere is this more evident than in *Dwarf Fortress*[s58], a game that manufacture massive piles of narrative-potential slag, which users seem to delight in mining through, digging out veins of narrative potential, and polishing them into a gem of a story than can be posted online. There is no shortage of users collecting *Dwarf Fortress* stories[2], telling their own long-winded adventures[208], and participating in long multi-user narrative collaborations. The most famous of these is Boatmurdered[434][3], a massive tale of a doomed fortress that has spawned fan art, an animation, and a narrated version. Even individual artifacts

[1] `https://twitter.com/Ondores_Lies/status/763011771955310593`
[2] `https://www.reddit.com/r/dwarffortress/comments/1ktz66/best_df_story/`
[3] `https://lparchive.org/Dwarf-Fortress-Boatmurdered/`

in a difficult-to-use experience can support quite large creative ecosystems, like a coral reef growing on a shipwreck in an otherwise barren ocean.

The usual approach to designing a casual creator is to make a generative space that is as safe (full of good quality artifacts) and easy to navigate as possible. Systems like *Dwarf Fortress* (and the more traditionally interactive *Caves of Qud*) showcase an interesting alternative approach that can be used in spaces that have the opposite property.

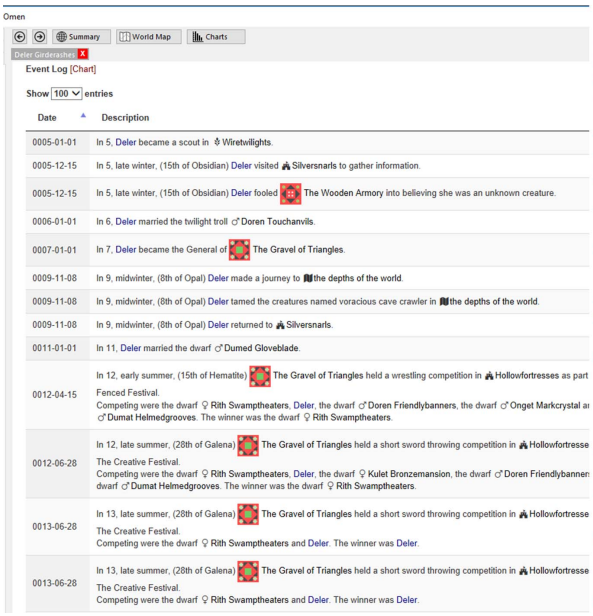
If we revisit the definition of a casual creator, we can see that these systems have *some* of the properties of a casual creator:

- privileges enjoyment of the creative process above productivity
- produces artifacts within a limited-yet-meaningful domain space, enabling automation and support, both passive (encoded into the domain model and system constraints) and active (responding to user actions)
- supports a state of creative flow by restricting choice and preventing hard failures while allowing rapid iteration
- results in the user's feeling of pride and ownership toward the produced artifact, and sense of pride in their own creativity.

They are an exploration of a generative space (of strange hyper-simulationist stories), with automation and support (the simulation of rules, as well as the improv-partner behavior of adding strange twists to a story). These games *are* still games, and can be played to win (or succeed), but they can also be played as "tragedy creators", where users are making choices (in game, or in the telling) to construct a beautifully tragic story to show off to an audience.

If this is true, then we should see some tools that players of this "tragedy creator" have created to speed up the grokloop for making and evaluating these

stories. And in fact, we do! *Legends Viewer*[s121] is a utility to explore exported *Dwarf Fortress games*, allowing users to browse around a frozen copy of all of the history, events, murders, and relationships in a fortress, which they then share with other users[91]. Jacob Garbe[146] has also proposed using story "scaffolds" (narratively-significant sequences) to search a large space for particular kinds of stories, such as vampire histories that follow a particular narrative arc, then translate them into natural language narration with grammars. While this system is not interactive (much less casual), we can see two pipelines (world-simulation to narratively-interesting event-sequences, and event-sequences to narration) that could be used for more playful explorations of the space.



In the year **121, as summer withered into fall**, trouble found **Rakust**. It was in **Painttarget**, where members of **The Humid Lance** accused **Rakust** of **murdering a local. Ironically, Rakust had not murdered anyone...in Painttarget, at least. Her** accuser stepped forward and **laughed unpleasantly. "To kill is a terrible crime."**
Rakust flicked their eyes to the **grumbling** crowd. **"What is that supposed to mean to me?"**
After muttered imprecations, the **locals dissipated**.
After that, **Rakust** decided discretion was needed, and so left **Painttarget** to settle in **Championgirders**.
For **a while things were quiet, but once again Rakust's past, or at least reputation, would come back to haunt them**. This was **4 years** later, **in spring, when the crocuses had just sprouted**.
Everything had been uneventful since she'd re-settled in **Championgirders**.
Then one day, there was a knock at the door. **Rakust** opened it, to reveal **most of The Fortification of Truth**, who immediately accused her of **murdering some nobody**. Ironically, **Rakust had not murdered anyone...in Championgirders, at least. Her** accuser stepped forward and **glowered. "We don't take kindly to murderers here."**
Rakust **swallowed carefully. "I'm sure I don't know what you're talking about."**
After **a few minutes of hot silence, the mob dispersed**.
After that, **Rakust** decided discretion was needed, and so left **Championgirders** to settle in **Foggyrims**.

Figure 20.2: (left)A *Legends Viewer* view of one character's progression through their *Dwarf Fortress* lifespace. (right)Output from Jacob Garbe's narrative search-and-retelling algorithm. (both photos used with permission of Jacob Garbe)

20.3 Contests

Many users are naturally competitive, competition is a common form of game motivation, so it may not be surprising that contests are one of the most common forms of value creation in casual creators. Competitions can cater to a wide range of user skills and interests, and can be run in-game or in out-of-game channels, and by the official development team of the system, or unofficially by users.

Bullinger and Moeslein catalog the range of community innovation contests, such as the "Google Lunar X Prize" and "The Sims 2 H&M Fashion Runway"[s206] which solicit contributions from an open field of the community[45]They identify 10 design elements of contest-building and their options, in the image below:

Design element *(synonyms):* definition	**Attributes**					
1 Media (-): environment of IC	Online		Mixed		Offline	
2 Organizer (-): entity initiating IC	Company	Public organization	Non-profit	Individual		
3 Task/ Topic specificity *(problem specification)*: solution space of IC	Low (Open Task)		Defined		High (Specific task)	
4 Degree of elaboration *(elaborateness, eligibility, degree of idea elaboration):* required level of detail for submission to IC	Idea	Sketch	Concept	Proto-type	Solution	Evolving
5 Target group *(target audience, target participants, composition of group)*: description of participants of IC	Specified		Unspecified			
6 Participation as *(eligibility)*: number of persons forming one entity of participant	Individual		Team		Both	
7 Contest period *(timeline)*: runtime of IC	Very short term	Short term	Long term	Very long term		
8 Reward/ motivation (-): incentives used to encourage participation	Monetary		Non-monetary		Mixed	
9 Community functionality *(community application, communication possibility, tools)*: functionalities for interaction within participants	Given		Not given			
10 Evaluation *(ranking)*: method to determine ranking of submissions to IC	Jury evaluation	Peer review	Self assessment	Mixed		

Figure 20.3: Example "community innovation" contests (from [45])

Their ontology is specifically for contests with an extrinsic goal (to facilitate space flight, or develop content for a product), so additional elements in casual creator contests might be Solution Threshold and Creativity. In a low solution threshold task, users might only need to submit a form (or retweet a tweet, like a

post, or join a Facebook group) in order to be eligible to win. In a high solution threshold task, users might either have to meet some bar (solve a puzzle) or beat out other contestants in order to win. A high Creativity contest would require a "creative" or playful solution from the user, while a low-creativity contest might still be difficult, but not creatively linked, such as the unofficial *No Man's Sky* creature size leaderboard from the previous chapter. It does not take creativity, as such, to find a large creature, but it does require skill and patience. For an example of a medium-threshold, high creativity task, one year the *Spore*^{s215} team held a fruit-cake contest, in which players had to make a fruit-cake in order to enter, but winners were randomly selected from that group[4]. In a contest with higher stakes, one could imagine receiving low-quality creations as submissions, but the prizes were some small tchotchke and *mostly* the fame of having their names featured on the front page ("Prizes: Glory - as always - and 10 lucky Sporumers will be chosen randomly to receive a free gift from Maxis!"), a prize which meant more to creators who *already invested in their creative reputations on the site*. Players were more motivated by increased fame and creative reputation than the monetary advantage.

We also had competitions for serious money from granting agencies, contests won by the highest quality creation, contests won by being the 100,000,000th creation, and a contest where users created tools *outside* the *Spore* editors (the API contest). We also saw many, many user-driven contests. In the same way that users invented the idea of mailboxes to fill a need, they also hacked together structures to create and run contests and award prizes.

[4] `http://forum.spore.com/jforum/posts/list/47291.page`



Figure 20.4: Official *Spore* contests with a range of rules, prize styles, and motivations.

20.4 Alternative channels

One surprising property of many successful casual creators is that they encourage or enable users to communicate on channels *outside* of the system itself. This runs counter-intuitively to the belief that the ideal is a walled garden of a game/social-network/app-store where all communication is under the umbrella (and control) of the system's creators. However, letting users have the freedom to self-promote on the channels *where they have already established social, reputational, or monetary investments* provides a much bigger boost to their motivations than asking them to keep communications to channels where they have fewer friends and commitments.

Spore was my first exposure to this principle. I don't know the reasons be-



Figure 20.5: Unofficial Spore contests, with trophies invented and awarded by users.

hinds the decision, but every save file of a creation in *Spore* (such as a creature) is a .PNG file, a lossless image file with **all the model data needed to create the finished artifact of the creature**, stenographically embedded in the transparent pixels. This made it easy for players to manage their saved creatures in desktop folders: creatures are clearly identifiable in file thumbnails. It *also* meant that users could share creatures on *any* of the internet tools that supported non-



Figure 20.6: A Christmas contest for *Oculus Medium*, keeping the *Spore* traditions alive.

compression image uploading. This included nearly every existing piece of forum software in 2007, so *Spore* quickly developed alternative forums to the Sporepedia where users could take advantage of more traditional forum features like threads and conversations that weren't implemented in the Sporepedia, as well as a lack of oversight from EA/Maxis as to what was permissible. SomethingAwful became the new home of "offensive" creations that would be banned on the official site, to the advantage of users who wanted this content *and* EA who did not want responsibility for its existence[335].



Figure 20.7: A creature save file. Go on, drag it into the Spore editor. It should load *from an image embedded in a dissertation.*

Around the same time as *Spore* was released, another indie game, *Fantastic Contraption*, created by Colin Northway, was released on Flash[s725]. In Fantastic Contraption, users build simple machines from levers and simplified motors to autonomously solve problems in a 2D physics simulator. The solutions could be quite convoluted or clever, so users would naturally want to show that they had not

[5] `http://fantasticcontraption.com/original/`

only beaten a level, but had done so with elegance and creativity. Northway very wisely made a website where users could register their solutions (a reasonable task since the game was already on a web page). In addition, each solution had a unique URL, and loading that URL in a browser would open *Fantastic Contraption*, load the level, and spawn in the solution. Like the *Spore* saved-games, a saved game in the form of a URL can be passed over almost any medium.[6] Thus, users could quickly pass them back and forth, view them, upvote them, and learn from and modify the solutions from other players. Each time a URL was posted to say "look at my clever solution", the game itself was propagated, like a seed buried in the sweet fruit of self-promotion.

Though Flash has nearly faded out from the web, and the latest version of *Fantastic Contraption* is VR and lacks all the social sharing of the first version, at least one other game has followed the lessons that it laid down. *Opus Magnus*[s155], like *Fantastic Contraption*, is a game about solving puzzles with physical constraints. From a designer (Zachtronics) known for his diabolically difficult and addictive puzzle games about building machines, *Opus Magnum* requires the player to construct alchemical engines to transform different bubbles of matter into others, via components like tracks and levers laid out on a hexagonal grid, to create a repeating cycle to manufacture some alchemical result. When a solution is complete, the user can see where they rank on number of cycles, cost, and area against the global leaderboard of all other players. Crucially, the player can *also* export their solution as a looping gif, a hypnotic artifact of their solution, which can be posted to social media. For a few months in 2018, Twitter seemed full of looping alchemical engines, as users showed off their skill or creativity.

These solutions cannot be loaded into the game (as *Spore* and *Fantastic Contraption*'s save files could be), but they do make it easy to see common techniques

[6]With the exception of Uber messages and Zillow postings which prohibit URLs.

and verify high scores[420]

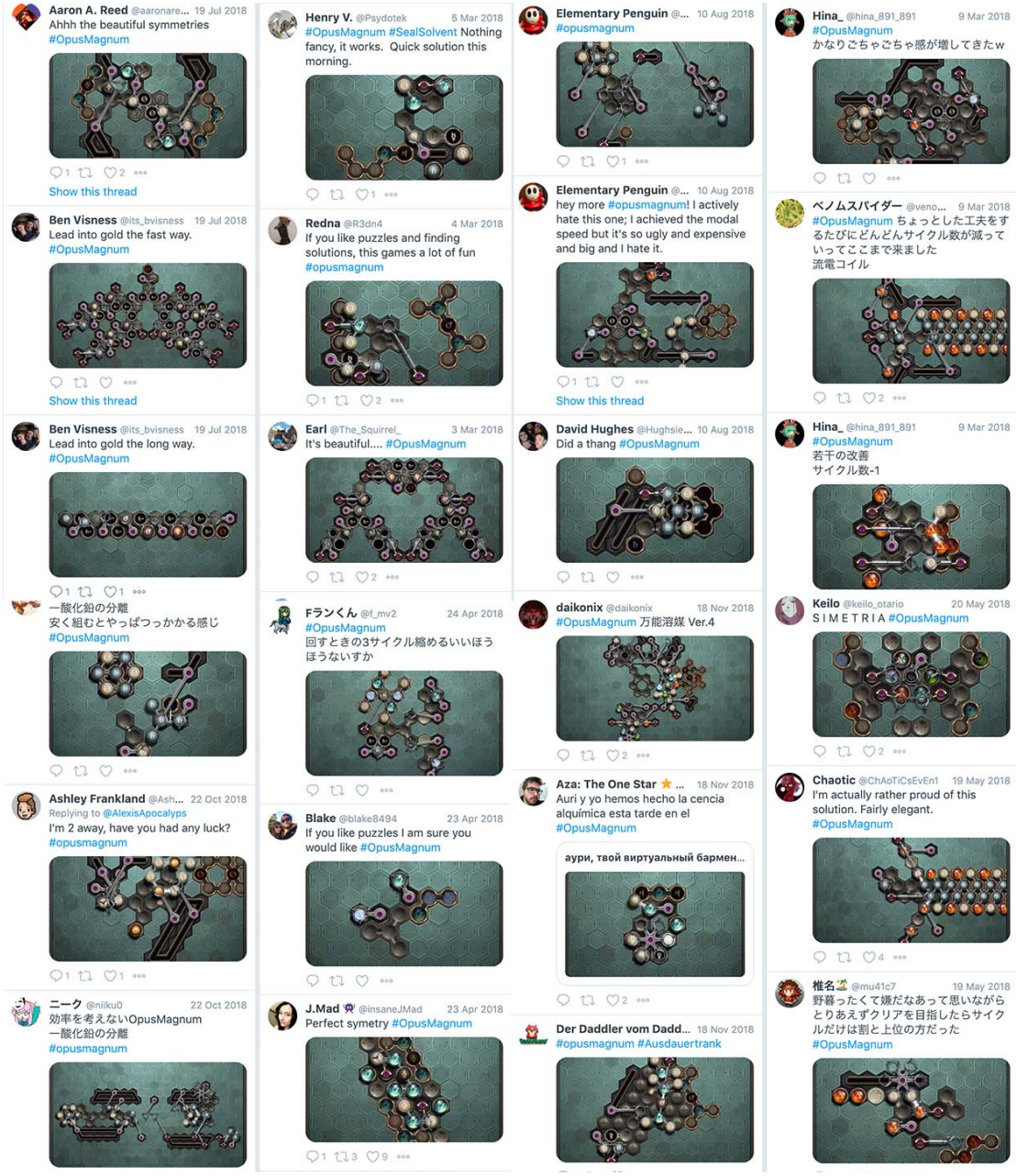


Figure 20.8: Solutions posted to Twitter.



Figure 20.9: A creatively-terrible *Opus Magnum* solution(from user @britishgaming (`twitter.com/britishgaming/status/922022684116045825`)).

20.5 Fame and Money

Fame and money need little introduction as motivations. If users can gain fame or money from a platform, they have incentives to use it and promote it. To see this in action, one could look at the rise and fall of *Second Life*[s198] as a creativity and money-making platform[435]. Users could trade real estate, get

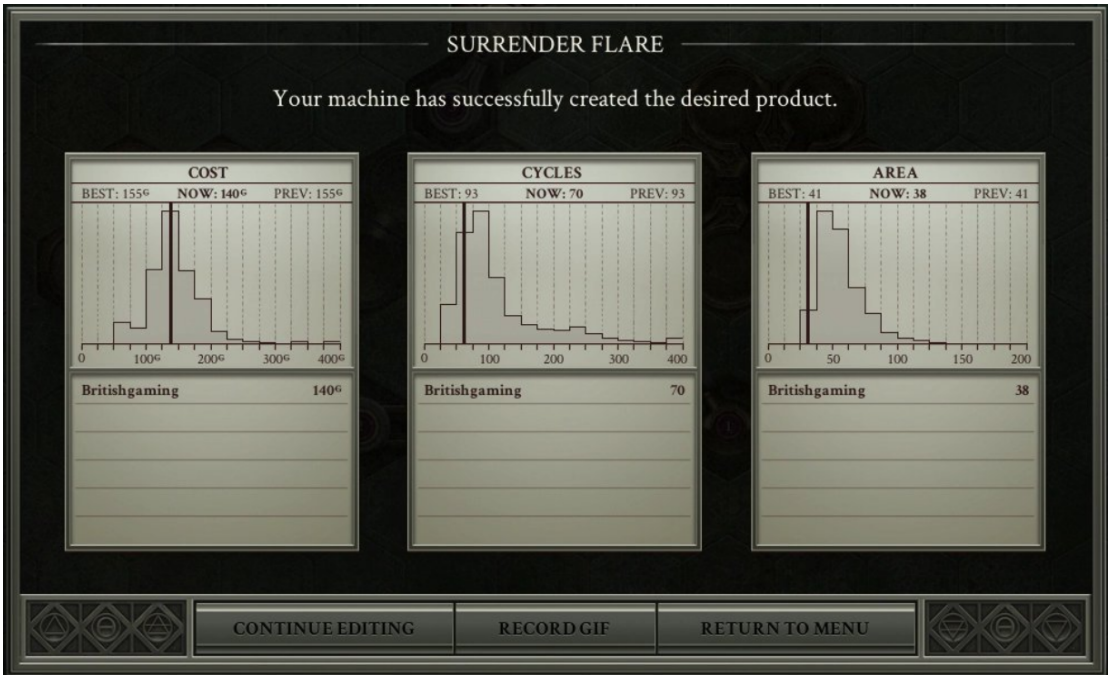


Figure 20.10: An *Opus Magnum* leaderboard (from [420]).

jobs, and make a range of creative content, for sale or on commission. Items could be built in the in-game editor, or imported from a variety of external tools (3D modeling programs or photo-editing tools). All items were paid for in Linden dollars, an in-game currency which could be exchanged for real-world cash.

Not many people would recommend playing *Second Life* for a career now, but the focus has instead shifted to the money and fame showered on Youtube and Twitch celebrities[7]. The most streamed games are competitive games, but creativity games like the Sims and *Minecraft* are also popular for streaming. While there is demand for watching highly skilled players be competitive in a rule-focused game, there is also pleasure in watching people exercise playful creativity in interesting constraints. That is, there is room on television for football games *and* competitive creativity shows like "Project Runway", "Cake Wrecks" and "RuPaul's Drag Race", as well as meditative creativity shows like "The Joy of Painting".

[7]Let's see how well this paragraph ages in the next ten years.



Figure 20.11: Books and a magazine cover from *Second Life*'s zenith as a money-making platform.

What makes a casual creator a good fit for streaming performance?

- Fan service: making custom content according to fan desires (themes, personalization)
- Technical challenges, showcasing exception technical skill at using the tool
- Creativity challenges, e.g. reinterpretations of famous people or characters into game editors, or design challenges
- Humor: potential for humorous creativity, such as juxtaposing semiotic assets (see the next chapter on chapter 24)
- Unexpected emergence: discoverable secrets, surprises that the streamers can react to

Not every streamed casual creator has these properties. Some streams are also focused on a meditative, soothing "ASMR" experience, where the viewer participates in the flow state of the user. The classic example of this is Twitch's

rebroadcast of all 403 episodes of Bob Ross's long-running PBS show, "The Joy of Painting"[8], but people also broadcast drawing, knitting, painting[9], and yes, filling in coloring books[10]. For digital systems, *The Sims*[s206] is a popular game for streaming, both for users playing the game, but also users focusing exclusively on the house editor and the *Make-a-Sim* mode.

Many of the top-paid streamers began their careers playing *Minecraft*[s136] on Youtube[61]. *Minecraft* seems to be particularly well-suited to streaming. It has immense potential for customization, so users can set it up as

- a survival mode for intense terror and stress (horror games are streamer favorites for their over-the-top-reaction potential)
- a no-holds-barred creativity mode with macros and scripting for creativity challenges, like the reconstruction of **Game of Throne's** "Kings Landing" city or the *Minecraft* computer[372]
- a low-threat no-macro mode, to focus on building technically difficult bases within the original constraints of the game
- a large-map exploration game where the focus is on exploring a generative world with hidden secrets
- multiplayer team efforts or competitions, e.g [285]

Microsoft (and Mojang, before it was purchased) recognized that every time people showcased their creativity, humor and skill through *Minecraft*, they were also *selling copies of Minecraft*. Not every game company supports streaming

[8] `https://blog.twitch.tv/for-the-love-of-bob-6b3b73957938`

[9] `http://makerstyle.ca/blog/introducing-twitch-creative-will-you-stream-your-craft`

[10] `https://www.twitch.tv/videos/348829800`

(Nintendo has notoriously had a rocky relationship with it), but for *Minecraft*, the streaming and self-promotion communities around it were the key to its success. In return, Microsoft has engaged in a variety of initiatives to support creative users, like showcasing their work (fame!)[272] and creating a marketplace where users can sell creations[11](money!). This has resulted in professional *Minecraft* users making art[18] and making a living at it:

> "Back then I was really into creative-style games, games where I could create my own things, and me and few of my friends were chatting about that and one of them thought of Minecraft," Freeburn says. "I bought the game a week after that and sunk in so many hours, just building and building things." (...) Having turned 18 and assumed ownership of the company from his father, Freeburn has built the business to the point where it's doing well enough to put him through university and even pay off another member's student debt, too." [213]

Minecraft is not the only creativity system[12] with a large Twitch/Youtube community. *The Sims* has a large population of streamers, some *exclusively* streaming in the *Create A Sim*[s206]. I sampled a set of them on a recent night[13], and found a range of streamers performing with a range of goals:

- Fan service: making portraits of followers, making Disney fan content
- Technical challenges: trying to raise 100 babies, or marry and kill a husband as fast as possible (the "Black Widow Challenge")
- Creativity challenges, participating in design challenges like the Zodiac Design Challenge[14] or Disney Princess Challenge

[11] `https://marketplace.minecraft.net/en-us`

[12] *Minecraft* is an autotelic creativity tool, but is not usually played as a system focused on the exploration of a generative possibility space. It could be argued that it is or isn't a casual creator, but it is large and famous enough to have developed a number of phenomena which can be applied to casual creators in general.

[13] `https://www.twitch.tv/directory/game/The%20Sims%204`

[14] `http://kozykale.com/2018/12/31/the-zodiac-elements-community-build-challenge%EF%BB%BF-collab-with-simarchy/`

- Humor: goofing around making cats in sweaters
- Unexpected emergence: reacting to unexpected moments that the Sims get into (fires, mistakes)

Many streamers were also soliciting funds for themselves, or for charity.

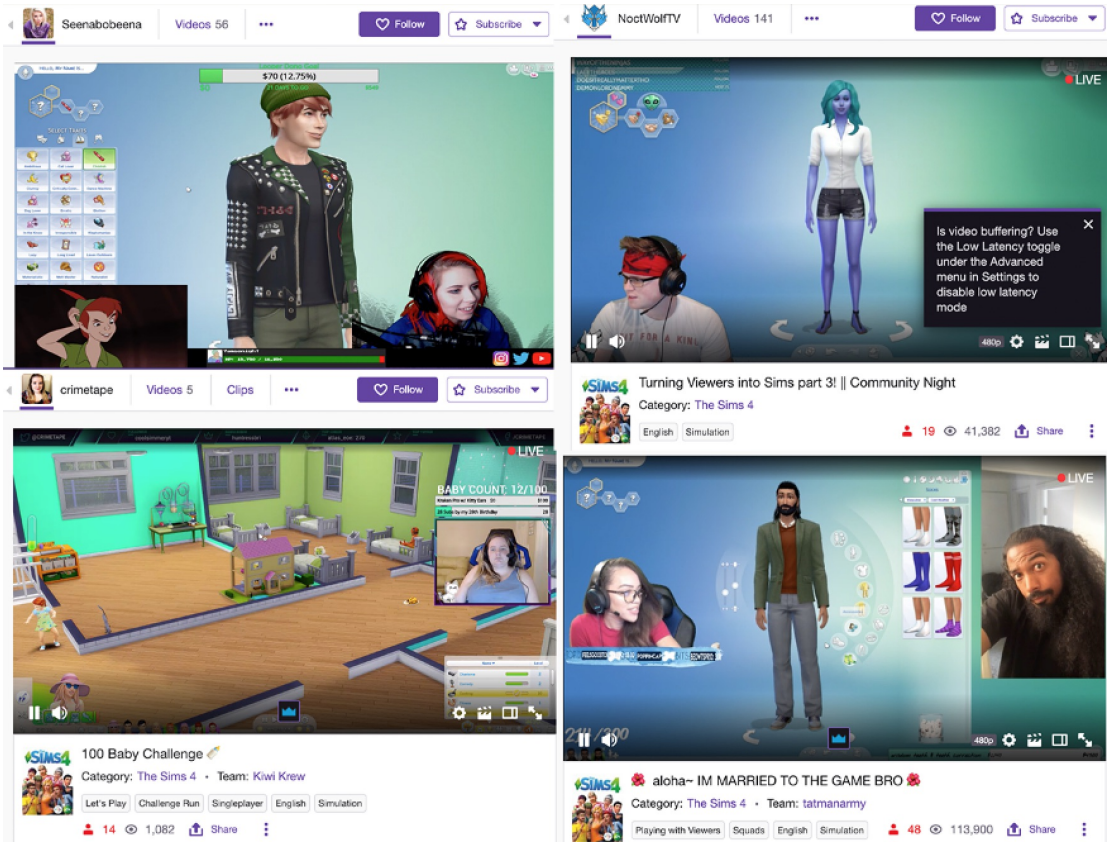


Figure 20.12: A sampling of Twitch livestreams active at a single moment on Jan 11th, 2019. We can see a gameplay challenge (have 100 babies), a fund-raising challenge, a viewer service challenge (fan portraits), and creativity challenges (making Peter Pan, a portrait, a castle).

Not every attempt at letting users gain fame or money for their creativity is welcomed or successful, however. Steam enabled modding-for-pay in an established community of *Skyrim* modders (a game where users can create new dialog, quests, rules and more) with the intention of supporting the development community. While playing *Skyrim* (or modding it) is not a casual creator, these events

provide a rare look at a company providing in-game monetization of creative expression:

> "We think this is a great opportunity to help support the incredible creative work being done by mod makers in the Steam Workshop and to encourage more top-quality work. This new feature allows mod authors to choose whether to list their items for a fixed price, for pay-what-you-want, or to make their item available for free. As a customer and fan of Skyrim, you're able to explore both paid and free mods, quests, and items."[15]

After a disastrous launch with players from the modding community (and outside of it) they quickly disabled it.

> "To help you understand why we thought this was a good idea, our main goals were to allow mod makers the opportunity to work on their mods full time if they wanted to, and to encourage developers to provide better support to their mod communities. We thought this would result in better mods for everyone, both free & paid. We wanted more great mods becoming great products, like Dota, Counter-strike, DayZ, and Killing Floor, and we wanted that to happen organically for any mod maker who wanted to take a shot at it. But we underestimated the differences between our previously successful revenue sharing models, and the addition of paid mods to Skyrim's workshop. We understand our own game's communities pretty well, but stepping into an established, years old modding community in Skyrim was probably not the right place to start iterating. We think this made us miss the mark pretty badly, even though we believe there's a useful feature somewhere here."[16]

Letting users gain fame and money, and develop creative networks is a strong pattern in casual creators. But while we can identify common structures and needs of the community, there are still pitfalls that we do not yet know how to avoid. In a way, self-promotion can feel antithetical to casual creators: casual

[15] https://steamcommunity.com/games/SteamWorkshop/announcements/detail/208632365237576574

[16] https://www.reddit.com/r/pcmasterrace/comments/3434it/paid_mods_in_the_steam_workshop/

creators often rely on a space where people do not feel creatively judged, as we saw in chapter 18. Does implementing a system where some users are *clearly* better and more famous (and even paid!) suppress the participation of less-skilled users, even as it increases the motivation of skilled users? However, we may want to build casual creators in which the user *values* their own creative output, and part of that contract is providing ways for users to share, promote, and **own**[17] their creative work.

[17]Ownership can be the ability to put their name on it, keep their save file outside of the system's servers, monetize it, or many other ways to interpret ownership that can vary by system or domain.

Chapter 21

Pattern - Repetition, Repetition, Repetition

> "You know what Aquinas says: The three things requisite for beauty are, integrity, a wholeness, symmetry and radiance." - James Joyce

Casual creators' central principle is a loss of control in exchange for power. We've seen a few kinds of this power: power for correctness, error-checking, feedback, and more. In the next three patterns, "Repetition", "Transformation", and "Make it Real", we will see a few common ways that users actions are magnified and empowered by computational systems (and a few non-computational systems).

Perhaps the most easily-implemented (and therefore most common) casual creator power-play is to add precision and symmetry to the user's actions. Humans find it very difficult to make precise or symmetrical strokes. For proof, try drawing a perfectly symmetrical circle. It requires a range of different muscle movements as your hand move up the page, down the page, to the right, and to the left. The wobbly lumpiness of the circles we draw even records information about our cultural backgrounds[177].

One alternative form of symmetry and precision is translational symmetry, or repetition. Its uncertain why humans enjoy repetition, symmetry, and the precision needed to make them, but it is undeniable that we do. Likewise, numerous crafting techniques exist to make imprecise humans capable of inhuman precision, such as weaving or wallpaper patterns. The techniques of Italian millefiori[s135] and Japanese yosegi (or Hakone) marquetry, in which thin strips wood or glass are carefully bundled together, and then sliced very thin, in order to make dozens or hundreds of identical patches, which can be inverted, and applied side by side to create dazzlingly repetitive patterns.



Figure 21.1: Mechanically-assisted repetition in Italian millefiore and Japanese yosegi woodwoorking; both are done through bundling long strips then thinly slicing the bundle to create identical tiles. (from Wikimedia user *Captain76* and Ulrich Mayring)

As hard as precision is, paradoxically, randomness can also be hard for humans to achieve, and easy (at least for pseudo-randomness) for computers. So precision, symmetry and randomness are often paired together, as in these images from the first computer art contest in August 1965, in the magazine Computers and automation:

Precision comes for free with computers; it requires conscious effort to make a computer imprecise. Symmetry, likewise is easy to achieve programmatically:

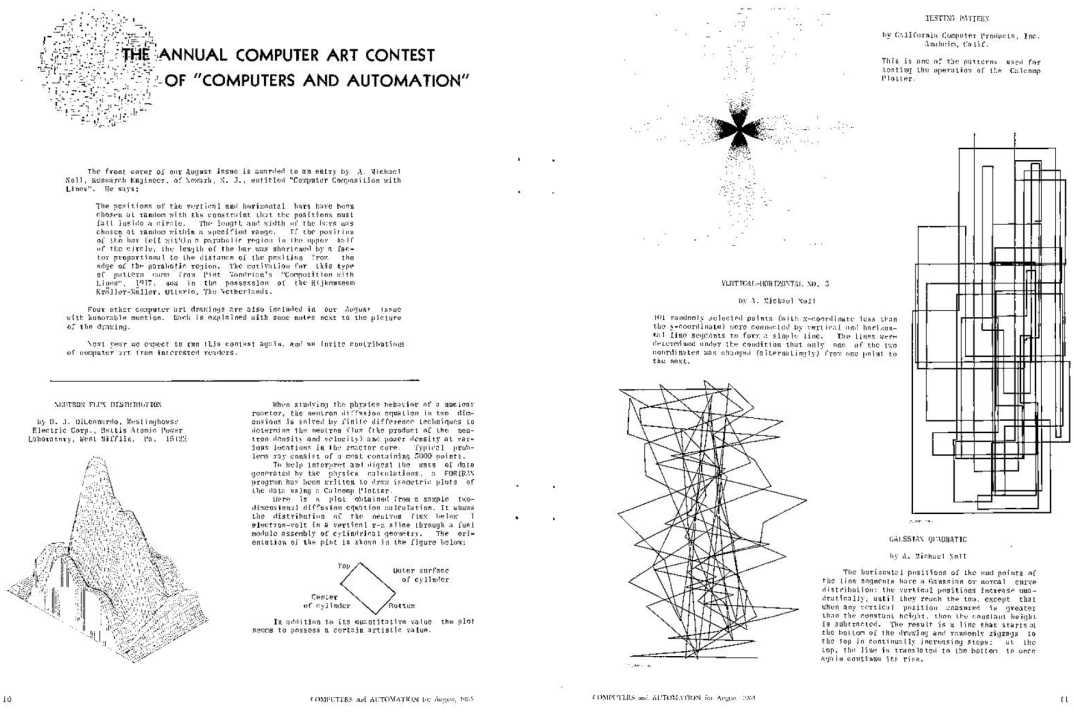


Figure 21.2: a page from computersandautomation, aug 1965:The winning artist's statement from A. Michael Noll: "The positions of the vertical and horizontal bars have been chosen at random with the constraint that the positions mus fall inside a circle. The length and width of the bar was chosen at random within a specified range. If the position of the bar fell within a parabolic region in the upper half of the circle, the length of the bar was shortened by a factor proportional to the distance of the position from the edge of the parabolic region. The motivation for this type of pattern came from Piet Mondrian's "Composition with Lines", 1917, now in the possession of the Rijkmuseum Kröller-Müller, Otterio, The Netherlands."

in its simplest form, it can be created by adding a negative sign to the plotting instructions, or for translational symmetry, repeating the instructions with an offset. What is notable in these examples is how quickly in the birth of computer art **randomness** becomes an ingredient. Three of the five entries in that early contest use it, for visual interest. Even then, they seem to have intuited the equation: "randomness + precision = art". Precision and symmetry elevate randomness from noise to an aesthetic pleasure.

Indeed, long before this computer art contest, the toy kaleidoscopes[s110] of Victorian children were combining symmetry and randomness. These devices, invented in 1816 by the later inventor of the stereograph, Sir David Brewster, used a tube of mirrors and a glass tray of beads and colored glass shards into a perfectly tiling wallpaper of colored light.

21.1 Precision, symmetry and interaction

When precision and symmetry provide such an uplifting transformation to randomness, it is not difficult to believe that if the randomness is replaced with human action, the emergent aesthetic properties will continue. From the array of interactive symmetry apps available, this does indeed seem to be the case. These seem to fall into two categories, based on what the user is providing: gesture or image-based symmetry.

21.1.1 Gestural symmetry

The first one is **gesture-based symmetry**, in which the user provides a path that is symmetrically copied. In these apps, the user is almost always drawing with a mouse or a touch pad, and usually the app *continuously* replicates the drawing symmetrically around an axis. As we've seen repeatedly in this dissertation (and in chapter 12, continuous feedback seems to be critical in allowing users to see the result of a hard-to-visualize process, such as the emergent criss-crossing patterns and negative spaces of a symmetric pattern.

The most common symmetries I see in casual creators are:

- mirroring: reflecting across an axis (almost always horizontally)
- rotational: rotating and duplicating the stroke around a central point

- mirrored rotational: rotating both the stroke and the mirrored stroke
- wallpaper symmetry groups: reflecting, rotating, and translating
- scale-and-rotation: creating a spiraling effect

Creators that apply simple horizontal repetition or tiling seem uncommon, perhaps because they lack the emergent qualities of reflective symmetry. The exception to this is tessellation creators, like *Interactivate:Tessellate!*[s2021], in which, like an M. C. Escher tessellation, the creativity and surprise emerges from the re-use of each side's inverse profile in some creative way.

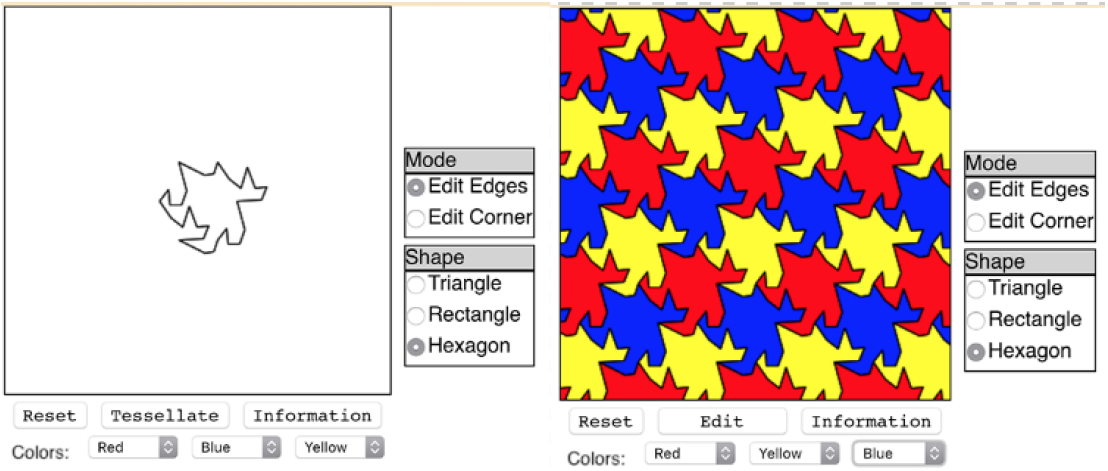


Figure 21.3: Making a hexagonally-tessellating shape with the online *Interactive:Tessellate!* editor[s202].

The gestures themselves may be interpreted as individual points, but often the system provides precision-assistance, by:

- letting the user place individual points, though which it draws straight lines
- re-interpreting the drawn points into a smooth (e.g. Bezier) curve, smoothing out any jankiness (often providing a "smoothing" slider)

[1] `http://www.shodor.org/interactivate/activities/Tessellate/`

Some systems also provide *randomness* for user gestures. Notably the *Silk - Interactive Generative Art*[s2032] app takes the user's gesture, smooths it, then multiplies it, offseting each control point to create what looks like a twisting skein of filaments. *Silk* also provides a variety of symmetry modes: 2, 3, 4, 5, and 6-fold rotational symmetry, mirroring, and spiraling (a combination of repetition and scaling).

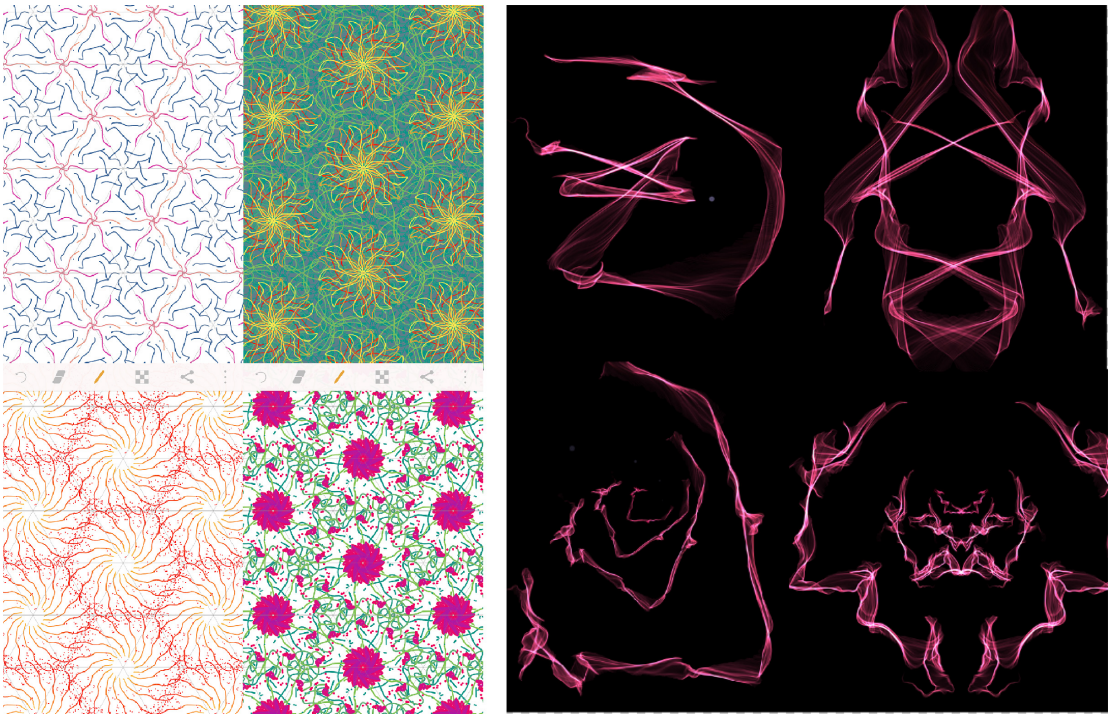


Figure 21.4: (left) Example of *Spiro* drawings. (right) Symmetries in the *Silk* app (none, mirrored, spiral, and spiral-mirrored).

The Android app *Spiro*[s213] lets the user draw the usual radially-reflected mandala shapes, but also has a translation/repetition mode, in which the users can draw continuously-updating wallpaper-style patterns. Mandala-makers are common, but this was the only example I saw of rotationally-symmetrical mandalas also being used in repetitional symmetry.

[2] `http://weavesilk.com/`

Tilemaker[s234] is an educational casual creator for teaching a few principles of Islamic tilemaking, by Qatar's national cultural organization.

> "Qatar Foundation International's (QFI) Mosaic Tilemaker application introduces students to fundamental concepts of Islamic art and architecture through the exploration and creation of mosaic tile art. Through this app, users can learn more about the history and geometric principles behind this art and engage in an interactive learning process to explore these concepts by designing and sharing their own mosaics through the apps Create Tool."

Its gestures are highly-constrained: users can draw circles and lines, but can only draw from point-to-point on existing intersections. This severely restricts the placement of geometry, but in the service of making a crisper and more traditionally-appealing pattern.

Engare[s66] is a game and casual creator by Iranian mathematician Mahdi Bahrami. Players are challenged to discover the indirect movements needed to replicate geometric patterns drawn from Islamic art. There is also a "sandbox" mode where players can use a variety of indirect and direct means (linkages, Spirograph-like machines, as well as directly controlled paths) to draw paths which are repeated into tiling structures. Some of these paths are very indirectly controlled by using sliding tuners, which control the behavior of linkages, which rotate to draw curves, which are *then* repeated symmetrically. This has the interesting feeling of imprecisely sliding a linear slider, but seeing that gesture translated into a dazzling set of complex yet precise curves.

> "Part puzzle game, part drawing tool, Engare is all about your geometric imagination. It's a meditation on the dynamics of simple machines and the beauty buried deep within repeating motion."[3]

[3] `https://store.steampowered.com/app/415170/Engare/`

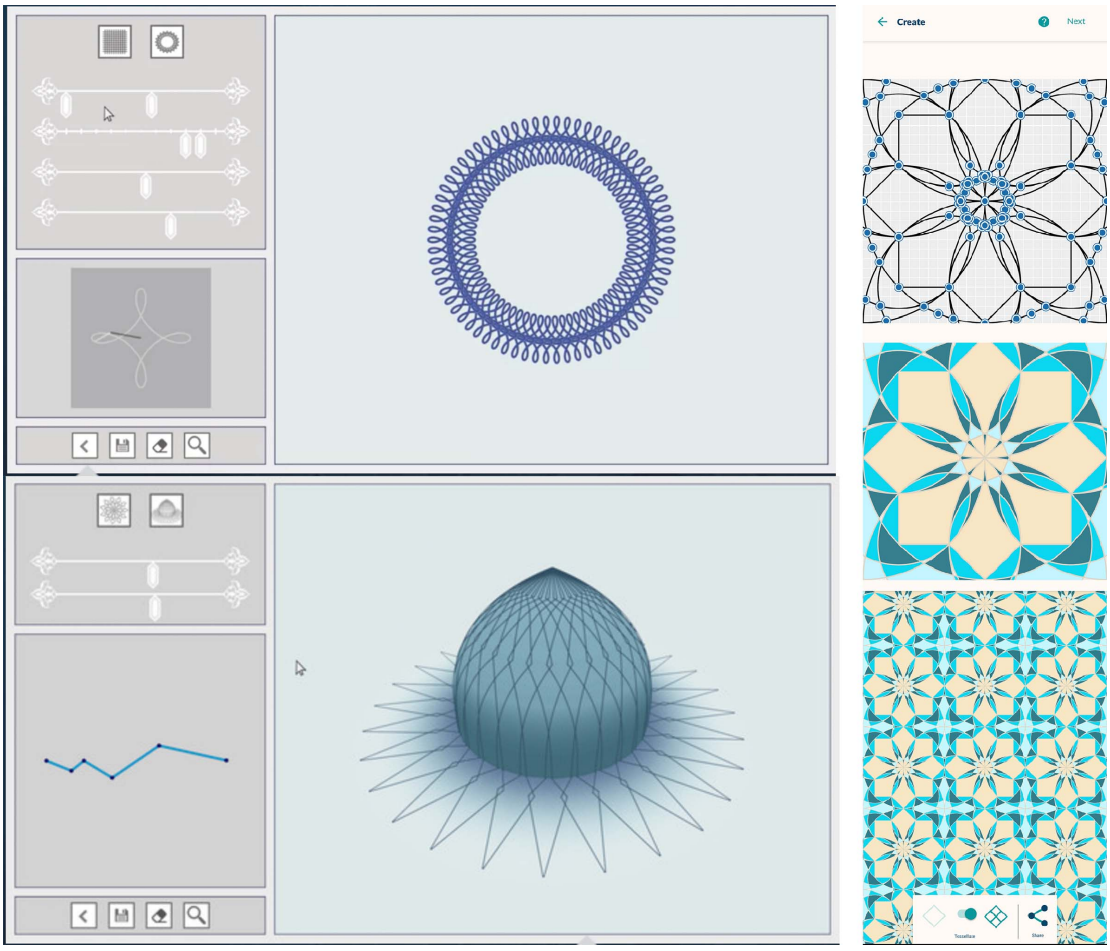


Figure 21.5: *Tilemaker*(right) and *Engare*(left) are two systems which use constrained user gesture (or physically-mediated gestures) to create patterns from repetition and symmetry.

21.1.2 Manipulating user-provided photos

A few casual creators take in user-provided images, instead of gestures, for their source material. Even the original kaleidoscope[s110] had an additional variant designed to allow the user to reflect the outside world, instead of glass beads. As we will see in chapter 24, emergence can come from seeing familiar images reinterpreted in new ways. There are dozens of apps for users to see what they would look like with a symmetric faceFigure 21.6. Kaleidoscopes for user-provided photos are plentiful, as well as apps that let users view video-streams continuously

though a kaleidoscope filter. Like other casual creators that use photos, the user has the creative choice to capture or select a photo, but this can feel like a more limited, and "easier" input than trying to make the "correct" gesture.

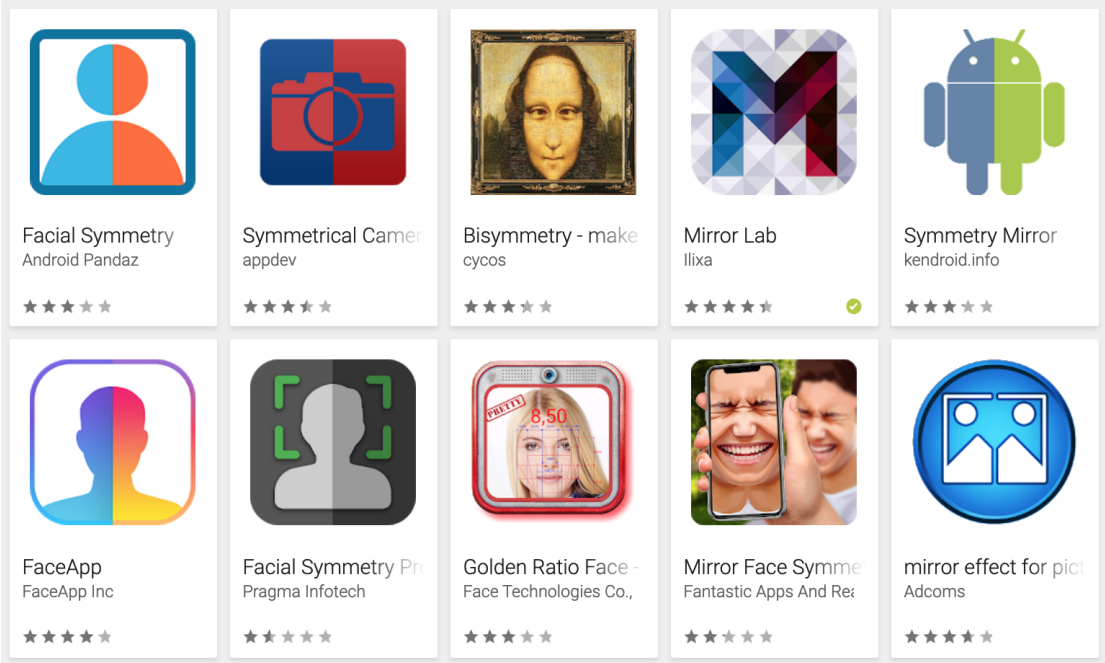


Figure 21.6: Symmetrical photobooths are common on the app store. They combine immediate feedback with the actractions of symmetry, and the interest of seeing one's own face in a new way (Android app store shown).

These are only a very few of the examples of casual creators that exhibit the pattern of symmetry and repetition. Mirrored symmetry is the default mode in almost every character creator, including the *Spore Creature Creator*[s216] (until players developed glitches to circumvent it). But simple symmetry and rotation and precision can be combined with even the most simple line-drawing interaction to create viable experiences. To verify this, one needs only search for "mandala" or "kaleidoscope" on app stores to find dozens of them.

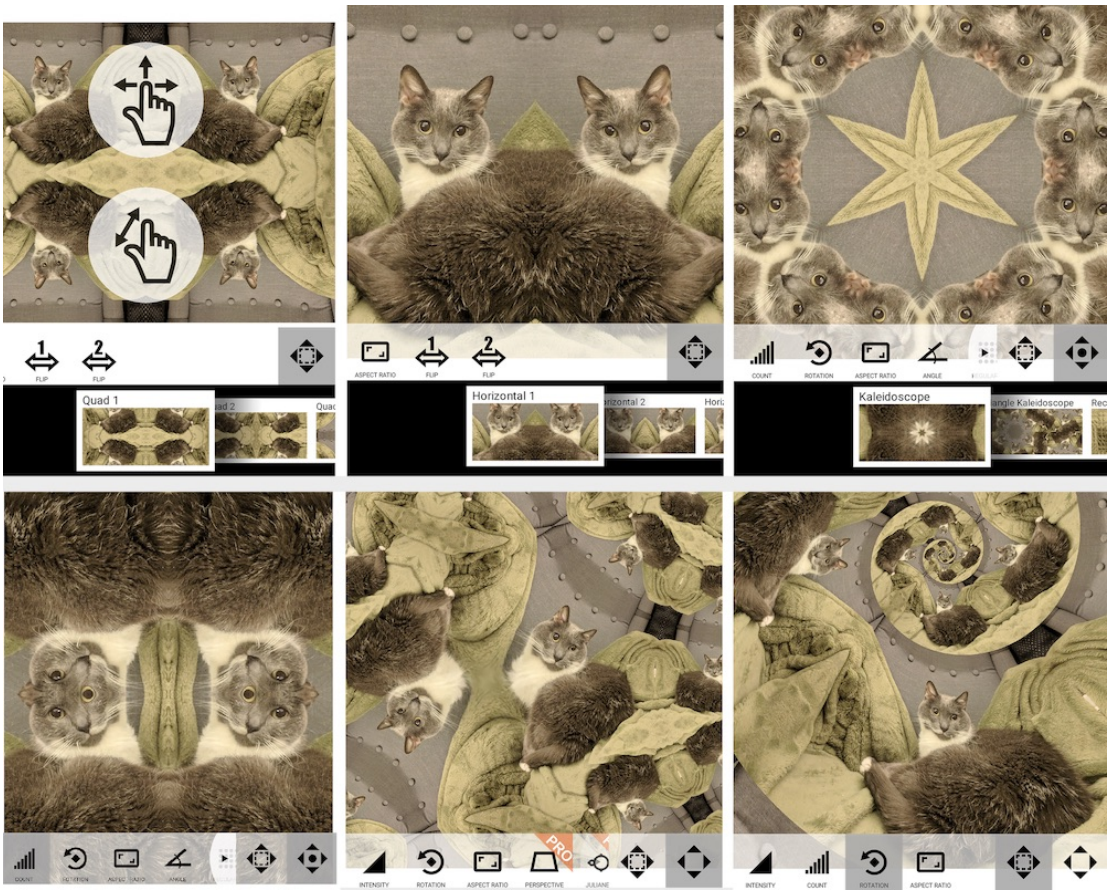


Figure 21.7: *Mirror Lab*, the most complex of the photo-symmetry apps, has a number of different symmetry and mathematical distortion filters. We can see a few ways of looking at one of my cats. The results are *more interesting* than they would be if the manipulated image were not of a cat.

21.2 Embedding symmetry in external systems

These symmetrical systems are so simple to use that they can be embedded or used in other systems. In *Monument Valley 2*[s139], at the end of the level, the user can use the touch screen to draw lines, which are continuously duplicated into a mandala. This glowing user-created mandala is then used as the symbol for the section. In that example, the interactivity relies on the constrained space of rotational symmetry to ensure that whatever the player draws is satisfactory (there is no "undo").

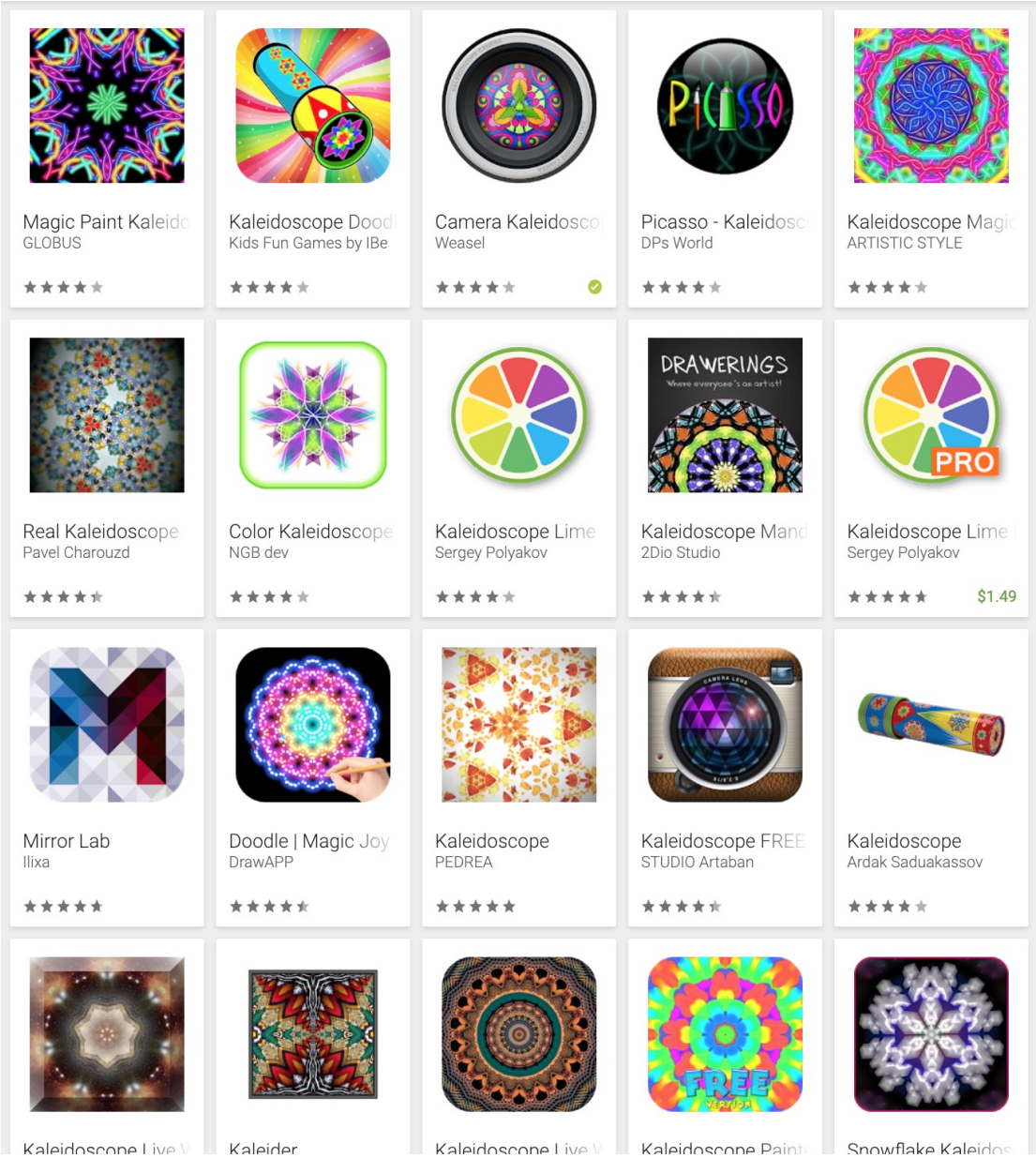


Figure 21.8

As we found with the Spore Creature Creator, if you give a user a constrained possibility space, some percentage of them will attempt to do something truly virtuosic, at the limits of its domain. When artist Sonia Harris[182] was recovering from chemotherapy, she started using the radial symmetry app *Amaziograph*[s5], and discovered a complex appreciation for its limitations (note the lack of automated

support for text, but the importance of the instant grokloop):

> "Amaziograph works a lot like drawing on paper works; there are no layers, you can't cut and paste things, you can mix colors but only as many fit in your palette and you can't save them, there are no alignment tools or text tools, so you have to write things yourself and align by eye. Except for the amazing ability to see patterns tile in real time as they're drawn, it is all very much like simply drawing in real life, which makes it that much more satisfying and enjoyable to tweak something elaborate out of it."[182]

Her drawings became successful, and she began producing them as physical products. From her quote, we can see the importance of *social creativity*, community and participation, no less strong here than in the examples in chapter 4

> "When I was drawing these patterns and posting them online for my own edification I found that people gravitated towards them. Their enthusiasm and enjoyment of my drawing fueled my recovery, and there were many times when sharing laughter with strangers over another swearing pattern was the brightest moment of my day. People often asked when I'd be able to put these patterns on clothing and other things, so I made it my mission to recover as quickly as possible so that I could give people this. Now my brain and my strength are slowly returning enough for me to handle making stores and filling them with things covered in my patterns."

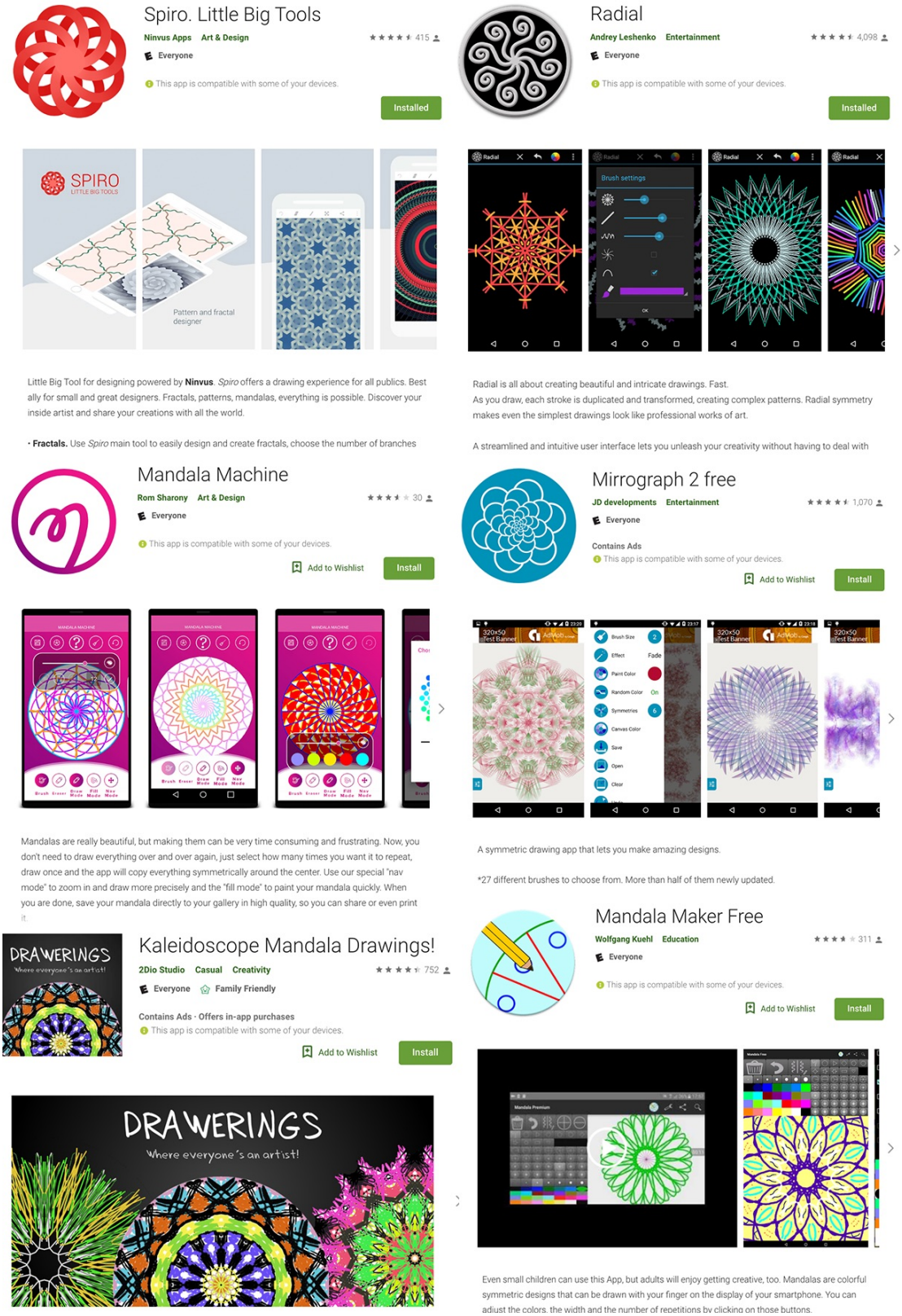


Figure 21.9: :A variety of mandala (rotational symmetry) drawing apps

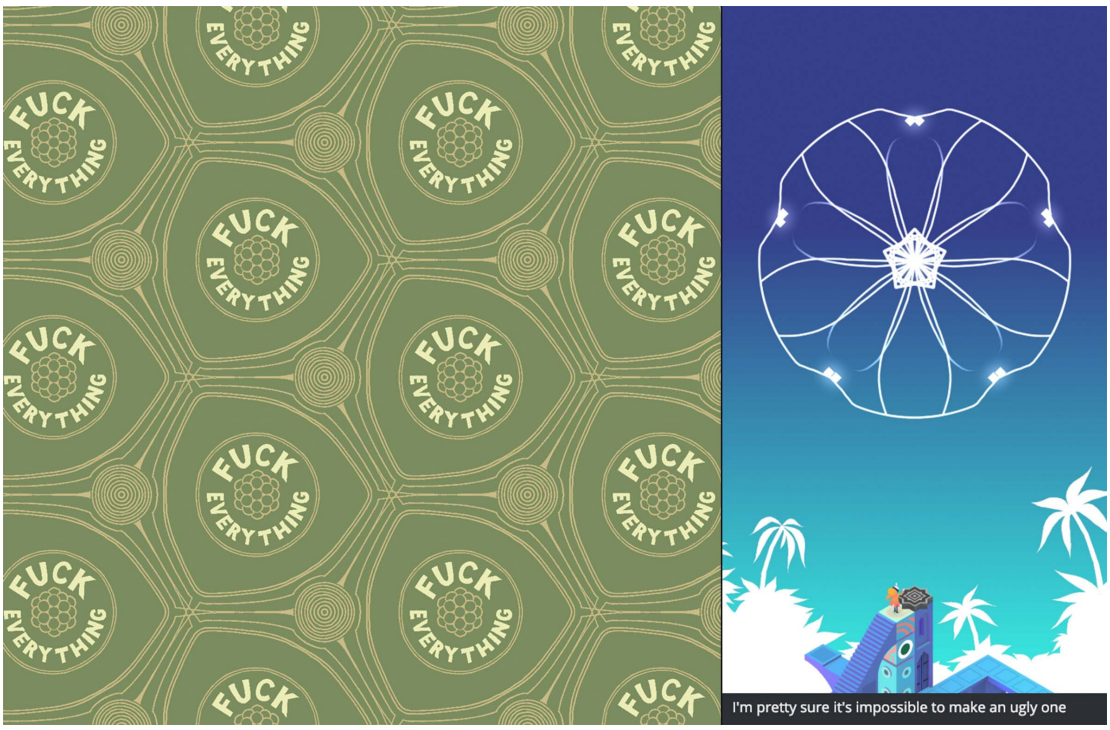


Figure 21.10: Sonia Harris's sardonic linework, User "blagh"'s *Monument Valley 2* mandalas, with appropriate commentary[416].

Chapter 22

Pattern - Transformation

This pattern ("transformation") is seen when user's work is somehow transformed beyond what they could normally do with the controls available. This can comprise a large range of possible software behaviors, but they are all linked by the feeling that the transformation is somehow magical, or out of reach for the user. Unlike symmetry, this is not a power that could be replicated by time or attention (at least within the app), but feels like a wholly "other" ability.

Transformation is too broad of a category to cover in a single chapter. There are many ways to achieve this transformation (one of which is the pattern examined in the next chapter 23), but within this chapter, we will examine the practice through systems that use "coloring" as an activity, and the dozens of digital applications that have replaced *some* part of the coloring paradigm with assistive or generative AI. We can look at "coloring" as a series of steps that a user takes, and ask, for each step, what does it mean to have a system provide a "magical transformation" at this step?

"Coloring", as in the act of coloring in enclosed regions in a coloring book, can be a type of casual creator in itself. There is no shortage of coloring apps on app stores, but the 2013-2017[331] adult coloring book explosion demonstrated that

even physical coloring books provide some of the experience of a casual creator. These books, sparked by the 2013 virality of Johanna Basford's "Secret Garden: An Inky Treasure Hunt and Coloring Book"[191], allow the colorists to feel creative control over *some* of the choices they make (color selection)[141], while not feeling judged on their drawing ability.

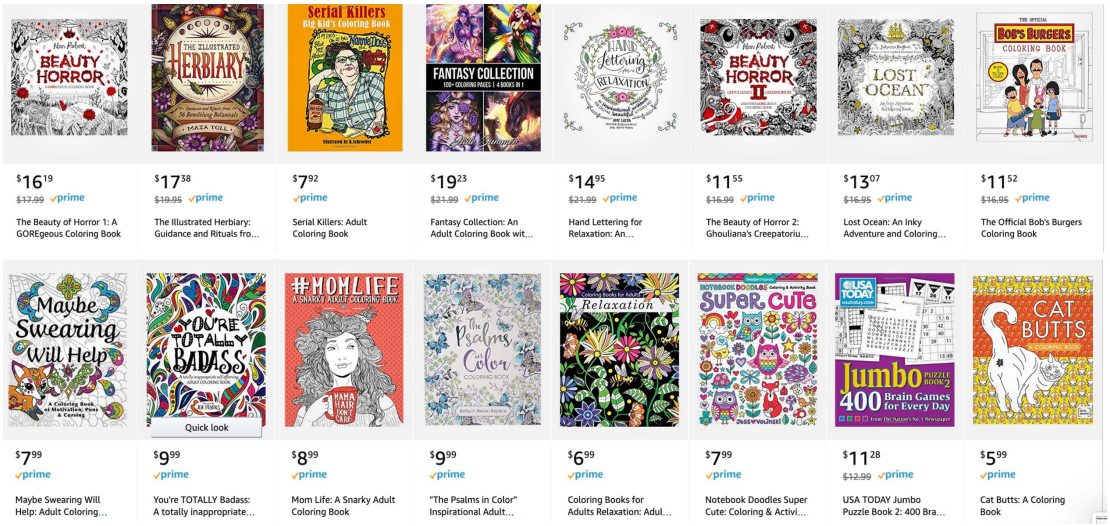


Figure 22.1: A variety of coloring books for sale on Amazon.com, with heavy use of psychologically or culturally meaningful signifiers (see chapter 24).

The act of coloring is the central mechanic to these systems. In traditional physically-printed coloring books, users fill in the regions with colored pencils, crayons, or paints. In the digital versions, they either tap to flood-fill a region or tap to fill in a single pixel (for more discussion of these apps, see chapter 10). For an image with hundreds of regions or thousands of pixels, this can be laborious, yet also meditative and calming. In all of these apps, there is a shared pattern of a system providing the "structure" of an outline, and a human providing the "coloring" of the regions.

As we have seen before in this dissertation, many interesting design ideas can be generated by "flipping the script" and reversing the roles of an existing

paradigm.

Let's examine the stages of coloring a coloring book. The roles in the coloring-book paradigm are:

- providing the lines and regions
- picking out
  - colors
  - regional direction
  - shading style (loose, striated, flat, internally-shaded, etc)
  - materials
- applying the colors with the style
- making use of the result

In a "normal" coloring book experience, the lines and regions are provided by a pre-printed book publisher, from drawings by a professional artist. The end user picks the colors, direction, and style, and then applies them by physically filling the region with strokes of some pigment.

We can see one common variation of this in the form of "paint by numbers" kits[s159]. These kits set the shading style (flat), material (pots of acrylic paint) and color, which is denoted by a number in each region corresponding to a pot of paint. In kits like these, the user has fewer choices, and less creative control, but more certainty that the end product will match a desired standard. Other activities like digital coloring books and embroidery kits share the overall structure of the experience (filling regions with fixed colors), despite the evident difference not having "pigment". The experience of all these style of activities is about

achieving the Flow experience of many small cumulative actions, rather than playful creativity. One shared property of these "paint-by-number" (and related) experiences is that their pleasure is very dependent on the satisfying tactile quality of these actions, whether screen taps, paint strokes, or embroidery stitches.

Paint-by-numbers kits remove creative choice from the color-selection stage of coloring books, replacing it with a preselected "safe" set of colors. We can also see what happens by *increasing* creative choice in some of the stages. The most open form of coloring would be a blank sheet of paper that the user could fill as they choose, but as we've seen in the chapters on creativitychapter 4 and blank slateschapter 17, *overwhelming* choice can be stressful and creatively paralyzing. The *Zentangle*[s247] art form (discussed earlier in chapter 12 encourages its users to make simple choices in one stage at a time, and without trying to achieve a particular look or mimetic image. The region creation step is intentionally simplified, often the user just makes crossing lines until there are "enough" regions. *Zentangle* users then use repetitive, heavily-algorithmic pattern rules to fill in each region. Users seem to delight in inventing and learning new patternsFigure 22.3, a collaborative creative interaction that takes place on social media platforms like Pinterest. We can see the relationship between coloring books, paint-by-number, and Zentangles, but also can see how small changes in the coloring book formula changes how users feel about the creative experience of each activity.

Each of these three variations on the coloring-book structure have some *casual creator properties*:

- the user is assisted by some structure (provided by the publisher or themselves)
- makes limited choices with the goal of inducing a flow state
- feels creative, but without fear of failure

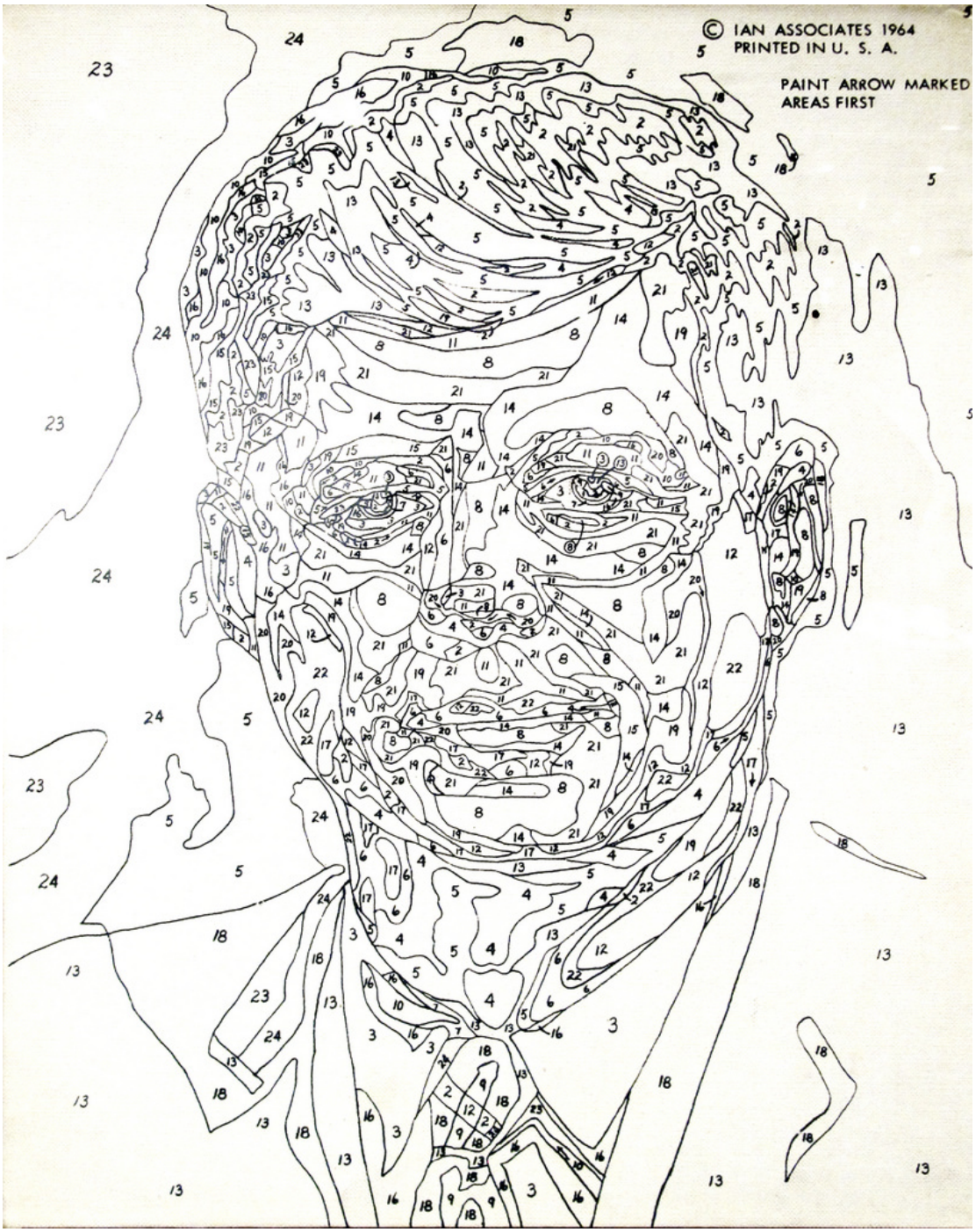


Figure 22.2: A paint-by-number of President JFK, uploaded by Larry Miller (Flickr user larry&flo).

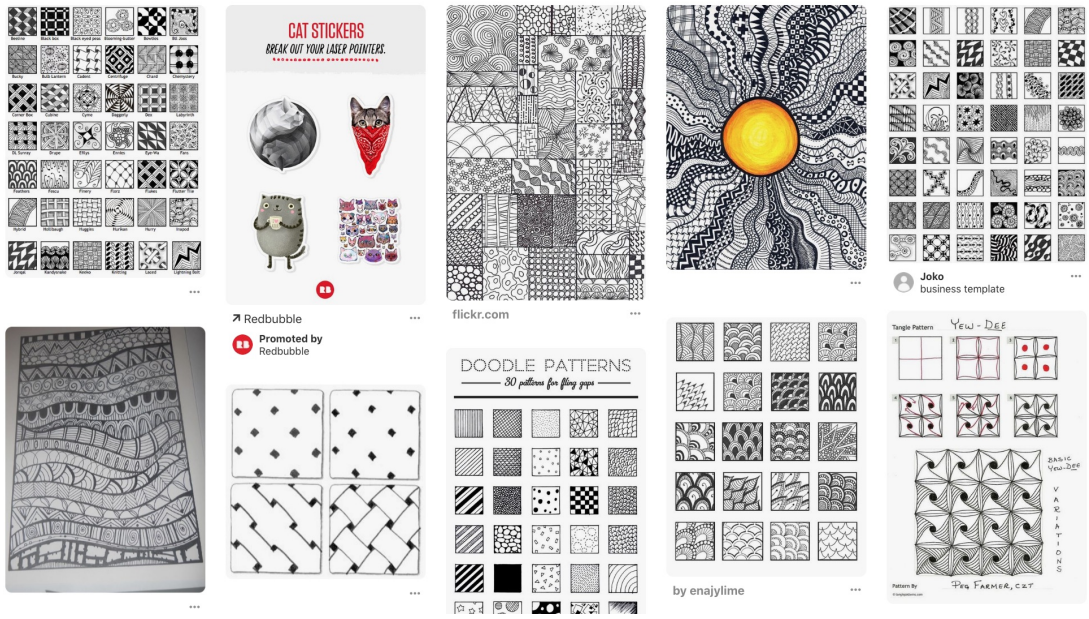


Figure 22.3: *Zentangle* patterns shared by users on Pinterest.

We can further develop this structure by adding computational generativity to any of the stages. In the rest of this chapter, I will list four sub-patterns that riff on the coloring-book structure in order to create **transformation**:

- computational coloring
- traveling to to a new medium
- coming to life
- magic mirrors

22.1 Coloring-in a drawing

One digital augmentation of the coloring-book structure is to use the computer to fill in the regions automatically with a flood-fill algorithm [381]. This raster-based algorithm expands outward from a single pixel, filling in pixels recursively

in horizontal and vertical directions until they run into a different color. Flood fill is a simple enough algorithm to have existed in several early computational drawing tools, including the first *MS Paint*[s141] in 1985[380] and yet *MS Paint* is still used for digital coloring today[1].

MS Paint's version only allows a single flat color to be filled. *Kid Pix*[s113], in contrast, allows users to select one of several patterns, or a random-pattern autoselector, to flood-fill regions with a repeating texture, as in the *Zentangle* I drew in figure Figure 22.4. None of the drawing tools I've so far used in my life (*Kid Pix*, *MS Paint*, or *Photoshop*[s168]) nor the coloring apps I've tried have let the user select fill-shading or texture-direction, though, despite the frequency of those techniques in hand-coloring.

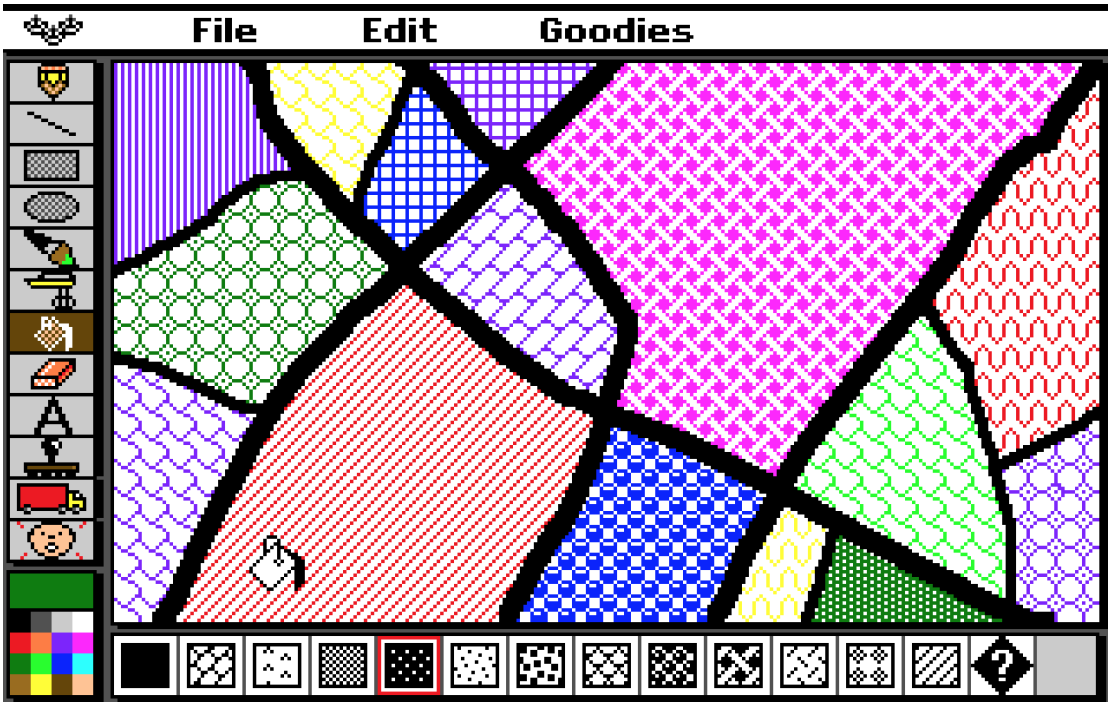


Figure 22.4: A *Zentangle* made with *Kid Pix*'s fill tool, set to randomly select a fill pattern. Made in a few minutes with the Internet Archive's DOSBox version of *Kid Pix* (1991) (`https://archive.org/details/KID_PIX_DOS`).

[1] `https://imgur.com/gallery/zaMqr`

Photoshop has recently extended the "floodfill" algorithm with a "content-aware fill" mode. This mode replaces pixels in a region, like the flood fill tool, but instead of a flat color or a pattern, the surrounding textures are instead extrapolated into a blended image to fill the selected region seamlessly with the surrounding pixels. However, in Figure 22.5, we can see that this does not behave as we would expect for a coloring-book problem.

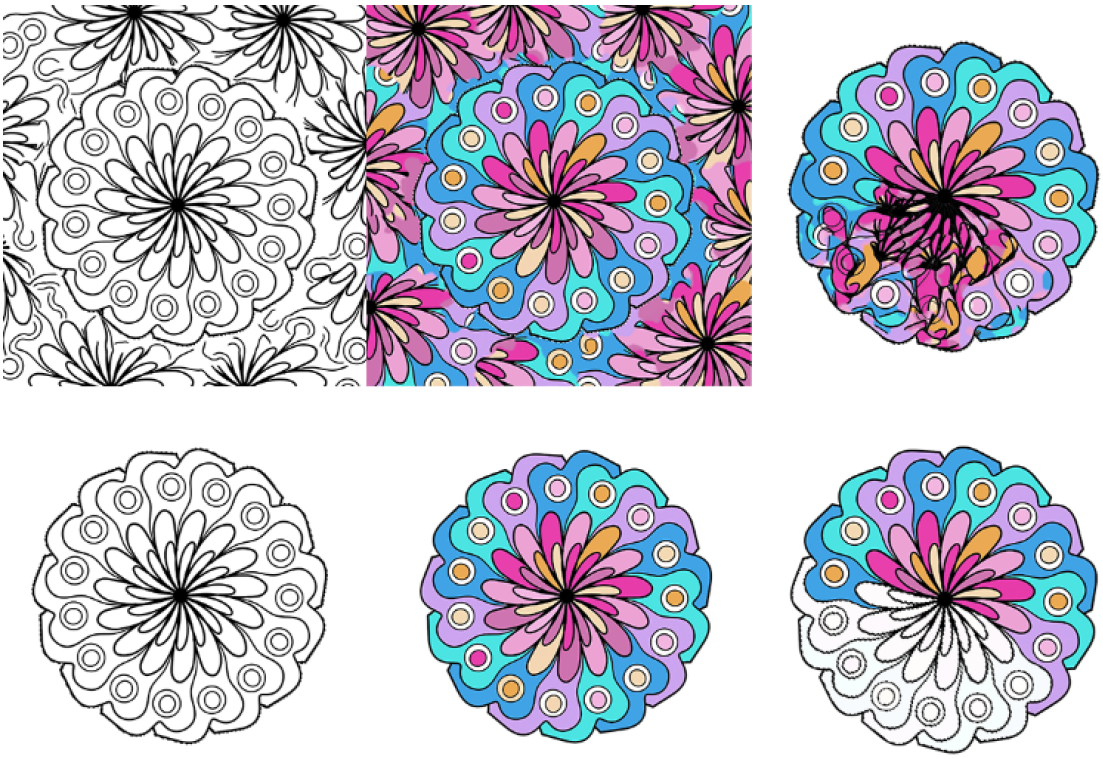


Figure 22.5: Exploring the use of *Photoshop 6*'s[s168] content-aware fill on a coloring book page. While it makes transformational magic across large single areas (first and second column), it can't reason about regional fills (third column) and the image becomes noise. (public domain image from (`https://picryl.com/media/mandala-coloring-page-coloring-for-adults-backgrounds-textures-89ba6f`))

Machine learning seems to show a lot of promise for texture-filling problems; indeed *Photoshop* is exploring using it to improve the behavior of the content-aware fill by considering not only the pixels around the region, but statistical

pixel patterns *in other images of that type*[2]. While no-one seems to be using content-aware fill *playfully*, at least a few projects are using machine-learning statistical methods for coloring.

The most advanced of these (or, of the ones publicly released at the current moment) is *Style2Paints*[s222][250], a web app in which the user uploads a line drawing (all examples are anime girls) and the algorithm automatically regionizes and colors the image. Unlike *Photoshop*'s content-aware filter, *Style2Paints* explicitly advertises itself as a casual creativity tool for novice users:

> "Style2Paints V4 is extremely easy to use! You can be a style2paints expert in 10 minutes!"

It also provides an intuitive coloring-book style interface, where users can gesturally draw colors (as well as lighting hints) onto the image.

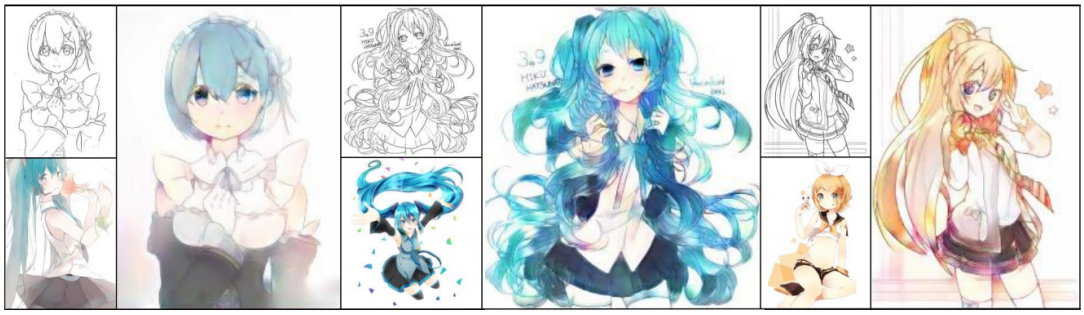


Figure 1: Examples of combination results on sketch images (top-left) and style images (bottom-left). Our approach automatically applies the semantic features of an existing painting to an unfinished sketch. Our network has learned to classify the hair, eyes, skin and clothes, and has the ability to paint these features according to a sketch. More results can be seen at the end of paper.

Figure 22.6: An example of the *Style2Paints* algorithm, almost exclusively used to recolor line drawings of anime-style girls [250].

In Sankloy et al's *Scribbler*[s196] project[341], the algorithm starts with a provided line-drawing. Like *Style2Paints*, the user provides hints for colorization, but can also interactively refine or redraw the sketch.

[2] `https://www.youtube.com/watch?v=rqRY92AcY6k`

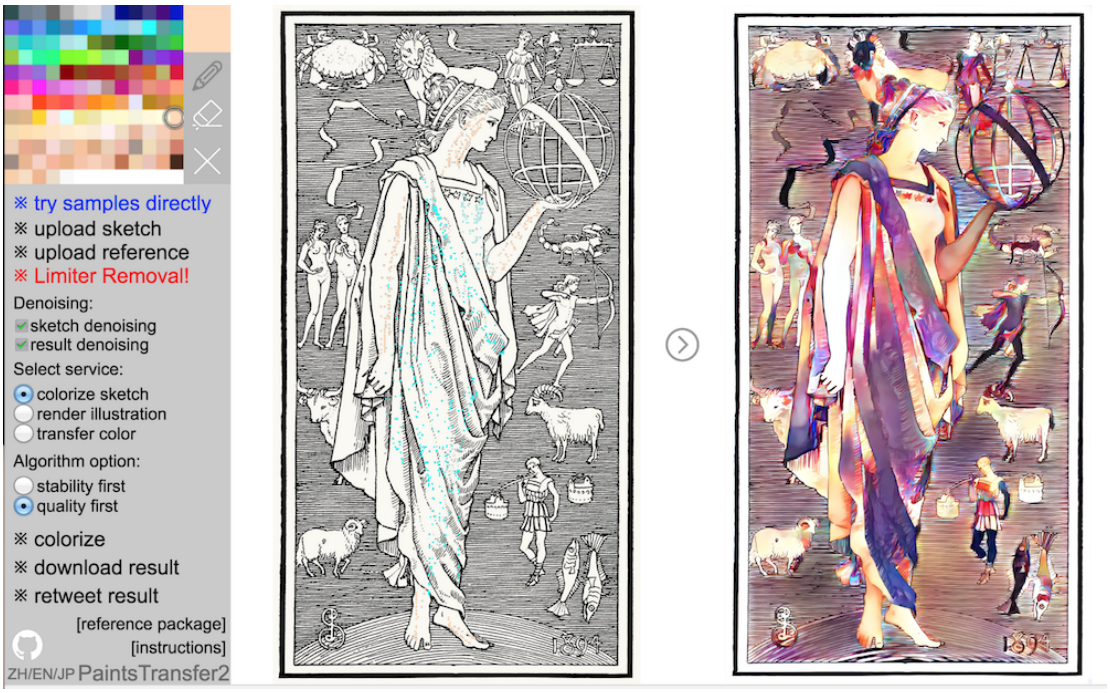


Figure 22.7: My attempt an a *Style2Paints* artwork, using a turn-of-the-century woodcut of Urania and some rough scribblings to indicate color regions. With little guidance, it admirably keeps hair, costume, and detail separate from skin tone.

Style2Paints uses convolutional neural networks (CNNs)[250] trained on a set of tagged anime board posts[3]. It is far from the first to use neural networks to generate or modify artworks. The most famous is likely Google's *DeepDream*[s44], a project by Alexander Mordvintsev[278], to visualize the internal "logic" of image recognition networks. Mordvintsev describes the algorithm:

> "Start with an image full of random noise, then gradually tweak the image towards what the neural net considers a banana (...) By itself, that doesn't work very well, but it does if we impose a prior constraint that the image should have similar statistics to natural images, such as neighboring pixels needing to be correlated."

When the code was open-sourced by Google, many artists and app designers quickly picked it up and twisted it to fit a variety of different interactive modes.

[3] [URL:https://www.kaggle.com/mylesoneill/tagged-anime-illustrations/home]

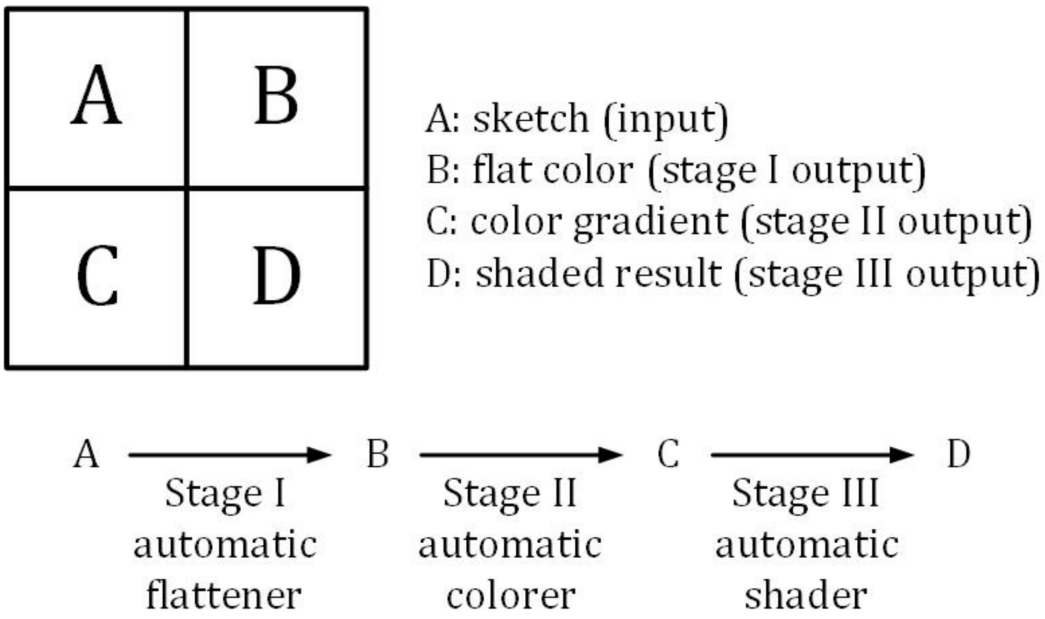


Figure 22.8: *Style2Paints* is one of the only coloring apps that I've seen that explicitly recognizes flat colors, gradients, and region shading as steps in the process, giving users control and access to each one[396].

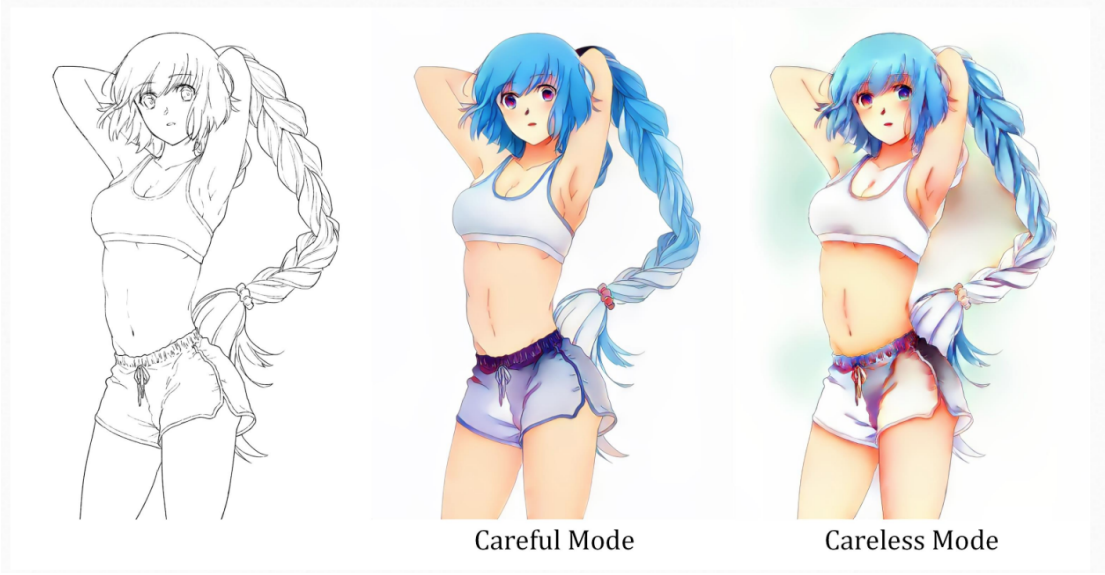


Figure 22.9: In addition to multiple "AI artists" styles, *Style2Paints* can render in "Careful" or "Careless" mode, providing a high-level stylistic control[396].

One particularly helpful property of the *DeepDream* algorithm is that it need not start with *random noise*, it can start with any source image. This property allows

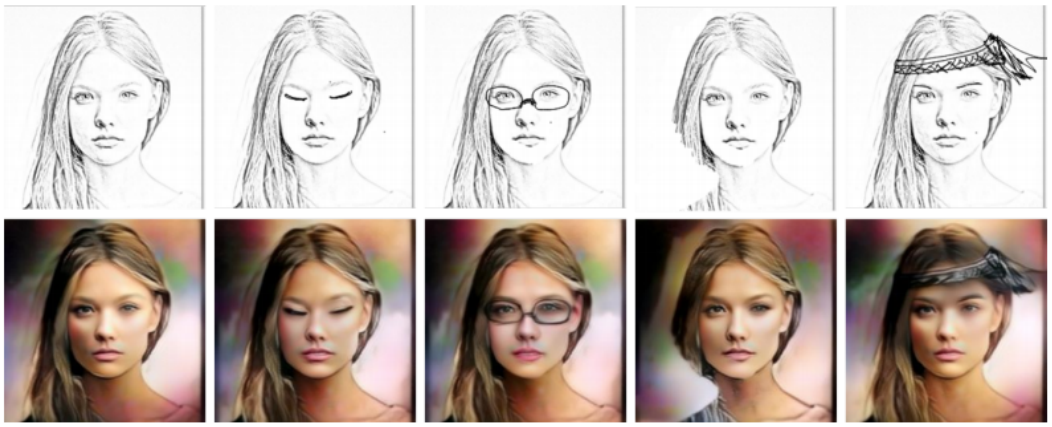


Figure 22.10: *Scribbler* allows iterative modification of the base sketch.

users to combine:

- a source image
- a pre-trained target "style" or optimization (defined by the training data, such as 'pictures of dogs' or 'Van Gogh paintings')

Choosing both creates an image with the *structure* of the input image, and the local statistical properties of the style. Compare this method with the coloring book structure defined above. Even without explicitly feeding the CNN line drawings, the algorithm is "coloring-in" the structure provided by the source image.

Remember that we can replace different parts of the coloring algorithm with humans or algorithms, so this made it straightforward for app developers to create casual creators for users to *DeepDream*-ify their source images, such as the app *Deep Dreamer*[s45]:

"Deep Dreamer takes your deep dreams to the next level with animated creations. Take a clip from your favourite movie, or any QuickTime movie file, and process it to reveal the shapes and objects hiding within it. Process a still photo

and as Deep Dreamer interprets your image it'll generate an animated GIF of the process." [387]

Other systems with use deep-learning to "recolor" images include photo filter apps like *Prisma*[s179], some of which explicitly describe which images were used as the style reference, to engage with the power of socially-significant meanings, like Picasso or Monet (see chapter chapter 24), and others which emphasize that *your* photos can be art, placing the heavier emphasis on the users providing the image. Which is the source of the "art", the user-provided "structural" image, or the selection of the "style" image?

> "Prisma is a photo-editing app that creates amazing photo effects, transforming photos into paintings. Prisma uses artificial neural networks that enable users to make photos appear like they were painted by Picasso, Munch or even Salvador Dali himself." [4]

Most of these *DeepDream* generators run the same set of algorithmic "patches" over the entire source image. However, as we saw in *Style2Paints*' structure, it can be helpful to *regionize* an image to determine, for example, where hair and skin should have different styles. Like a paint-by-number, we are giving the algorithm hints as to which metaphorical paint pot it should use for each region. Alex Champandard's *Neural Doodle*[s148] project makes this explicit: the user marks up a training image with *annotations* using a *MS Paint*-style interface, and uses the same interface to draw target regions. *Neural Doodle* then uses the regions to create differently-trained CNN styles for each region, and applies them to the regions with the same color.[56][57]

There are many automated coloring apps available, especially with the revolution sparked by the *DeepDream code*. My personal favorite, though is *Edges2Cats*[s61][192]. In this online app, users select between the target styles of "cats", "shoes", "handbags", or "facades", though "cats" seems to be the overwhelming favorite. They

[4] `https://prisma-ai.com/`

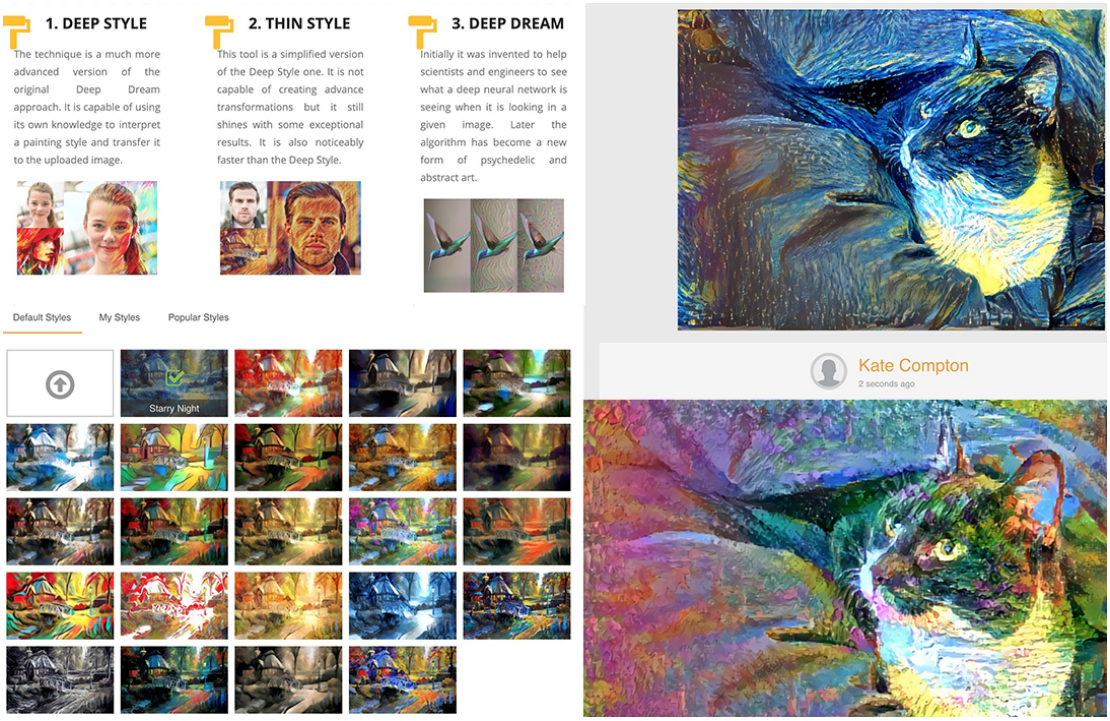


Figure 22.11: *DeepDream Generator*[s46], another *DeepDream* casual creator, takes a picture of my cat and a style to create new art. They don't say what images they've used to train their style networks. [1]

then hand-draw line drawings, which the CNN, trained on cat pictures, deep-dreams into more cat-like coloring. This quickly launched into virality, as users showcased their creativity by creating outlines of pizza, bread, Batman, and more.[101] Why do we see more truly transformational creativity with *Edges2Cats* than with any of the other algorithms? It may be that its simplicity and accessibility, combined with a very short grokloop compared to writing their own CNN generators, made it possible for people to quickly play with it. Perhaps also the ability to combine two very semantically different yet readable sources (Batman and cats) let users create more memorable and unique juxtapositions than the photo apps (combining a portrait photo and common portrait painting style is unlikely to create shocking novelty). User "cleverness" in Edges2Cats is only matched by other creative user experiments with DeepDream, such as the



Figure 22.12: In the top row, the user annotates regions of a style image, then draws a new source image with those same colors to create regions. Below, the network is recursively iterating to match the region style of the style image[56].

transformation of a drug-hallucination scene from Fear and Loathing in Las Vegas[5]. However, that example took expert programming and days of rendering on high-end machines[309].

In concluding this section, we can reflect on the many ways that a computer can color in a drawing, as well as the ways that that process can be influenced or directed by a human user. How does the user interact with these systems?

- take a target photo (*Pixma*, *DeepSketch*, *DeepDream Generator*, *Neural Doodler*)
- upload a line drawing (*Scribbler*, *Style2Paints*)
- draw a sketch (*Edges2Cats*, *Scribbler*)

[5] `https://www.youtube.com/watch?v=oyxSerkkP4o`

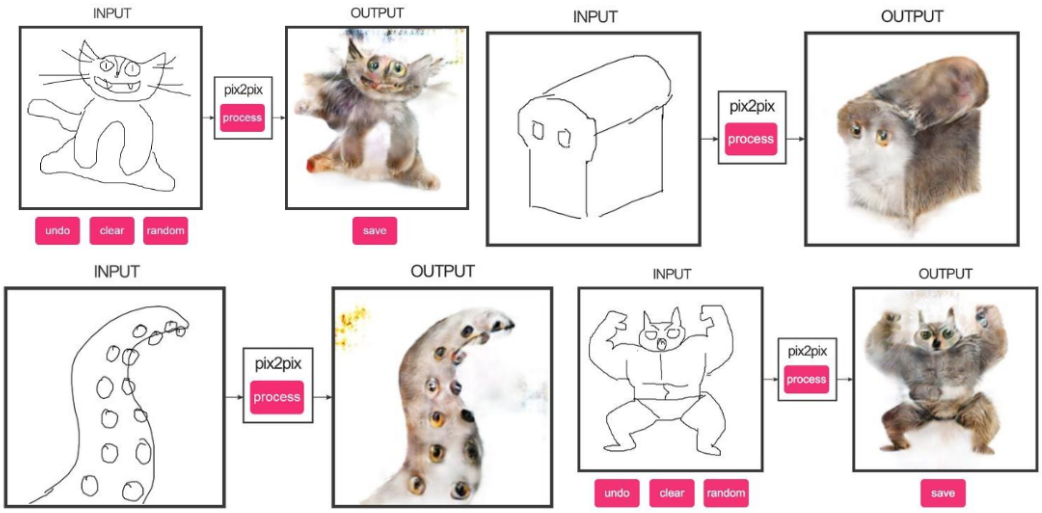


Figure 22.13: Examples of user creativity and playfulness with *Edges2Cats*[101].

- create region annotations (*Neural Doodler*)
- create stroke or point annotations (*Style2Paints*, *Scribbler*)
- upload or stream a video (*DeepDreamVideo*, as well as some apps)

We can see that most of these limit the interaction to the structure image. Oddly there don't yet seem to be any creators where the *style* image is dynamically created or modified (except for *Neural Doodler*'s annotations).

In the next two subsections, I show how drawing and coloring-book metaphors can have transformational properties only when the user is *finished* with the drawing. That is, the first stage is a normal, non-augmented drawing/coloring process, and the casual creator magic happens only after that is complete.

22.2 Traveling to a new medium

One example of this is when a user's artifact "travels" from one location, or medium, to another. 3D printing is an instance of this, as are works like *Barbie*

Fashion Designer[s10], a 1996 game where users digitally color in patterns in a fashion app, then print them out on fabric-paper sheets using an inkjet printer to create functional Barbie clothing. In 1996, even the ability to color on a computer was a novelty, so as a girl, I was fascinated by the potential ability to bring a computer art project into the real world in an unexpected way.

The ability to create physical artifacts (3D printed objects or printed fabric) can be expensive, so several apps work in the opposite direction. Coloring a piece of paper is a very accessible activity, the supplies are readily available and easy to use. In itself, coloring lacks the power-multiplication property of a casual creator, so after a paper is colored non-digitally, it can be scanned via a web camera (or other scanner) into a digital app, where some interesting transformation occurs.

A recent spectacular example of this is Disney Research's "Live Texturing of Augmented Reality Characters"[253][s51], a project where children can color normal-looking coloring book pages of cartoon characters. The pages are then scanned, and the textures remapped onto the UV map of a 3D character model, making extrapolations about what should go onto the parts of the character (the back, etc) that are not visible in the original coloring book page. The users then see their drawing "come to life" as an animation. This works well as a casual creator: the users have no control over the shape or animation of the resulting character, but just coloring within the lines of the app can still create novel-feeling characters (e.g. pink elephants, Superman suits).

This color-then-scan pattern is useful for interactive art for children, as seen in two different artworks by TeamLab as part of their Future Park installation[405]. *Sketch Aquarium* and *Sketch Town*[s208] allow children to color in pictures of fish or vehicles, respectively. These activities can accommodate many users at a time, as table space, crayons and paper are inexpensive and durable, and the coloring

activity takes up most of interaction time. The time to interact with the digital system is limited to the moment of scanning, which doesn't require the children to use a UI (less potential for broken or sticky equipment), just put their drawing on a designated platform where it can be scanned. After scanning, the drawings are "teleported" into the digital world projected on the walls, where their drawing is mapped onto a digital object and joins the other objects in the world simulation. The child, or their parent, gets to keep the physical artwork, but also sees the digital version semi-permanently on the wall of a museum or gallery, interacting with other children's art. There are numerous boundary crossings here:

- from physical to digital
- from 2D to 3D
- from in the child's hand to the wall
- from static paper in a real world to animating in a simulated world
- from a piece held by one child to one that is interacting in a social space
- from a casual novice drawing to a feature on a museum wall or professional art space

All of these combine to make this a very special experience which the children and adults seem to appreciate. When the "mediums" that the drawing is traveling between are not *as different* as these two, the jump can still be notable. In the Monterey Bay Aquarium's *Jellies Experience*[s106][438], the interactive component allows visitors to draw jellyfish bodies on interactive tablets, and then press the "Set It Free!" button to move it to a bank of monitors on the wall. The jelly is animated as though it is swimming from the tablet to the wall monitors, taking advantage of the fixed arrangement of the installation.

These apps rely on the fact that the user is coloring a *known* structure, not drawing "new" creations that they system does not know how to animate. Coloring apps, in all of their forms, rely on "coloring" as an activity that has very clear boundaries and expectations.[6] Casual creators can make use of that limited possibility space to create augmentations and generative pipelines that work within those expectations.

In the next chapter, we will see more examples of "magic transformations" that take digital creations into non-virtual space. The boundary between digital space and "the real world" is a clear demarcation, and everything that happens in an app seems to happen in a "magic circle"[201]. If we take an artifact that only exists in the magic circle, and bring it into physical reality, a boundary has been transgressed, and it feels like something extraordinary has happened.

[6] "Coloring within/outside the lines" is a metaphorical expression to indicate conformity or rejection of it.

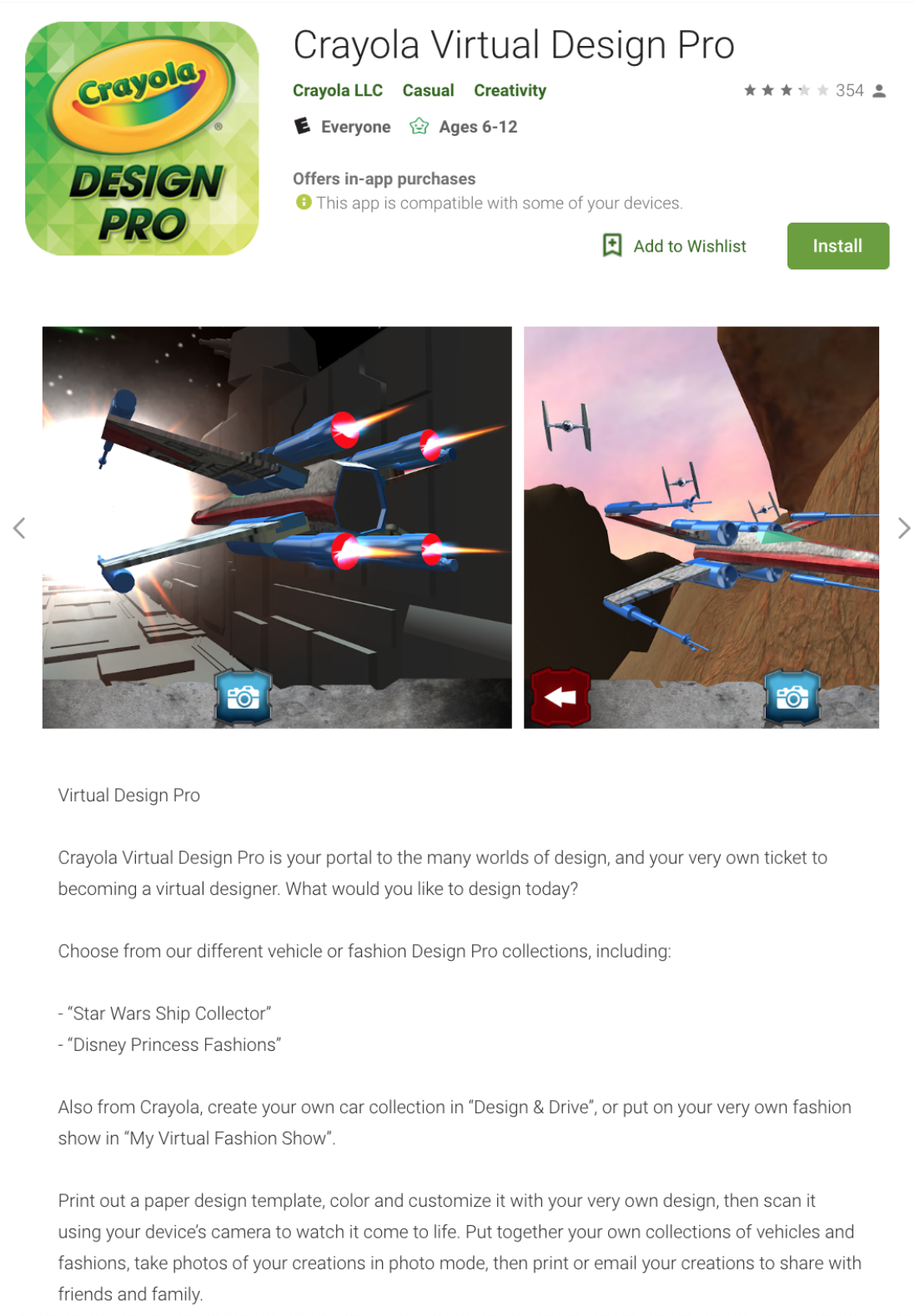


Figure 22.14: Crayola also has a coloring-to-3D app, but its very low app store reviews (and angry posts) suggest that this technique, while promising, can lose its magic quickly if the camera recognition isn't easy or reliable.

Chapter 23

Pattern - Make It Real

In this dissertation, we have seen many digital casual creators, and several physical ones (Spirographs, pottery wheels, Knitting Nancy). In the previous chapter, we saw physical objects, like coloring book pages, which travelled to the virtual world to become part of a generative pipeline. In this chapter, I will explore an interesting new trend of casual creators, which is the ability to take a virtual object and turn it into a real one.

23.1 Driving interest in personal fabrication

The past decade has seen wild optimism about the potential of 3D printers. At the beginning of the 2010s, it seemed like every article about this technology promised that they would soon be in every home, printing out consumer goods and replacement parts[14]. The technology improved, the devices got cheaper, but somehow that future never arrived. 3D printers were in schools (at least, those where a STEM-hopeful school board funded one), but underutilized. They could print out pre-created models and kits[44], but customizing the models required expertise at Autodesk 123D Design or Solidworks 3D CAD, professional design

software that the school might not have a license for, a skilled teacher for, or time to teach the students. 3D printing has created a revolution in manufacturing, but only in spaces that can perform customization by expert users (like specialist prototyping firms) or narrow design domains with specialist tools, like dental implant manufacturing.

Hobbyist designers of 3D-printables found an outlet on Thingiverse, an online platform for 3D printable designs. Amateur designers could upload designs, gaining appreciation, likes, or influence (all the social feedback from chapter 20). People who downloaded the objects got a design that they didn't have to troubleshoot themselves, but was a more niche or clever object than they could find available in a store, like 3D printableMedieval armor for Barbies[1].

Other machines arrived to fill out the ranks of the so-called personal fabrication movement: CNC-embroidery machines, laser cutters (prohibitively expensive and potentially dangerous), CNC paper-cutting machines[2], handheld CNC routers[3], as well as toy devices like Evil Mad Scientist's *AxiDraw*, *Egg-Bot*, and *WaterColorBot*[s238], CNC controlled rigs that moved pens or brushes. Mostly, like the 3D printers, these relied on libraries of designer-uploaded sample patterns to supply content to the users who were not proficient in design software. A few companies and power-users released custom editors or plugins for their gnarly data format[4]. Others created pipelines to let people use real-world tactile approaches, like 3D scanning to "import" physical objects[432] or Glowforge's clever idea to add a camera to a laser cutter, so that it could follow a hand-drawn line.

These systems often have much more stringent requirements for "printable"

[1] `https://www.kickstarter.com/projects/191197566/faire-play-barbie-compatible-3d-printed-medieval-a`

[2] `https://home.cricut.com/`

[3] `https://www.shapertools.com/`

[4] `https://wiki.evilmadscientist.com/AxiDraw#Third-party_software_to_drive_AxiDraw`

artifacts there are for artifacts like usable game content or printable drawings. A 3D printable object must be a "watertight" mesh, with a highly-specified format of edges and faces that can be used to computationally-determine an "inside" and "outside", in order to create the horizontal "slices" that are used in the 3D printing process. In addition, the printed object itself is subject to a variety of physical forces during printing, and cannot have significant overhangs (or the overhands must have temporary structs connecting them). Different materials, including multi-material prints, will have even more requirements, to the point that Shapeways, a popular 3D printing service, has multiple-page "design guidelines"[362] for each of its dozen available materials. All this is in addition to the physical constraints of the final object: whether a 3D bracelet is comfortable and fits over a wrist, or the links and fasteners bend correctly, and the material is light enough to wear but not breakably thin.

Nervous System[s1465] is a company that attempts to solve some of this by creating a number of interactive generative design tools on its website that can be used to design (and order) 3D printed objects. They offer a number of different kinds of tools, each with a different generative pipeline (often using Voronoi cells and surface tension to construct meshes).

There is interest in developing uses for 3D printing (and other fabrication techniques) to drive new users' engagement in the technology, and hopefully seed STEM interest in students using the technology. Some projects use these technologies in an *autotelic* way, using the printer to manufacture game pieces or game rewards.[402][32] However these items are pre-authored content, and the experience of using the software is not about moving through some possibility space, so these are not casual creators.

[5] `https://n-e-r-v-o-u-s.com/labs/`

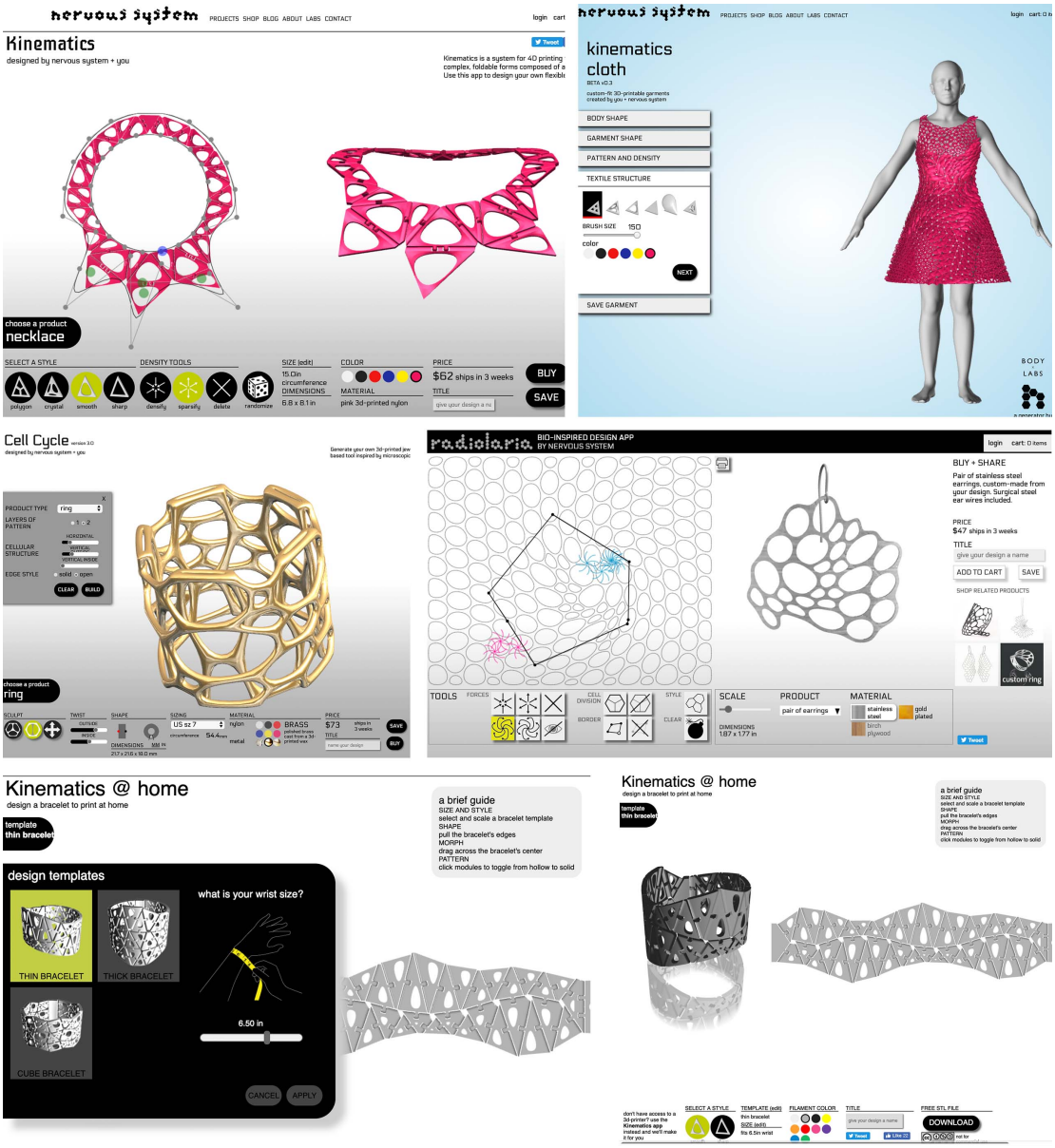


Figure 23.1: Several interactive creators for 3D printing by *Nervous System*. *Cell Cycle* creates a double-looped ring or bracelet, *Kinematics* creates a hinged Voronoi necklace. *Kinematics Cloth* creates a 3D printable dress. *Radiolaria* create a flat celled object which can be used as earrings or a trive. *Kinematics-at-home* creates a jointed bracelet that can be printed on an inexpensive home 3D printed (unlike most of their designs which must be ordered from a professional printer).

23.2 Casual creators for "personalization"

Personal fabrication formats, as an output, are no more or less suited to casual creators than any other type of artifact. Instead, it is the conditions of the current moment that make them so necessary: the possibility for everday people to *manufacture* custom objects coincides with a lack of support for those same people to *design* them. The other critical part of the equation is to determine *why* people would want to get custom objects. "Customization" was waved about like a reason in itself, but while people knew why they wanted a custom photo of themselves, they did not know why they would want it[67].

Casual creators will sometimes add modes to export an artifact into the real world, in the assumption that if a user has designed something, then it is *custom* and that may be enough motivation for them to pay to bring it into the real world. *Spore*[s215] allowed users to print out exquisitely brittle color versions of their creations, a service that was tested in 2006, even before the game shipped[8]. *Let's Create! Pottery*[s123] is a pottery wheel game/app for mobile phones and tablets, but users can order a 3D print of their artworks. *Petalz*[s167], which is covered in many chapters, had a 3D flower-printing service as well. The *Simpsonizer*[s2059] that you will read about in chapter 25 suggested that users print their Simpsonized self-portrait on a mug or t-shirt. There are a number of different 3D-printing custom doll companies, perpetually coming into and fading out of business. The latest is primarily an online character creator *Chibify*[s3310], which allows you to also print out your Chibi portrait. Why do people want these objects? The "magic" of

[6] `https://all3dp.com/2/3d-print-from-photo-how-to-do-it/3Dprintedphoto`

[7] Physical photos are stackable, collectable in large numbers, easily-displayed in established frames, and favorites can be selected from literally hundreds of candidates.

[8] `https://spore.fandom.com/wiki/3D_Printer_Service`

[9] `https://disneyandmore.blogspot.com/2008/09/are-you-ready-to-simpsonize-yourself.html`

[10] `https://www.chibify.com/`

3D-printing has been one of the reasons, like how *Spore* linked the technological advancement of its creatures to the advancement of 3D printing for marketing purposes. But this magic seems to fade fast, and few people are recreationally printing objects. What other motivations are there?

23.3 Social affordances of "real" objects

Since this dissertation has a chapter on *why* people create, and in what social contexts(chapter 4), we can begin to answer some of these questions.

One of the least-demanding personal fabrication formats is custom printing, which only needs a color image, and can then be printed onto a mug, t-shirt or roll of fabric. *Broderie*s[22] is a long-evolving project in which I make patterns by scripting particles to move down a "lane" of an image while leaving behind a trail, then repeating that lane symetrically until it form a pattern. There have been a lot of versions of this (including one where I animate through the parametric vectorspace of potential patterns[11]). But of all of my generative works, this is the work that I have most often tried to physicalize. In August of 2018, I generated a few thousand parametric fabric patterns, curated the best, then uploaded[12] them to Spoonflower (a fabric-printing service), printed out 12 yards of fabric, for 54 uniquely generated scarves. I then showed them at conference, and when the conference was over traded them away to conference-goers in exchange for other art. As a casual creator, the original software system was underwhelming, and barely interactive for users who weren't me. But as a social intervention, people were interested in the uniqueness of the scarves, shuffling through the large pile, suggesting names, and personal-shopper-curating the collection for newcomers (it

[11] `https://vimeo.com/259415883`
[12] `https://www.spoonflower.com/profiles/galaxykate`

was a fun group activity to find the scarf that "matched" a potential customer).

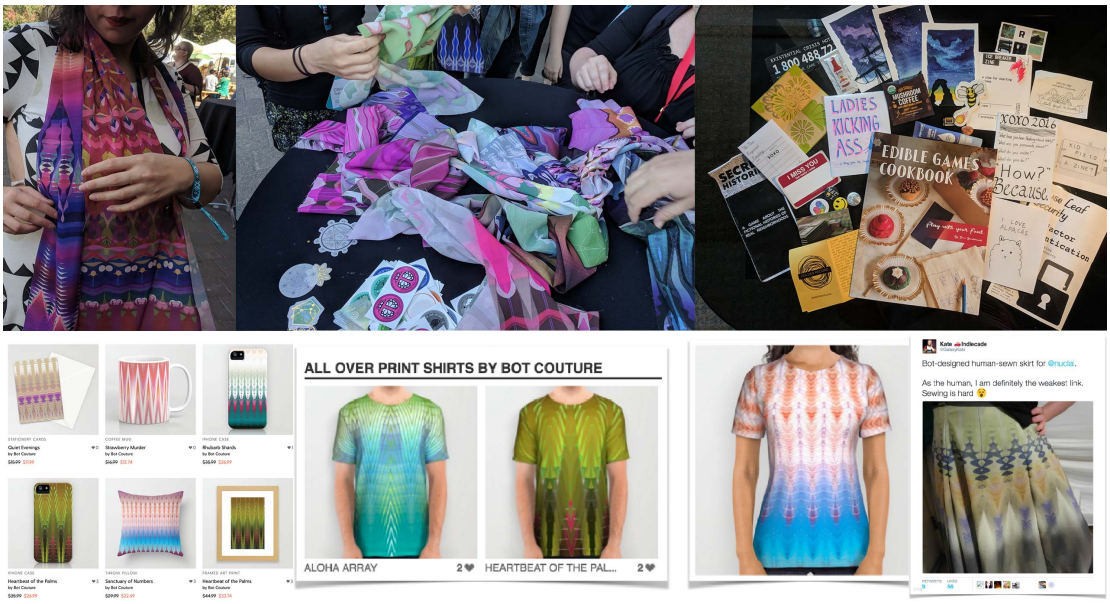


Figure 23.2: (top) *Broderie* scarves being traded at XOXO for a variety of personally-created goods. The physicality and limited-edition nature of these scarves seemed to make them more interesting than other experiments in physicalizing *Broderie* patterns, like making a digital art projection or printing them on tshirts for sale (`https://society6.com/galaxykate`). Users also really enjoyed that I was trading the scarves for other hand-made objects, linking them into the community of makers at XOXO.

That *Broderie* experiment was an example of a generative object *participating* in a social situation in ways that a non-digital object could not. In *Sky Knit*, machine-learning artist Janelle Shane fed knitting patterns into an LSTM (pattern-learning AI) that spat out new patterns with generative names[361]. In itself this would have been amusing, but it was adopted by the knitting community ofRavelry[13], who began knitting, debugging, repairing and cataloging this strange generative space. *Sky Knit* is not just a Janelle Shane project, it became a social creativity exploration:[361]

> "If you feel so inspired (and don't mind the kind-hearted yet vigorous swearing), join the conversation on the LSG Ravelry SkyKnit thread

[13] `https://www.ravelry.com/`

> - many of SkyKnit's creations have not yet been test-knit at all, and others transform with every new knitter's interpretation. Compare notes, commiserate, and do SkyKnit's inscrutable bidding!"

The most common form of customization is personalization, and one of the most common forms of that is to put one's photo or name on a piece of merchandise. This can be done for many social reasons: to wear clothing showing team affiliation (church groups, bachelorette parties, or fun runs), to show pride in the self (monogrammed apparel), or pride in a creation. We live an environment where almost all production is done on an industrial scale, so to see one's own graphic on an item of clothing is a signal of its importance, even if that perception is a holdover from prior industrial conditions. Like "seeing one's name in lights" on a movie marquee is a metonymy for fame, turning something into a physical object is a symbol of its permanence and importance. So it is unsurprising that casual creators for fabrication often take as their input **photographs**[231], text (and names), and broad signifiers, especially those that carry a lot of social meaning[14].

Early on in my grad-school career, I had two different consulting/contracting gigs to make casual creators for two 3D printing startups. For *PendantMaker*[s164], the goal was to make a 3D printable object that anyone could "draw" into existence, without creative paralysis, so that the company could demo the customization-magic of 3D printers "live"[15]. It was very successful as a piece of software: it was usable, fun, and made objects that were ideally suited for very-rapid 3D printing. The company was not ultimately successful, but the pendants didn't magically save 3D printing. They were cool, but not personal:

> "At the design bar you can sit down and design and print a pendant using their Pendant Maker App. The app allows anyone regardless of ability to doodle a line and the app takes the doodle as an input and

[14] `https://www.spreadshirt.com/create-your-own`

[15] `http://blog.inventables.com/2013/04/3d-printer-experience-retail-store.html/physicalstore`

> makes it into a pretty cool looking pendant. You can then use the sliders to adjust the size and shape of the different design elements. When you're happy with your design you can print it out. Pendants take between 20-50 minutes to print out. If pendants aren't your thing they also have a 3D scanning station where they can scan you and print you out!"

The other project, *Exonumia*[s71], was to make a casual creator for commemorative coins, with user-provided graphics and text extruded into a 3D printable object that could be printed at home or sent to a centralized print service like Shapeways to be printed in metal. Commemorative coins (known by the numismatic term "exonumia") are custom non-monetary sculptures in coin-form. While this project failed for technical reasons[16], I think it might have been a better fit for the social needs of fabrication. Remember above that fabrication is a way of saying that something is permanent and worthy of physical recognition. Commemorative coins, unlike t-shirts which are tainted by functionality, are a pure expression of that. They also have physical properties that make them collectable in large numbers, stashed in a geocache, printed cheaply or expensively (to signal low or high regard), or used as asecret membership signal[17] that can be carried in a pocket. Even if the creator has been less juicy than Pendantmaker, the social participation of the artifact in a creativity environment may have made it a more compelling experience.

23.4 Artifacts as memory

Perhaps the best use of a casual creator for fabrication is to memorialize the experience of the creation. We can see this in casual creators that process snapshots of the user's actions into an animated GIF, for the user to save, post, or

[16] Ask me about mathematics for offseting curves!

[17] `https://99percentinvisible.org/episode/coin-check/`

reflect on their experience. Games like *Threadsteading*[s233](a territory-acquisition board-game whose moves are recorded as stitches on a CNC embroiderer)[10] and Loominary (a choice-based narrative who's choices are selected and recorded in threads woven onto a loom[397] are two works where there is an external autotelic experience (a competitive game or an interactive narrative), but by experiencing that, the player is creating an artifact. Systems like these have properties in common with other constrained creativity works, which are sometimes about noticing and recording experience. Lea Redmond's book *Knit The Sky*[s117] encourages users to "Record the beauty, emotions, and experiences of everyday life – not in your journal but with your knitting needles" by adding a line of yarn each day that records the color of the sky.

When done without consideration, fabrication-as-record does not ask the user to experience anything, or make creativity choices in how they record that experience[18], as in apps that purport to turn a brain wave or sound wave into an object. It may be no more relevant to the experience than a QR code recording the same data. When done well, it can create a record of many individual creativty choices participating in a physical space, as Roman Ondák's *Measuring the Universe*[290], a "generative" gallery show where users measured their height against a wall, drew a tick mark with their name and date, slowly building up a unified shape of gallery-goers presence that still carried a legible signal of each individual's experience in the piece.

[18]`https://www.dailymail.co.uk/sciencetech/article-2354821/The-neuro-knitted-scarves-printed-exact-image-brains-activity.html`

Chapter 24

Pattern - Signifiers Add Power

Casual creators let users create artifacts using generative methods. We have so far seen many examples of different generative methods that create content (chapter 21, chapter 6), but it is noticeable that often the content that is made is abstract or geometric, or creates variation on a single "type" of thing (like a radially-symmetrical flower, or vases). Are there Casual Creators that create things with different semiotic meanings? Or that allow users to play with semiotic meaning? After all, casual creators are all about leveraging algorithmic power to turn small actions into powerful output, we should also consider how to leverage **semiotic** meaning to do the same.

Semiotics is the use of signs, the oddly human behavior of taking one thing (a "referent"), representing it with a "sign", and then using it to connect metaphorically to other apparently-unrelated things:

> "a sign is any physical form that has been imagined or made externally (through some physical medium) to stand for an object, event, feeling, etc., known as a referent, or for a class of similar (or related) objects, events, feelings, etc., known as a referential domain."[354]

These referents can be "denotative", like using "cat" or a picture of a cat to refer to a cat, or "connotative", where we stretch the connection to apply to some-

thing related: "I'm such a cat" (to imply liking warm spots, physical attention, and not respecting personal boundaries) or combine it to create new signs ("catnap"). Signs can be "icons" (representations of the referent), "indexes" (representations that "point" to the referent, like by showing something related temporally or physically to the referent), or "symbols", culturally agreed-on pairings between referents and signs, like the word "word", or an American flag.

All this Semiotics 101 paragraph is to say is: by using signs we *summon* their referents. Whether we do this, directly, or by analogy or metonymy, is an open choice. We could choose a red and yellow scarf, a lightning bolt scar, the word "expelliarmus", or a photo of Hogwarts to conjure up visions of the Harry Potter universe. And by summoning that universe, any nearby signs are read in relationship to that referent (nearby cars and rats become suspected of harboring magic powers). The stronger the sign, and its associated referent, the stronger the shadow that is cast on nearby signs.

24.1 Semiotic Theming

We can see examples of theming in casual creators where the activity has not changed, but the user is acting *within* a story or fantasy or referent. In the Downton Abbey and Outlander coloring books, the possibility space is not very large: one can color them in realistically, or expressively, according to the show's colors, or picking out non-canon colors for the dresses and hair, for example (maybe the Abbey would look better in magenta).

Likewise, the "Seedling Disney" craft kits of Figure 24.2 are simple creativity kits, but remind the user of this object's part in a larger story. While I wouldn't classify these kits as true casual creators (they have only the most simple forms emergence or generativity) they are creativity activities where *every action taken*

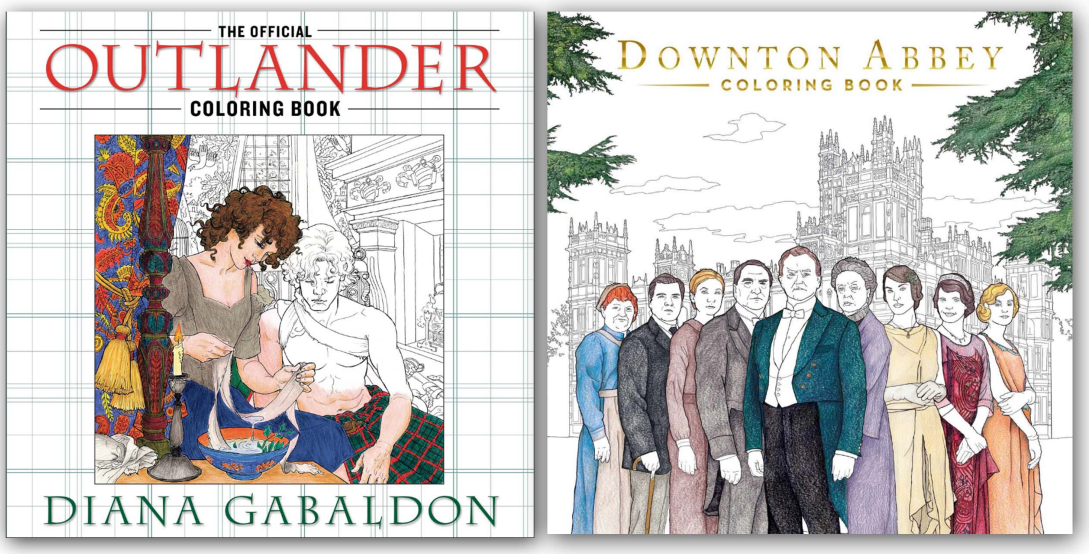


Figure 24.1: "Downton Abbey" and "Outlander" coloring books, two media properties that share a large marketing-demographic overlap with coloring books.

is uplifted by its association with the movie[1]. The child is decorating a hairbrush *just like Rapunzel would.*



Figure 24.2: Add rhinestones to Rapunzel's hairbrush, excavate grubs from a tree stump, design your own mermaid tail[443].

[1]...even when the activity is part of the larger story universe of that character, not of the original movie itself

Participation in a grown-up fantasy is part of many traditional crafts;[154] many Victorian crafts were considered practice for womens' future householding duties. We see a bit of this in childrens' cooking games[s26s27s130], although I was most surprised to see the *truly massive* number of Indian wedding casual creators.

From one with more documentation than most[s99], this casual creator (narrow socket-based selection and distribution creative levels, interspersed with task-fulfillment levels) explicitly aims to teach about the cultural significance of different parts of a traditional wedding.

> Indian weddings are known for their rituals, style, rules and traditions. Here in this Indian Wedding Part 1 game you can enjoy all the wedding preparation activities from head to toe and also learn about Indian makeover, culture, style, fashion, rituals and ceremonies in weddings in India. So Get ready take a part into Indian wedding and enjoy each and every single ritual with the beautiful Indian couple in this Indian Wedding Salon game. Rings are the most important accessories for your engagement. In Indian culture, they are normally worn by married men or women as it is a symbol of engagement. So select your favorite wedding ring in this Indian makeover game and design it using your creativity. Invitation card is the first step of wedding So let's prepare a beautiful invitation card in this Indian wedding game. Select unique patterns, colors, symbols from the different options and insert your favorite text. Once your invitation cards are complete then sent it to guests and relative forms an integral part of the ritual. Mehndi holds a lot of cultural significance in Indian wedding so select a unique style of Indian Mehndi designs for hands as well as legs to make her look more beautiful. It's so much fun in this dress up game.[2]

This casual-creator fantasy property can be seen in the *Sims 2*[s206], which inked a deal to showcase Ikea and H&M content, as well as other content in games, to give users a way to engage with the fantasy, as well as the creative possibilities of that fantasy, like how an apartment would look with an unlimited Ikea budget.

> "It really is the ultimate in brand engagement," said Steve Seabolt, VP-global brand development for "The Sims," an Electronic Arts la-

[2] `https://www.amazon.com/GameiMake-Indian-Wedding-Part-1/dp/B0778B9V4Y/ref=sr_1_30?s=mobile-apps&ie=UTF8&qid=1547609213&sr=1-30`

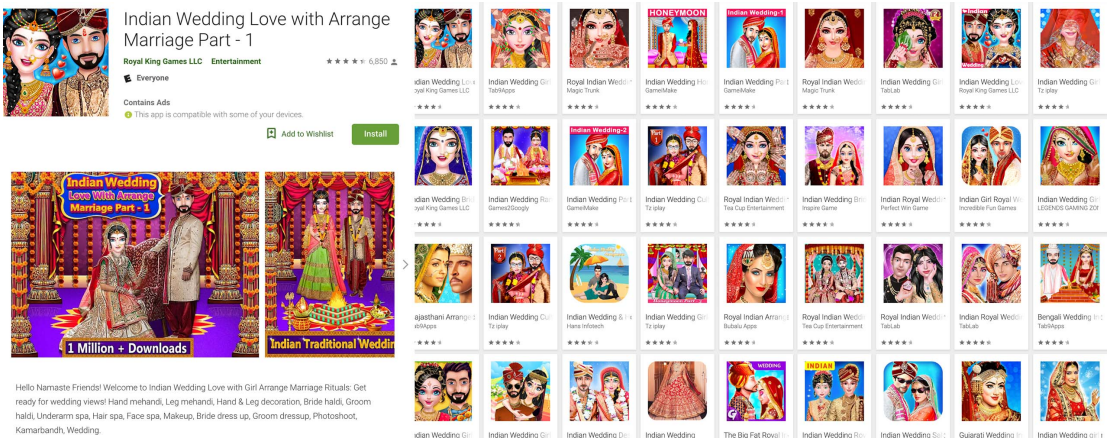


Figure 24.3: So many Indian wedding apps. More that western-style wedding apps. I'm not sure why these are *this* popular, and nothing online has been written about them (in English, that I've found, at least).

> bel game. "It really is a way to bring a product or a service into your virtual life, become familiar with it, play with it and, hopefully, incorporate it into your real life."[449]

This property can be used to *subvert* the expectations of the genre as well, as in the needlepoint Lara Croft portraits from Becky Schaffer[206]. Lara Croft is a referent for the *Tomb Raider* games, an experience about looting, adventure, and violence, but is rendered amid a bucolic landscape in a medium that is a shorthand for passive, well-bred ladies. As yet there are few digital casual creators that play with this type of tongue-in-cheek juxtaposition, though there are any number of foul-mouthed coloring books and needlepoint kits. But as yet there is no embroidery minigame in Fallout or quiet cooperative quilting mode in Fortnight, or other such incongruous juxtapositions (to their detriment).

24.2 From K'nex to Lego to Playmobil

Semiotic properties can be *very* potent for generative methods. We can look at a simple generative method like the *Spore Creature Creator*'s[s216] metaball method.



Figure 24.4: Recontextualized Lara Croft needlepoint by Becky Schaefer (from [206]).

This takes the curve of the spine, and extrudes into a tube of chewing gum. Is this a creature? No, it just looks like a tube. However, when a spine shows up in the rollover, we compare tube and tube-with-vertebra-in-it and begin imagining it as a creature, but only a little. Abstract vertebra aren't the strongest and most familiar of signs. If a mouth is added at the end, that semiotically-heavy piece *emphatically* announces that it is a creature (and which way it is facing)[3]

Without the semiotically-heavy pieces in *Spore*, we would have had to create a much more sophisticated editor for players to create "creature-like" creatures. Instead, just adding eyes, a very small action, let them give "life" to our tube of chewing gum, a very big change in outcome. However, the "chewing gum" of metaballs creates the freedom to create arbitrarily weird structures on which to hang their eyes and mouths. A casual creator that want to maximize "creativity"

[3]On Spore, I remember someone mentioning that this framework had been inspired by topiary techniques. A topiary bush grows the necessary form, but not detail, so inorganic sculpted pieces can be stuck in to provide it.

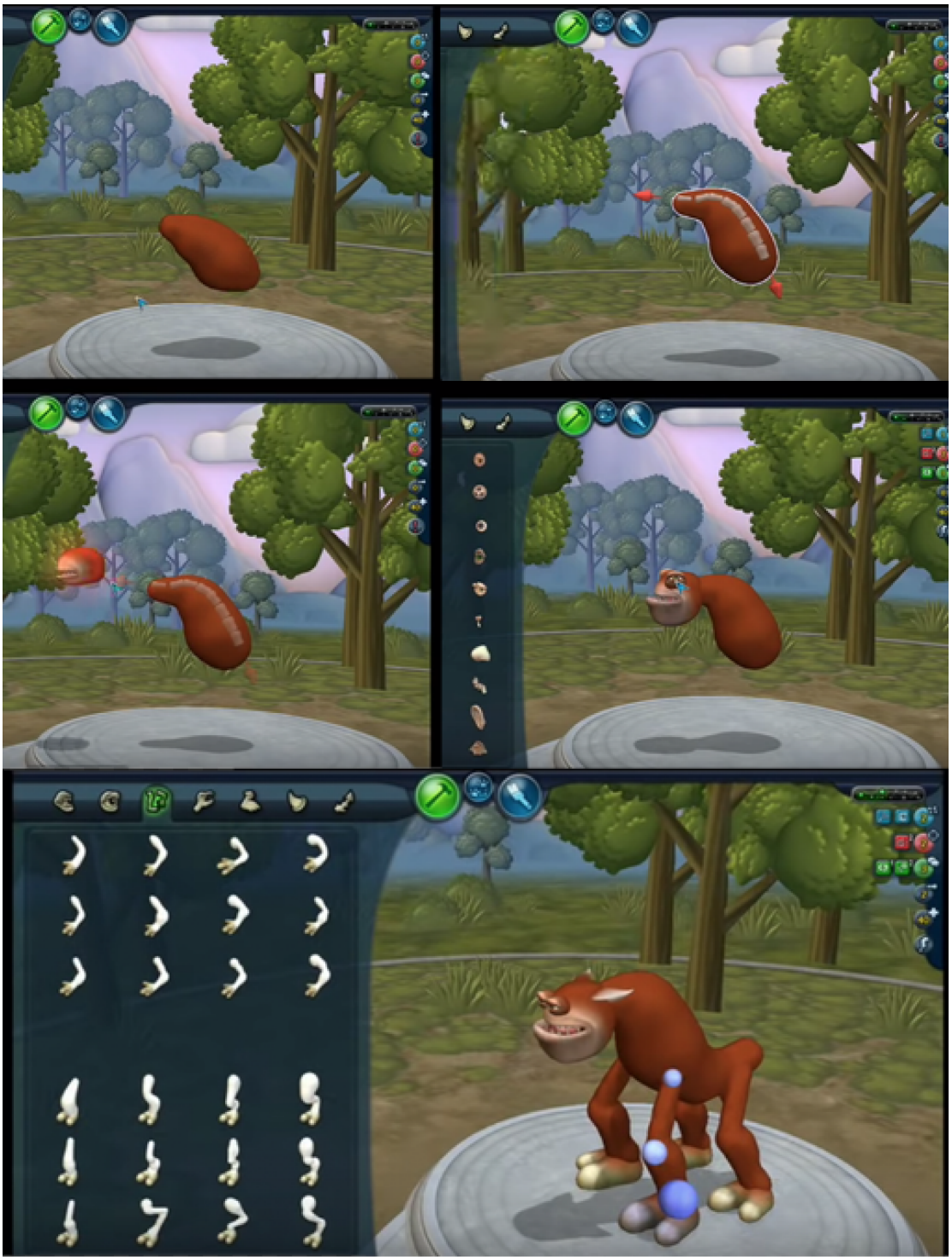


Figure 24.5: A *Spore* creature becomes more creature-like with the addition of eyes, mouth and limbs. ([386])



Figure 24.6: Anything with eyes on it is a character.(photo used with permission of Marie Huynh).

should have two levels of components:

- "chewing gum": a generative method that allows for structural variation
- semiotically-heavy pieces: components that carry lots of semiotic meaning (eyes, rabbit ears, emoji, Harry Potter glasses, pirate flags)

This is what I'd call the **Lego Principle of casual creators**. *Lego*[s122] has specialized in branded tie-in sets in recent years, with most of their website[4] consisting of branded content (both in-house like Ninjago and out-of-house brands like Harry Potter). These sets often have **semantic pieces**: Harry Potter's scarred

[4] `https://www.lego.com`



Figure 24.7: Semiotically-heavy Lego sets from Lego.com (`https://www.lego.com`). These sets carry semiotic meaning from books, movies, videogames, and cultural knowledge.

face, magic wands, a golden snitch. However, they also have lots of "chewing gum": 1x2 studs, 2x6 plates, jumper plates, flat tiles, to let the builders create their own structures. The power of the semiotic pieces is that whatever you construct with the chewing gum *inherits* the flavor of the semiotic pieces. A Lego monorail with Harry Potter pieces becomes a Harry Potter monorail. A pirate ship with Batman pieces is a Batman pirate ship.

This principle is clear in the *Lego Movie*[245] when the characters: the townie, the astronaut, Batman, and the rainbow kitty-unicorn all build a submarine together. The resulting submarine has Batman specialty wings on the tailfin, com-

ponents from the 1980s space kits, and lots of pink pieces (pink being a rare semiotically-heavy color in Lego, as it is only included in "girl" kits). The chewing gum has created a clearly recognizable form, while the semiotically-heavy components have made it readable as a high femme/Batman/space submarine.



Figure 24.8: The submarine from the movie, reconstructed by fans (`www.thebrickfan.com/the-lego-movie-master-builders-submarine/`).

To see this principle play out in other kids toys, we can look at the balance

struck in *K'Nex*[s115] construction toys, *Lego*, and *Playmobil*[s175] websites.

- *K'Nex* is entirely chewing-gum; these kits are solely about constructing new structures. They have no branded/tie-in kits on their website, and the signs are more abstract and focused on structural properties ("car", "roller-coaster"), though "Lincoln Logs" is a now-nearly-meaningless sign that was invented in 1916 to evoke hard-working Americana[371][5]

- *Lego*: as previously discussed, there are branded kits (Star Wars, Dodge, NEXO Knights) and a non-branded high-semiotics set ("Corner Garage"). All of these sets but the NEXO Knights have "chewing gum" components for potential structural rebuilds

- *Playmobil* has lots of different semiotically-heavy kits: police, horses, dinosaurs, shopping malls, and a few brand tie-ins. Any of these kits *could* be used together, but have no chewing-gum pieces to create new structures.

24.3 Juxtaposition and the Imagination Playground

Remember that for a casual creator, we want to create big effects from small inexpert actions. One effective proxy for an inexperienced user is a bot: if a bot, acting randomly can produce interesting output, then the activity may be a good potential casual creator for a human user as well. The human only outranks the bot on their ability to evaluate the artifact[6] One apparently strong pattern for creating *lots* of semantic interest from small changes is with **juxtaposition**.

[5] `https://www.knex.com/`

[6] In the chapter on Entertaining Evaluations we've seen that that can be a pleasant experience, too.

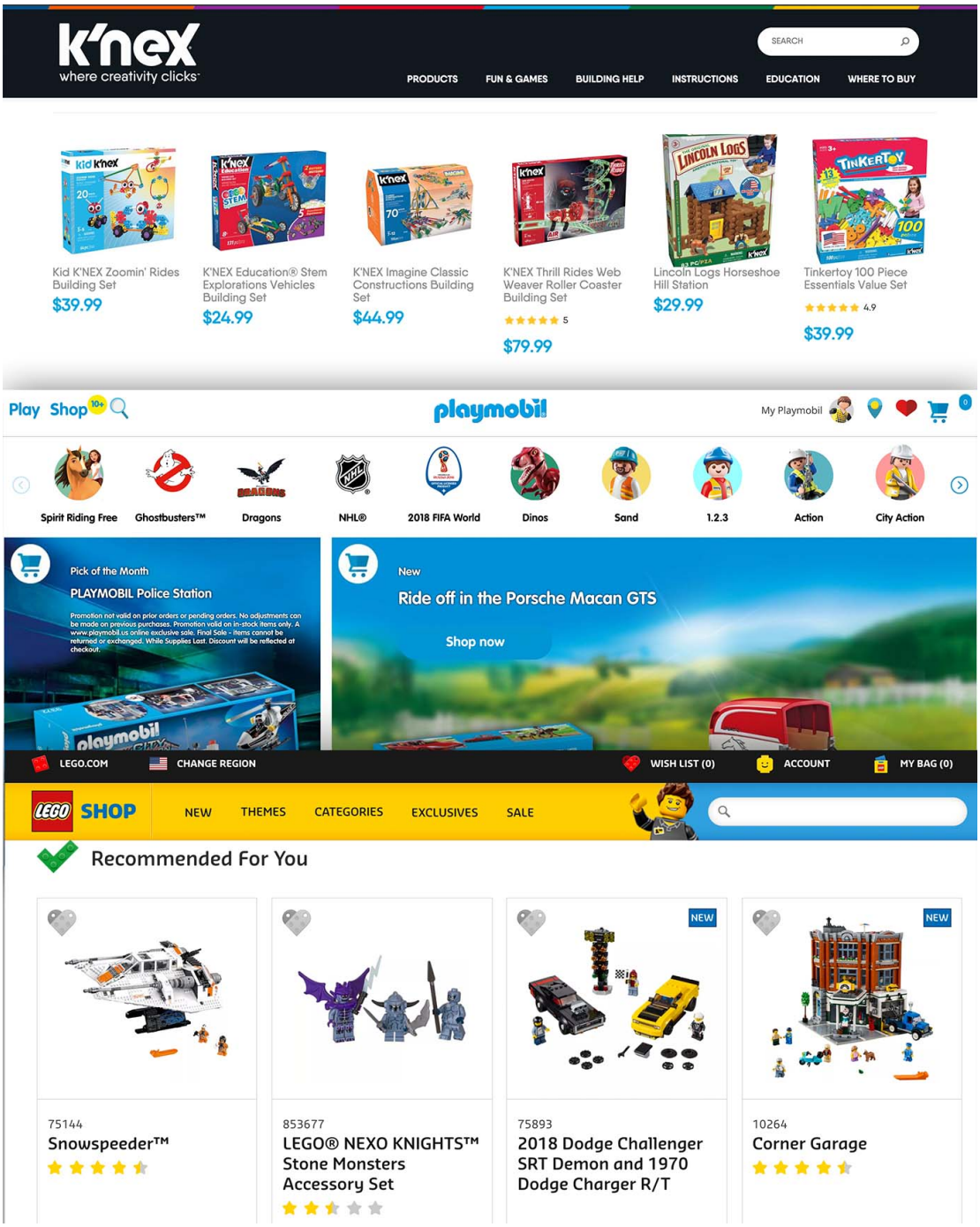


Figure 24.9: The *Lego*, *K'Nex*, and *Playmobil* product pages

I discovered this early on with my Tracery bots and generators. If I used a few forms of structure that juxtaposed different semantically-significant components in

different ways, then I got an interesting range of meaning. Structures were things like "snowclones"[287], phrases that carry a lot of connective meaning between one or more empty sockets, i.e "X is the new black", or "I _SOME_SYMBOL_ NY".

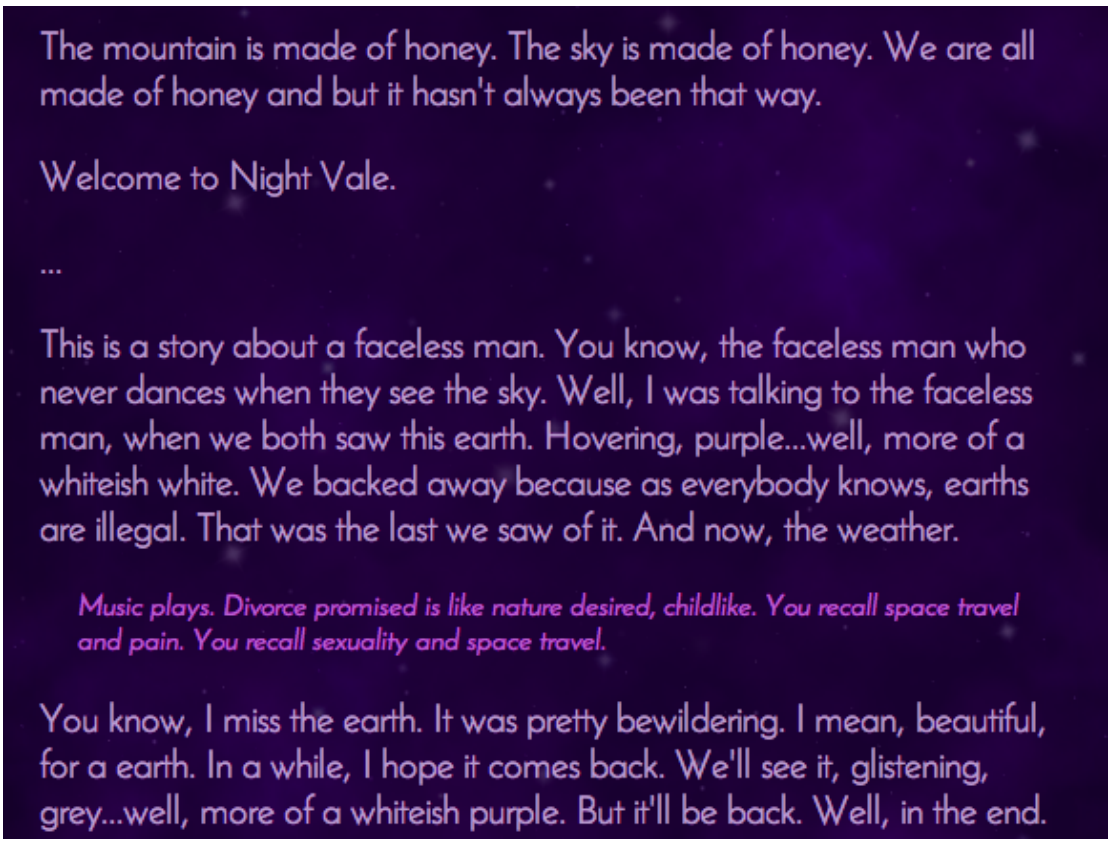


Figure 24.10: A full output of (`https://galaxykate.itch.io/eternal-night-vale`).

I eventually developed the term "imagination playground" for why this seemed to work for bots. The connective structure declares a specific relationship between the semiotic components. The job of the reader is to determine *how* that relationship can be possible. The *cognitive distance* that must be overcome to weave together that rationale is *crucial to the enjoyment of the generated text.* If the relationship is too obvious (**small distance**), filling in the gap isn't interesting. If it



Figure 24.11: A selection of possible first lines of the Nightvale generator. Given a large number, it is easier to see the structures and the semiotic components.

is too distant, then it seems impossible. But if it is tantalizingly plausible, it seems to be engagingly fun to attempt. Thus the term "imagination playground" fits: intentionally tricky-yet-possible jumps to be recreationally crossed by a playful user.

So far the best example of this from my own work is the bafflingbot, a bot that generates "locked-room" style mystery puzzles. Though they don't *have* a solution (its a context-free Tracery bot), people seem to enjoy making guesses as to what happened. I intentionally built in semiotic components that are open to interpretation or "unpackability" in a murder-mystery context (melting snow, winning lottery tickets, incongruous clothing or paraphenelia).

This principle holds with a range of semiotically-heavy twitterbots and text generators that encourage this form of "imagination playground" play.



Figure 24.12: People occasionally try to solve bafflingbot's murders.

These bots aren't casual creators, but the jump from bot pattern to casual creator is a short one. There *are* some casual creator experiences that use automated juxtaposition but many of them so far only use *emoji*.

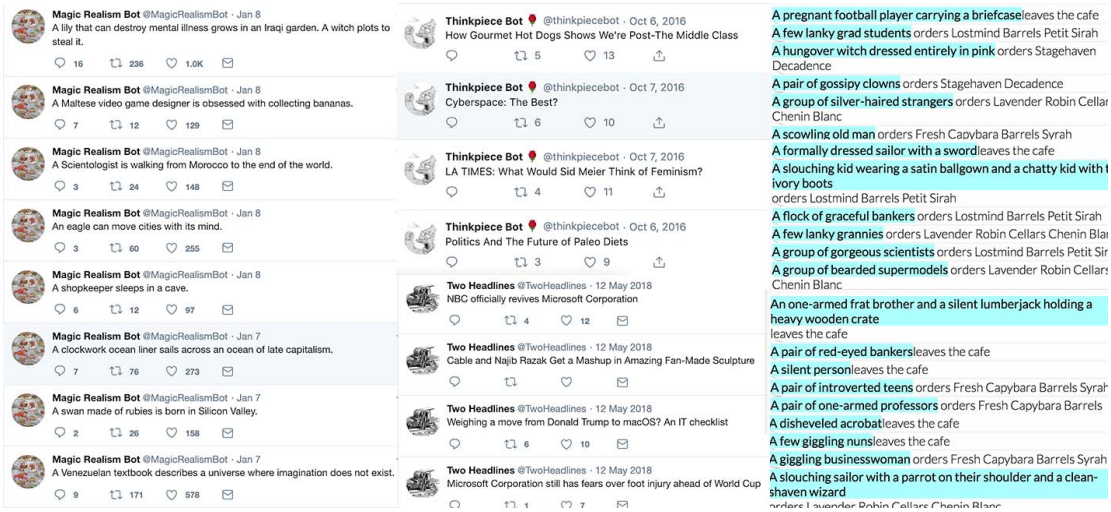


Figure 24.13: magicRealismBot, thinkpiecebot, twoheadlines, and the customer generator from *HipChef*, my unreleased hipster cafe sim.

24.4 Pure refined semiotics: Emoji

Emoji are wildly popular, going from their invention in 1999 to Oxford Dictionary's 2015 "Word of the Year" [113], and appearing in cellphone messages, chat apps, social media posts, domain names[7] and a movie.

Danesi[113], a semiotician, hails emoji as "a new kind of artificial universally usable writing code", stating that "[t]he goal of a universal emoji code has always been to help people avoid ambiguity, which is potentially destructive of human relations". But one of the advantage of an emoji is that it is simultaneously very precise and very ambiguous. An emoji has one denotative referent (defined by its official unicode description "Orange Heart", "Woman With Headscarf", "Person in Steamy Room", "Bowl With Spoon"). But an emoji has many denotative meanings in different communities. Different colored hearts are symbols that can declare Irish pride, Duke pride, kink enthusiasm, autism support, friendship, hate, jealousy, or love[385], but probably dozens of more unique culturally-defined

[7] "poopEmoji" + ".la" is a valid existing domain name



Figure 24.14: Semiotically-heavy terms, but only for game studies programs. Placed in whiteboard-style juxtaposition structures to create an imagination playground of research topics, for the game *Revisions* (`http://www.galaxykate.com/revisions/`).

meanings in any given group of teenagers.

The meaning of emoji can also be influenced by the emoji or words around them

[22] and where they are placed in juxtaposition structures ("I[Emoji:screw]NY", "[Emoji:graduationhat][Emoji:upsidedownface]"). This makes them into veritable machines for producing imagination playgrounds. Several twitter compositional structures have become memes, like "Bunny with a Sign" [8] and the Sheriff[9], which allows them to quickly be transformed into bots. The Sheriff became such

[8] `https://twitter.com/bunnywithsign?lang=en`
[9] `https://twitter.com/EverySheriff`

a popular expressive tool that there is a casual creator Sheriff generator[s20110] to create new Sheriffs. Not every structure is shared and viral, some are created for a single bot, which then seems to "own" (or exhaust) that expressive space, like TinyProtests[11], emojifart[12], tiny_gardens[13], choochoobot[14], bot_ideas[15], and thetinygallery[16].

Writing prompt generators (and a large number of school writing prompts) exist for creating emoji sequence prompts for stories. *Emoji Prompts*[s6517] is interactive, presenting the user with a "then what happened" button, to turn a random generator into a casual creator; the *user* is in charge of reading the story and deciding whether to continue it or stop it there.

If the Sheriff generator is a generator for Sheriff memes, Tracery/Cheapbots-donequick have provided a casual creator for creating wholly new emoji templates.

> "Tiny Gallery may be cute and fun and a little throwaway, but it does point playfully to the way the web offers a home to a growing proliferation of alternative art spaces which, as Winston suggests, "seem to be open to people who are comparatively excluded from conventional means of cultural production."[24]

Emoji provide a common language, if not of meaning, than of "meaning to mean". We expect emoji to signify a kind of common significance, to the point that we can create new "emoji", and imagine how they could fit into common language use. This is done textually by the bot better_emoji: e.g. "A skeleton in a dragon costume", "Superman baking a cake". In "How Shell and Horn make a Unicorn: Experimenting with Visual Blending in Emoji"[112], Cunha, Martins,

[10] `https://howdy.petetnt.xyz/`
[11] `https://twitter.com/TinyProtests`
[12] `https://twitter.com/emojifart`
[13] `https://twitter.com/tiny_gardens`
[14] `https://twitter.com/choochoobot`
[15] `https://twitter.com/bot_ideas`
[16] `https://twitter.com/thetinygallery`
[17] `http://byrdseed.com/emoji/`

and Machado create a bot to visually generate new emoji based on semantic mashups of other terms. Though they have an instagram account to post new ones to[18] it seems like this would be a fun interactive toy that could become a casual creator.

Explorable explanations maker Nicky Case created a cellular automata creator called *Simulating the World in Emoji*[s20719] that makes use of emoji. This simulator lets users create automata rules (and thus a sketch of a simulation), as well as picking a symbolic emoji for each type of cell. The emoji are ideal here: they are readable, HTML compatible, and emphasize that each cell is an archetype and a symbol, and not a real entity. They also suggest that its not a scientific simulation, but something playful: "If any of this sounds intimidatingly intellectual, don't worry: it's all rendered in highly relatable emoji"[168] the Killscreen review states, soothingly.

Emoji are just one of many semiotically-heavy tools used to create "meaningfulness" in casual creators. Brands, tropes, grouped imagery (Christmas symbols, "hipster" signifiers), and even certain color sets (Mardi Gras colors, red-white-and-blue) can all be used as part of generative pipelines. These signifiers can create meaning while standing alone, but they also provided a powerful juxtapositional meaning-generating engine when a generative system recombines or repurposes them.

[18] `https://www.instagram.com/emojinatingthings/`
[19] `http://ncase.me/simulating/`

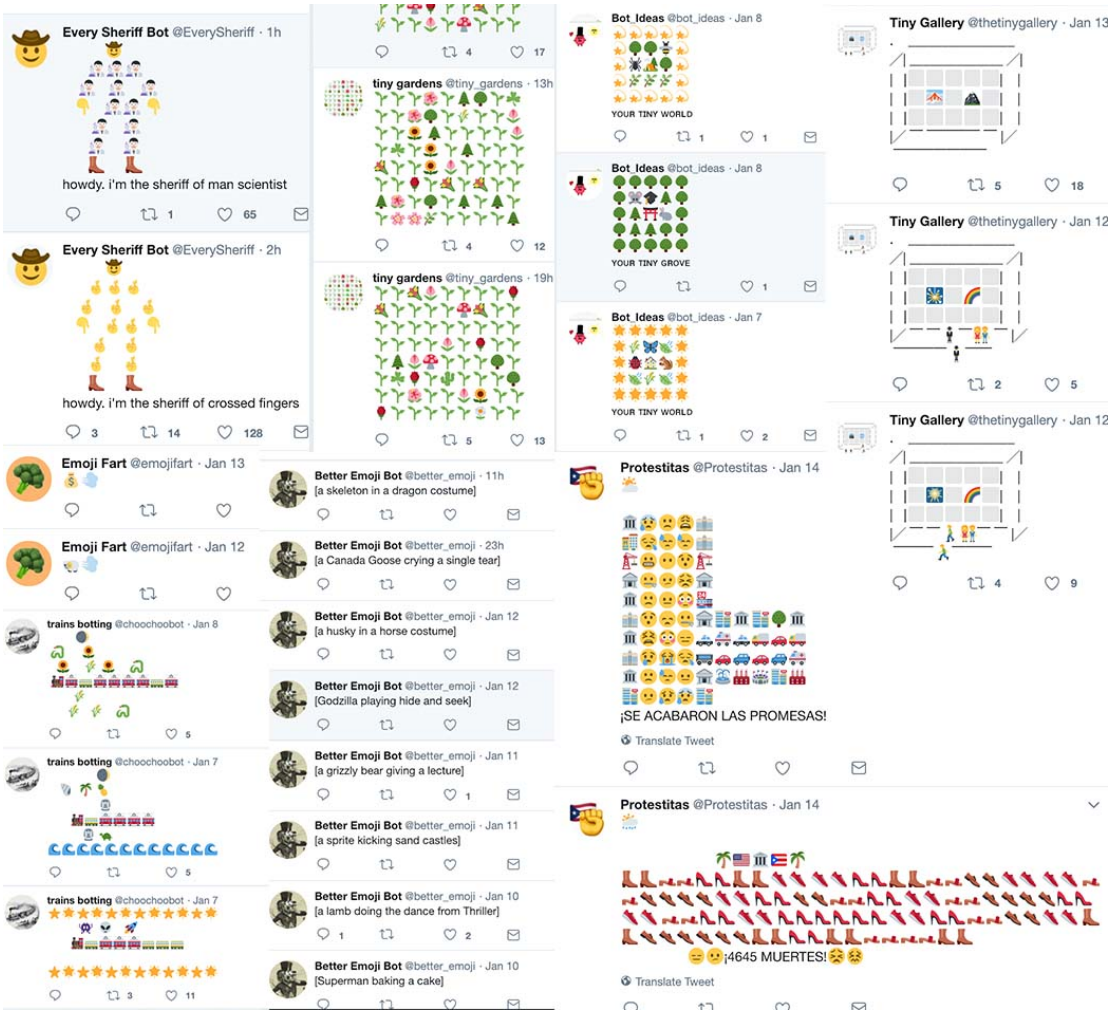


Figure 24.15: Emoji bots everysheriff, tinyprotests, emojifart, tiny_gardens, choochoobot, bot_ideas, and thetinygallery.

Chapter 25

Pattern - Expressing Identity

Perhaps the most commonly encountered casual creator (for gamers, at least) are the ubiquitous character creators that come with nearly every multiplayer game. These character creators exist in a spectrum. Some games have a deep multi-layered character creators where one can tweak cheekbones, jawlines, and eyebrow density (*Blade and Soul*[s16],*Monster Hunter*[s137]). Some, like *World of Warcraft*[s244], let the player select from a fixed number of options, for each of a number of parameters like hair style or skin color. Other games limit the player to a fixed roster of stock characters (*Team Fortress 2*[423]), though many of these allow the player to choose from a variety of outfits, or buy hats. The concept of *who we are online* is a topic well out of score for a short chapter (or a dissertation)[414], but we can confidently assume that whenever people interact with other people online, some of them will want to buy hats.[276]

We have seen that most casual creators have a generative pipeline, which creates a possibility space that the user explores and learns to navigate. In identity-creators, the possibility space defines which *identities* are contained in that space, and which ones are not. What identities exist in this space? Do identities with



Figure 25.1: IMVU, a long-running avatar-based chat app and virtual goods platform. Users can buy clothing and hair, or create and sell them (via pro-creative tools like *Photoshop* or *Maya*). All options are select-only: there are no sliders or color-pickers here, as many objects are user-created and non-editable. Everything in IMVU is designed around participating in social situations, like buying the ability to create custom emojis or gestures, or post to a virtual in-game Instagram-like feed[205]

black skin, wheelchairs, or large bodies exist in this space?[1]

Identity editors are often some of the least *technically* complex generative pipelines in casual creators. We can see an example of a few of these in Figure 25.2. Most often, their generative space has some number of customization slots, with a content-selection menu of several options for each slot (*Create Your Own Unicorn*). Sometimes they also have sliders, or color pickers[2]. Rarely, they may include placement/distribution of items(*DIY Fashion Star*). As we have seen in the chapters on possibility space shapes, this give them a few options for ways that users can move through the possibility space, but systems in this genre *very*

[1]The Microsoft XBox avatar-editor is a rare example that includes wheelchairs and prosthetic limbs in their identity space[url:https://twitter.com/XboxQwik/status/750191281196838912]

[2]The most complex color picker are the 9-dimensional palette selectors for *each object* in *Monster Girl Maker!*

rarely use anything but direct controls like selection menus and sliders.

One exception to this is the automated "creation" of an avatar given a photo, which has been implemented by both Facebook[439] and Google[115]. In Google's editor (part of *Allo*), users snap a picture of themselves. The image is then processed by a classifier (a neural network trained to pair photo data and content-selection choices for each slot) and used to select the "right" content choice. Fortunately, the team recognized the need to override the classifier's guess, and also included a traditional editor that activates after the computer makes its guess.

> "Rather than aim to replicate a person's appearance exactly, pursuing a lower resolution model, like emojis and stickers, allows the team to explore expressive representation by returning an image that is less about reproducing reality and more about breaking the rules of representation. [...] Reconciling how the computer perceives you with how you perceive yourself and what you want to project is truly an artistic exercise. This makes a customization feature that includes different hairstyles, skin tones, and nose shapes, essential."

Once the user is satisfied, the content-selections and resulting avatar model are used to create a bounty of new digital stickers starring the user (or their avatar version of themselves) which can be used in social chat. This nicely mixed-initiative flow gives the user override and control over the algorithm's choices, but let them take advantage of its generative power to create the starting avatar (no blank slate!) and generative stickers.

Google has also used its deep-learning networks for other self-expression or self-identity experiments. The *Google Arts and Culture* app contains a mode where users snap a selfie, and the app measures (via deep learning), the distance between that picture and a number of other artworks in Google's collection, to find a doppelgänger of that user hanging on a museum wall.[249] That information retrieval was the end of the pipeline (its not much of a creativity app), but it was a powerful reminder of people's interest reflected back at them through an

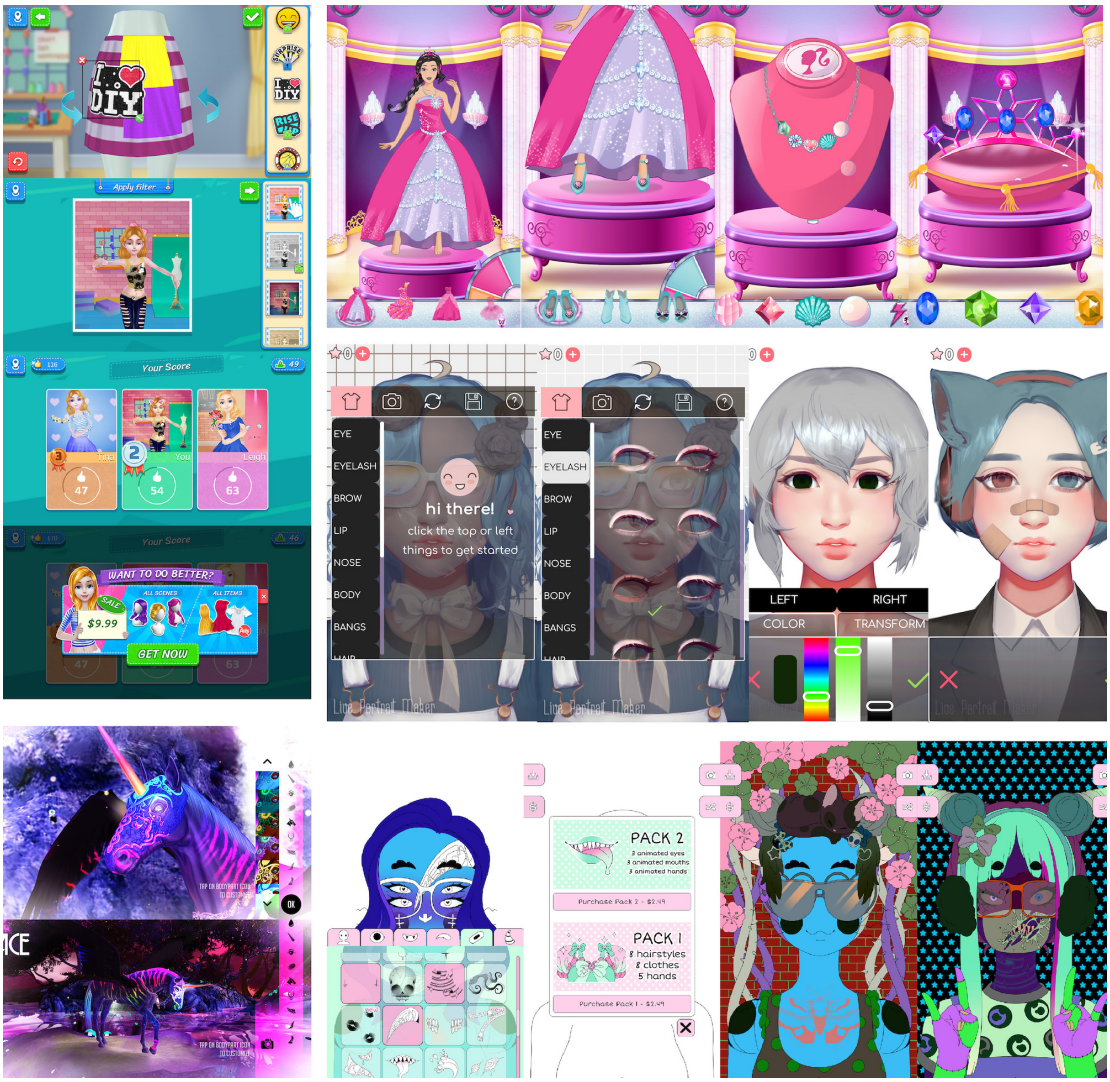


Figure 25.2: A range of dress-up and avatar apps on the Android play store. (top-left)*DIY Fashion Star* is a fairly full-featured casual creator for outfits (notice my cats' picture on the skirt). The avatar then poses and is rated on a virtual Instagram clone (see chapter 16). (top-right)*Barbie Magical Fashion* is a typical narrow-possibility content-selection creator, but includes two socketable-gem jewelry creators. (bottom-right)*Live Portrait Maker* and *Monster Girl Maker!* both have large content-selection spaces, with complex multidimensional color-pickers, and in *Live Portrait Maker*, the ability to break left/right symmetry. (bottom-left)The app stores contain a spectrum of human avatar-makers, furry avatar-makers, and animal creators, with no clear distinction. *Create Your Own Unicorn* follows many of the patterns of avatar makers, but for an animated 3D unicorn instead.

algorithmic funhouse mirror. And it increased users' level of engagement with new art that they found:

> "After a short moment, you will see your results along with a percentage to estimate the visual similarity of each match and your face. Your selfie becomes a doorway into art—tap on your lookalike to discover more information about it or an artist that perhaps you've never heard of before."

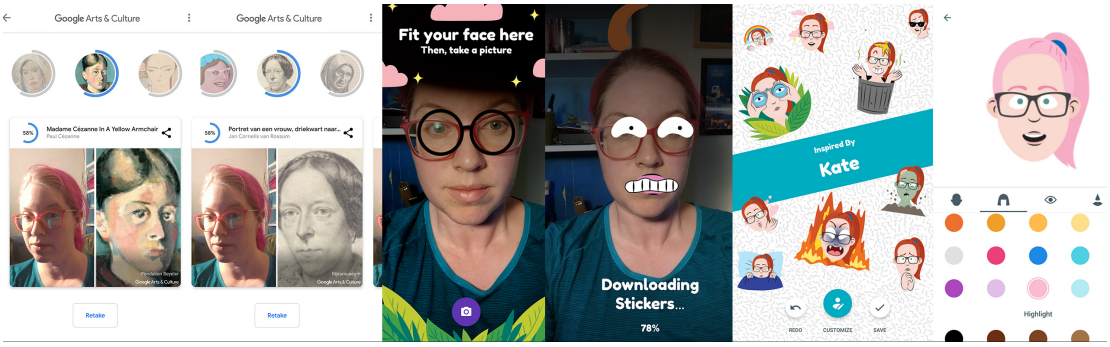


Figure 25.3: The avatar-sticker-maker in Google *Allo*.

It is also not uncommon to see avatar creators have a "Randomize" button. The *Stardew Valley* character creator has this, as do *Live Portrait Maker* and *Monster Girl Maker!*. *Soul Caliber*'s costume editor has the most advanced version of this. As a fairly complex system with a high-dimensional possibility space, the designers may have been concerned about the user's ability to randomly jump to a "good" artifact in the space, and only end up seeing mixed-up jumbles of outfits. Instead, users can limit properties of the editor, such as the outfit's era and location, to only generate medieval European-style outfits, for example.

As we saw in chapter 24, signifiers are a major part of identity, and of identity construction in a social space. Systems like *IMVU* and the *Sims* have always had a lively marketplace of copyright-infringing goods (even when they attempt to suppress them). The closer a virtual space is to the real world, the more people will want to express their *current* identity, which in our corporate-controlled,

marketing-flooded world is often through brand affiliation. Branded clothing, especially, is an important part of these apps, where users want to live out a fantasy-life that expresses their values of brand-loyalty or fantasies of wealth.

In systems that are about creating characters in worlds that *aren't* the real world, there are signifiers of importance for *that* particular world that must be included for characters in that world to be "complete". Every Harry Potter character must be sorted into a house, and receive a wand that represents their personality, so many apps and systems have sprung up to let users engage in those fantasy moments[3]. Choices like these are **moments of self-expression through the lens of a fictional world's values and objects**[4]. Letting users "Simpsonize" themselves, as the *Simpsons* movie advergame did[5] is to invite them to see themselves in that fictional world, and consider what their place would be in it. This is an interesting cognitive space for a user to explore, it must be matched by the possibility space in the branded character-creator. If the character creator prohibits people like them (their hair, their body, their skin, their glasses) from the possibility space, they will feel excluded from the world.

This chapter has focused on how users express their identity and sense of self through avatar-, fashion- or character-creators. But many users often playfully create representations that are *not* themselves. A popular Youtube series, *Monster Factor*[6] attempts to discover the *worst* possible creations in any given editor's possibility space. In Figure 25.6, we can see examples of users (including me) enjoying virtuosically creating creations that are unexpected or outside of the thematic range of the creator, like Ronald McDonald in a fighting game, or cre-

[3] `https://www.pottermore.com/news/discover-your-wand-on-pottermore`

[4] Are you TeamJacob or TeamEdward, dear reader?

[5] `https://disneyandmore.blogspot.com/2008/09/are-you-ready-to-simpsonize-yourself.html`

[6] `https://www.youtube.com/playlist?list=PLaDrN74SfdT6duuVl_8qxJ5eaaPHRX_ij`

ating a Sarah Palin spaceship that one can use to fly around the galaxy, shooting lasers.

Personality and self-expression are strong motivators for creative activity. As we saw in chapter 24 and this chapter, people will express personality with brands, tropes, and signifiers, and also through the tropes of particular worlds. We have seen that casual creators provide generative spaces for users to explore, and this is also true for creators that create a possibility space to express *identity*. If we consider that casual creators are designed to help a user navigate quickly and safely through a possibility space, it should not be unexpected that casual creators, like avatar creators, would be used by players to experiment with gender and social presentation[348][350].

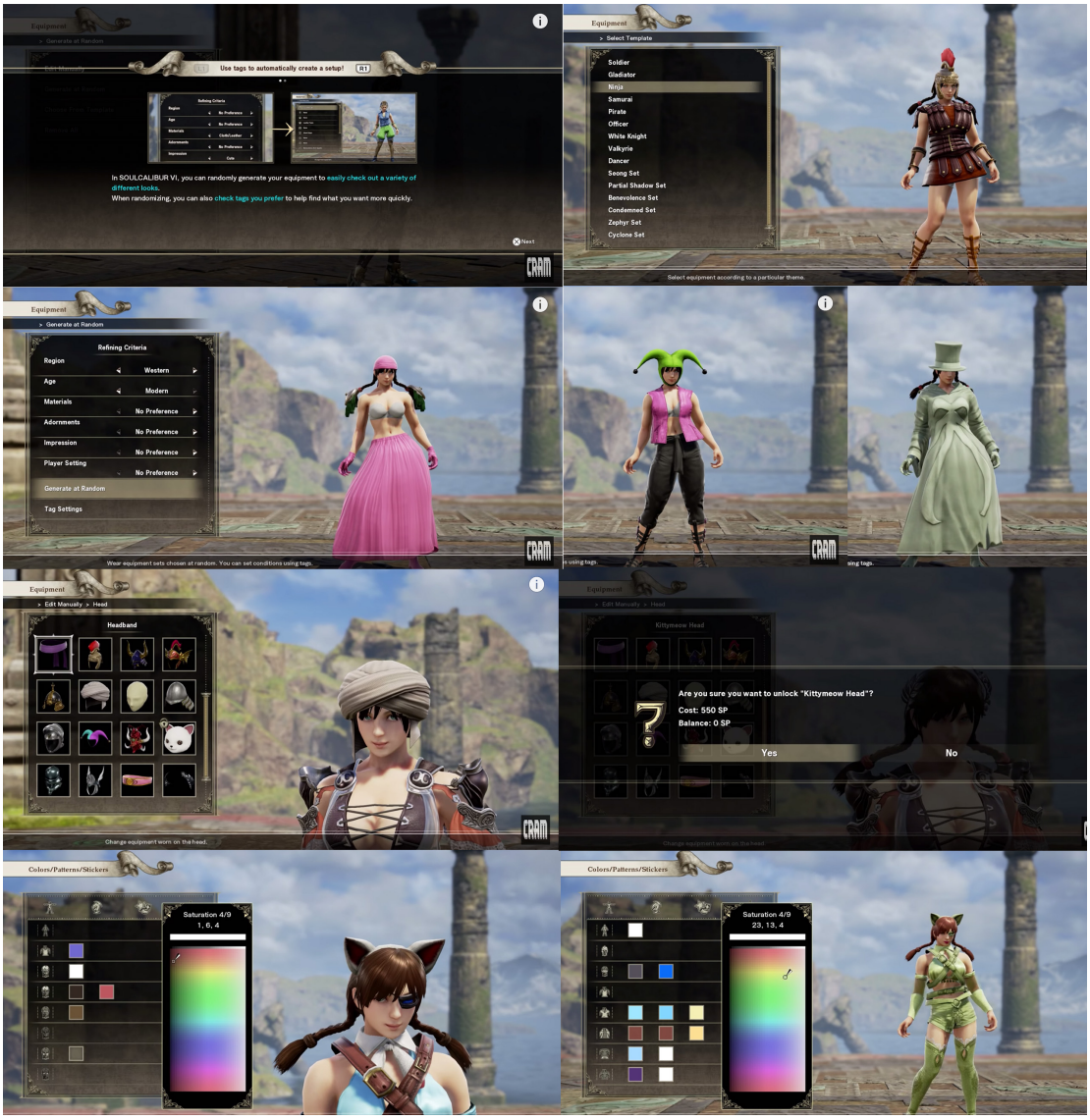


Figure 25.4: *The Soul Caliber* editor has the most interesting casual creator features. While other creators have more options, this editor is one of few that lets the player browse their options in ways other than manually setting a slider. After selecting a base character, they can use the editor normally, randomly roll a look, or jump to a "landmark" in the space, like pirate or knight (row 1). Randomness can jump to any point in the space, or can be constrained by era and region (row 2). As usual in these editors, accessories from a range of semantic domains can be purchased (row 3). Individual items can be colored, or the whole outfit can be color-changed as one (row 4). Accessories can be further spatially manipulated (with a gamepad), and personality, e.g. outbursts, can be changed (row 5) (`https://www.youtube.com/watch?v=ukpE7ERWtSY`).

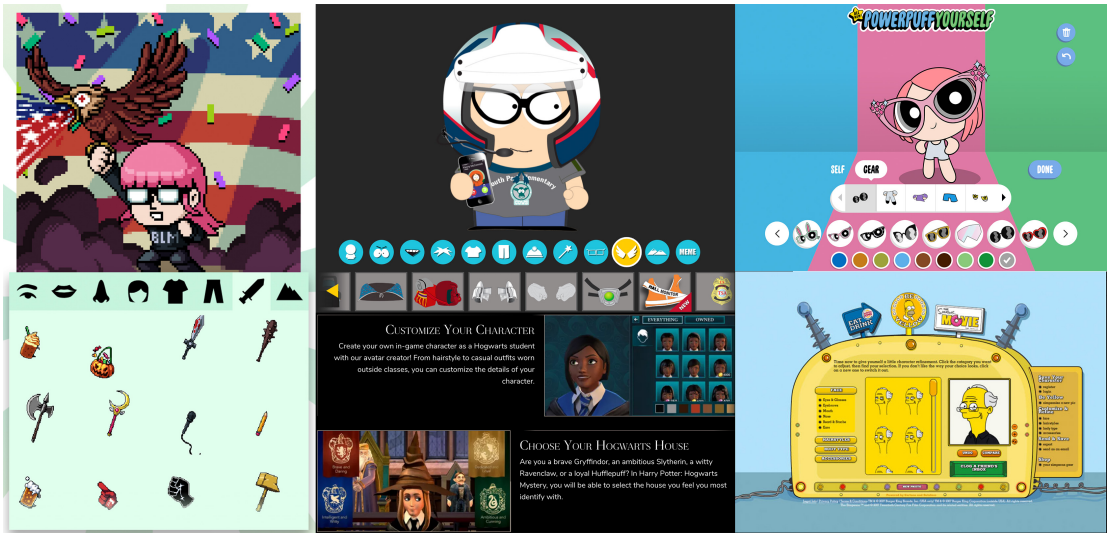


Figure 25.5: Character creators to promote a voter registration campaign (`https://voterhero.com/`), South Park (`http://southpark.cc.com/avatar`), Harry Potter (`http://www.harrypotterhogwartsmystery.com/`), the Power Puff Girls (`https://powerpuffyourself.com/#!/en/my-powerpuff/create`), and the Simpsons (`https://disneyandmore.blogspot.com/2008/09/are-you-ready-to-simpsonize-yourself.html`). In each, the creator includes signifiers that are important to that world, like patriotic symbols or in-world gags, as well as items that are identity signifiers like glasses and hairstyle.



Figure 25.6: Users do not only represent themselves in casual creators. They often attempt to make caricatures of acquaintances or celebrities. (right)Ronald McDonald and Bojack Horseman in *Soul Caliber 6*'s editor[122](left)Barack Obama in the wrestling game *WWE 2K16*[167], and candidates for the 2012 election that I made with the *Spore* spaceship editor for marketing purposes for Spore[391].

Chapter 26

Expert evaluation

Recall (from chapter 1) that my final two research questions were:

- What set of design patterns provide prescriptive guidance for designing new successful casual creators?
- If these design patterns are used to design new casual creators, do they ease and improve the design process?

One of the proposed contributions is to "motivate the identification, study, and creation of casual creators, and catalyze the formation of an interested new community of research, development, and construction of shared knowledge"

In the previous ten chapters of patterns (and six chapters of possibility space categories), we have seen a variety of common threads that apply across casual creators. I have proposed that these pattens, and the language that I am using to describe them ("generative pipelines", "possibility spaces", "grokloops") will be useful for both building better casual creators, and identifying and understanding the many that already exist. How can I **evaluate** whether these are the correct patterns, or a useful set of terms?

One way to evaluate this, informally, is to design systems with these frameworks and patterns in mind, and see if that eases the design process. I have designed several dozen casual creators (of varying levels of completion) over the course of this dissertation, and many of them have been designed using these tools. *Idle Hands*[s97] is an interactive art installation where LeapMotion hand-tracking data is translated into a colorful Voronoi diagram projected on a screen, and was designed with the *Generominos*[s82] cards. Using the cards made it clear what data was being passed along the pipeline and that it needed no difficult "translation" steps, and the whole experience was able to be constructed in an afternoon.[1] In the design of *Binary Fission*[s13][83], a DARPA-funded game, we needed users to construct a series of algorithmic filters to divide valid loop data from invalid loop data. The previous approach (in the game *Xylem*) had been to have the player construct equations, but that action took a while for the player to input, and was slow for them to evaluate. I looked at the problem from a casual creator perspective, and realized that we needed faster forms for those actions. In *Binary Fission*, the player creates decision trees of filters, so I created an "Iris view" of dozens of generated filters, which the player could browse, see evaluated (and animated) and click to select. Quite large trees could be browsed and generated within seconds. We ended up with a system where players created plentiful (and high-quality) solutions without frustration, thanks to the newly-optimized grokloop.

These are two anecdotal applications of "casual creators" frameworks as an intervention in the design process. But they were both accomplished by me, the writer of this dissertation, and so could be the result of my own thought-processes about design (which *led to the framework*), not clearly the result of the framework itself. I wish to argue that this framework will be adopted and used by *other*

[1] Much "yak-shaving" in interaction design happens when two pipelines of data require "just a little work" to connect them, and months of difficulty later, the pipeline mangles its data or still doesn't connect.



Figure 26.1: *Binary Fission*'s "Iris view" for changing the long process of manually inputing and evaluating an equation into a process of browsing and selecting generated equations nearly instantaneously.

practitioners, not just myself. It has been picked up as term by several academics [63][284][295][241], but it is difficult to determine if it is seeing any adoption in the games or software design industry[2].

To encourage the adoption of the "casual creator" term and framework across a broader audience. wrote an abbreviated 20-page version of this dissertation in illustrated "zine" format. A "zine" is an informal booklet, meant to be printed cheaply, given away (or sold, cheaply), and read quickly. Zines have been a way for under-recognized authors to communicate ideas outside of formal established channels[221], and they are gaining popularity in computer science (especially in games and expressive AI) as a way to share manifestos, political beliefs, education, and outreach both at academic/industry conferences and at social meet-ups[3]. I

[2]One milestone to watch for will be the first GDC talk with "casual creators" in the title.

[3]Several academic labs now have zine walls, and we trade zines with each other to add to our walls. Posting a photo of someone's zine on your lab's wall is a social way to express admiration

open-sourced the zine (to get a headstart on building "casual creators" as an idea), and asked a number of respected interactive-generative software creators to review the zine in a survey[4]. Seven of them filled out the survey[5]

The respondents were M Eifler (artist), Vi Hart ("recreationional mathemusician"), Adam Smith (professor in Computational Media at UCSC), Mike Cook (designer and researcher), Nicky Case (independent, designer of "explorable explanations"), Simon Colton (Professor of Computational Creativity, AI and Games at Queen Mary University of London and Monash University), and Nicole He (creative technologist). They were asked to answer four Likert scale or yes/no questions:

- "Is this zine applicable to your work?" (1-5)
- "Would you recommend this zine to a colleague or peer in your field?" (1-5)
- "Would you recommend this zine for a new practitioner in your field (junior colleague or student in a class)" (1-5)
- "Have you heard the term 'casual creator' before?" (yes/no/maybe)

They were also asked several open-ended questions:

- "Having read the zine, what (if anything) do you think is missing?"
- "What (if anything) should be removed?"
- "What (if anything) is unclear?"
- "Which, if any, concepts from this zine do you anticipate would be useful in your own work?",

and note the utility of their research.

[4]Accepted as "exempt" research by UCSC's IRB.

[5]For which I am very grateful. "Respected interactive-generative software creators" tend to be very busy people.

- "Would you use the term "casual creator" to describe any of your own work? (why or why not?)"
- "Which works in your field would you consider "casual creators" (if any)?"
- "Has this zine made you aware of other "casual creator" works that are relevant to your work? Which ones?"
- "Which design patterns are missing? Can you think of any patterns that you have seen that are not represented here?"
- "Which three (or less than three patterns) do consider most important or common in your work?"
- "Do you have any other feedback for this zine? (I'm hoping to make it a useful tool for our development community and students)"

The responses were generally very positive, and noted the usefulness of the concept of "casual creators".[6]

"Having read the zine, what (if anything) do you think is missing?" Respondents recommended adding information about some additional fields ("Intelligence Augmentation", Computational Creativity, Self-Determination Theory, improvisation). Two respondents requested more clarification on the distance between "serious" creators and casual creators, and works that exist in the middle (e.g. *Gamemaker*[s77]). They also noted that the example systems are listed but not described, or requested more information about particular systems that readers might not be familiar with

What (if anything) should be removed? What (if anything) is unclear? One respondent "found the distinction between 'geniuses' and 'sort-of-

[6]This may be somewhat biased, as the people who were more likely to respond to the request were those who knew me personally.

creative people' initially startling", while another noted that P/H creativity wasn't listed in the zine (another respondent noted in feedback that I also called it "personal/historical" without noting the acronym).

It was observed by several (over multiple questions) that the zine defines a lot of terms, and that those terms are hard to find or cross-reference, so a more polished glossary (and consistent definition style/location) would be helpful. Some also noted bad formatting and expressed confusion at some of the bullet point structuring.

Which, if any, concepts from this zine do you anticipate would be useful in your own work? Three of the seven responses called out "grokloop" as a useful term. Also noted once each were the hedonic curve, false evaluations, surfing/steering/control, power-vs-control, surprise/emergence, and the unfolding/scaffolded/little properties (from chapter 14) as applicable for more than programming languages.

Would you use the term "casual creator" to describe any of your own work? Responses were mixed for this, with negative responses cited for not having built one yet, considering their work more non-interactive (or non-casual) than a casual creator. Others reflected on parts of their work that is, in hindsight, casual creator-y, especially Simon Colton, who has retroactively adopted the "casual creator" for much of his past work.

Which works in your field would you consider "casual creators" (if any)? Has this zine made you aware of other "casual creator" works that are relevant to your work? Which ones? *Anyland* (a VR tool), kaleidoscopes[s110], *Sentient Sketchbook*[s199], *GanBreeder*[s80], *Fractal Doodler*, *Wevva*[s241], *Learning Music by Ableton*. Interestingly, two noted examples of possibility-space annotation: the "numbers with names project"[7] and user-created mandelbrot

[7] `https://www.doc.ic.ac.uk/crg/software/nwn/project.html`

videos. Users noted a couple of casual creators that they had learned about through the zine[8]. One reflected on game-creation apps at work that were trying to be casual creators but "kind of suck", which suggests that this zine could be a useful intervention in that environment.

Which design patterns are missing? Can you think of any patterns that you have seen that are not represented here? Respondents suggested "Enables skill ramping", "finished result" vs being part of a larger pipeline, rendering temporal experiences to spatial forms for faster evaluations (as in chapter 13), displaying artifacts in an informative context to aid evaluation, "Combinatoric Explosion"/"massive possibility spaces"[9], "Low floor, High ceiling, Wide walls" (from *Scratch*[s195]), evolutionary algorithms as entertaining processes to watch, and aesthetic polish ("looking/sounding fantastic").

Which three (or less than three patterns) do consider most important or common in your work?

- *Default to Always Working:* 4 votes
- *Joys of Generativity:* 4 votes
- *Instant Feedback:* 4 votes
- *Entertaining Evaluations:* 2 votes
- *Repetition Creates Interest:* 2 votes
- *Gestural Control:* 2 votes
- *Possibility-space Annotations:* 1 vote

[8]Several noted that they knew about most of them through following me on Twitter, so this is promising for outreach for those who *don't* know me personally.

[9]I would argue that this is a callback to the "very large number" fallacy and might be an anti-pattern.

- *Enable Self Promotion:* 1 vote
- *Create Channels for Users to Communicate:* 1 vote
- *No Blank Slate*, *Enable Group Learning*,*Adopt and Collage Signifiers*, *Make It Real:* No votes

Do you have any other feedback for this zine? (I'm hoping to make it a useful tool for our development community and students)

Respondents offered encouragement and enthusiasm for the project. They also noted types, an under-developed references section, and requested more illustrations and external information like a site of examples.

This is a small and non-randomly sampled group of practitioners, but they are well-respected and influential within industry, academia, and indie circles, so I consider that these answers are promising for the future of casual creators. The languages of casual creators, at least as used in the zine, seems to resonate and capture a part of the experience of design for the respondents.

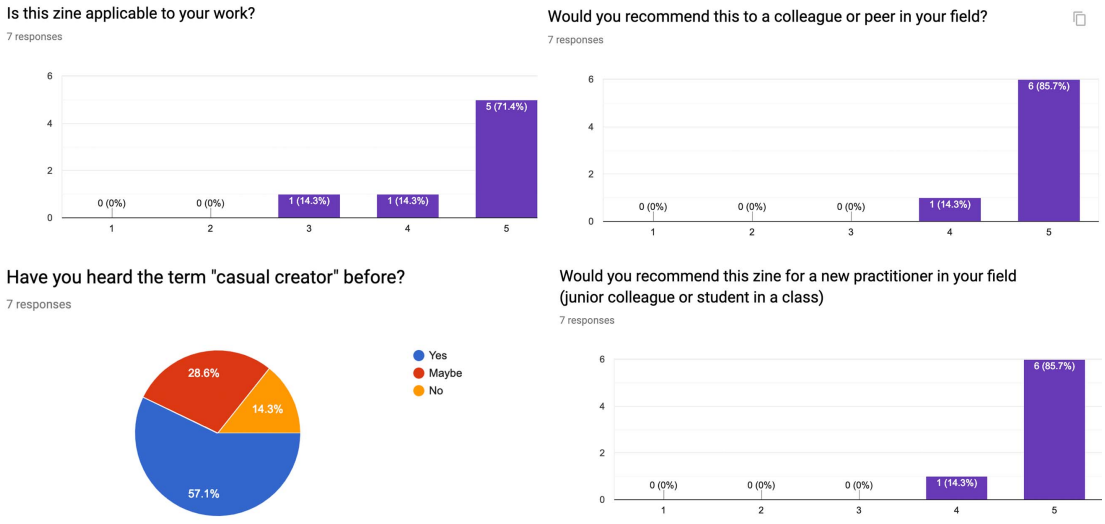


Figure 26.2: Results of the survey. Most respondents would recommend the zine to junior or senior colleagues, and found it relevant to their work, even those who had not heard the term before.

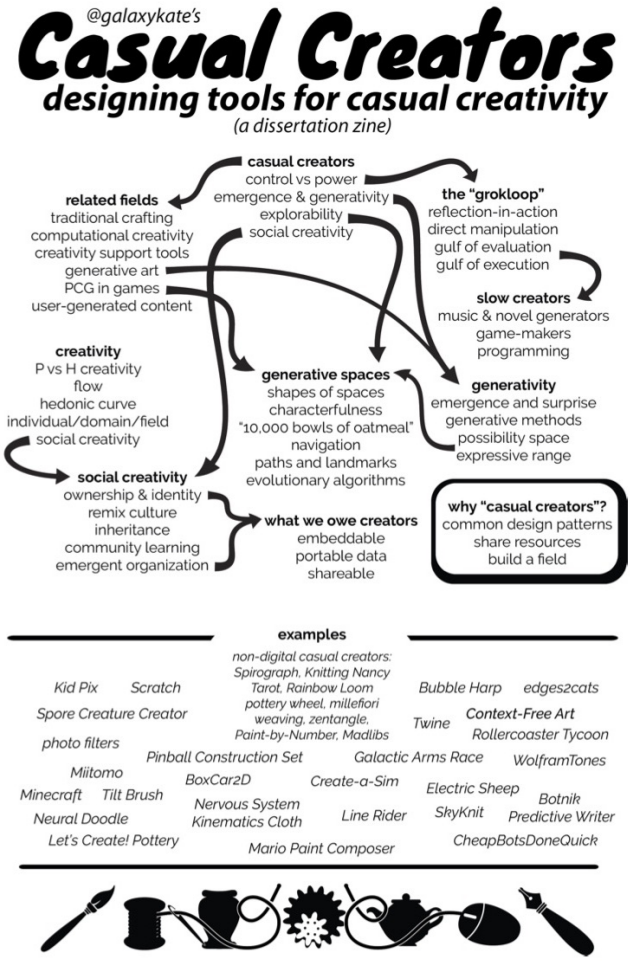


Figure 26.3

Can AI be a Kindergarten Art Teacher?

We have created AI that can trade stocks, AI that can drive cars, AI that can shoot weapons. Could we make AI of an encouraging kindergarten art teacher? **What would an AI Bob Ross look like?** This zine is about creativity for casual users, and how to write software systems (or apps) that *enable casual creativity*. I call apps like that "Casual Creators".

Anything with a built-in system to casual creativity can be a casual creator. This might be a photo editing or music-making app, or it might be embedded in a game, like a character creator. It might even be a physical object, like a pottery wheel! All these system share some properties, though, and what we learn from some can help us better design others.

What is a Casual Creator? A Casual Creator is:

- an interactive system that makes "artifacts"
- provides some mechanical or generative assistance
- generativity creates a rich possibility space
- enables fast, confident, and pleasurable exploration
- encourages a feeling of creativity and creative ownership
- often amplifies that feeling of creativity with social sharing

Usually they are digital, but not always (spirographs, rainbow looms, and pottery wheels are all examples of physical casual creators). These systems are used to make *things*, like art or music or patterns or 3D models. You can design a casual creator to make *any sort of thing* so in this zine I refer to the things the casual creator is making as "artifacts".

Here's my *academic definition*:

> A Casual Creator is an interactive system that encourages the fast, confident, and pleasurable exploration of a possibility space, resulting in the creation or discovery of surprising new artifacts that bring feelings pride, ownership, and creativity to the users that make them.

You have probably used lots of casual creators before. Lots of kids crafting kits make the casual creator trade-off (generative power vs control), but so do a lot of tools that don't expect mastery from their users (such as embedded systems like game-character creators)

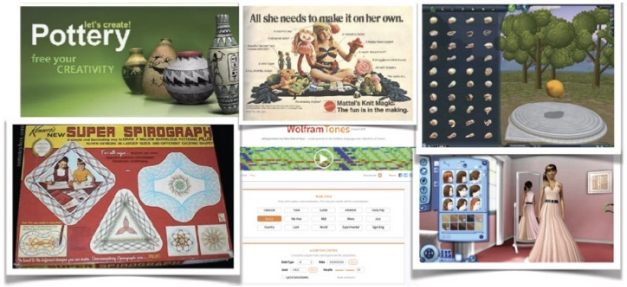
There are many pieces of software designed for **maximum creative control**, to help the potential Mozarts of their field to create the magnum opus of their dreams. This zine isn't about that software, or about those users. **Instead, this zine is about playful, doodling software that uses AI to help sort-of-creative people create sort-of-interesting things in not-very-much-time.**

This zine will help you, and your students and your coworkers, design better Casual Creators. Some casual creatorss are standalone software like apps, but others can be embedded inside other experiences, like a character creator in a game. Even "serious" creativity software will have casual modes or tools (like Photoshop's 'mosaic filter'). All these different creators still have many things in common, and often share best practices, which you will learn about in this zine!

Casual creators have a rich generative possibility space (more on that later) of many interesting artifacts that the system *can* make, but it is still easy to make new artifacts rapidly, to quickly explore this possibility space. Making new things is not only fast, but also fun (using the system feels good) and safe (its very hard to make something bad, or if you do, you can easily change it to something better).

Finally, when people are creative, they want to share that creativity, whether with their friends, their fellow makers, or with the world. Often casual creators will have ways for people to connect to each other, sharing tutorials, finished artifacts, artistic goals, or mutual appreciation. Even very simple forms of creativity can become interesting when they are part of a social interaction.

"making" vs "discovering" The possibility space of a casual creator is defined by its generative pipeline (i.e: you can't make something that can't be made by that algorithm). Does this mean that users are only "discovering" artifacts that already exist mathematically? You can shift their perception by the framing language you use in your casual creator: the user can *feel* like they were creative one for finding/making their new artifact.



Some casual creators. Some are physical, some digital. Some are experiments, some are apps, others are embedded in a commercial game. Have you played with any of these? Let's Create Pottery, the Knitting Nancy, the Spore Creature Creator, Spirograph, Wolfram Tones, and The Sims Create-a-Sim editor

Casual creators are a *genre*!

A genre can be defined as **"a set of mutually-supportive design patterns"** (Jonathan Lessard). So Casual Creators are a *genre* of software with a set of *design patterns* that all reinforce users' feelings of creativity, confidence, surprise, and pride. This zine will show you the design patterns shared by many casual creators, like trading control for power, surprise and emergence, explorability, ownership, but also many smaller and more concrete patterns. These are some of the overarching patterns that we need to keep in mind in the rest of this zine:

- **trading control for power**: users trade full control (you can't make *anything*) to gain automated assistance (you can make things *quickly* and *easily*). The automation can also guarantee **safety**, knowledge that it's impossible to build *bad* or *broken* artifacts (so you can be playful).

- **surprise and emergence**: users' input is expanded and elaborated on by a pipeline of generative methods. Something bigger and stranger and surprising *emerges* from the user's simple gesture.
- **explorability**: the generative pipeline has a **possibility space** of things it can create. It's a sufficiently large and high-quality space (containing good artifacts and not too many bad ones) that its worth exploring. The system provides enough **assistance and guidance** for exploring it that the user doesn't feel lost or stuck.
- **ownership**: users feel *pride* in what they've made. This is even more powerful if users didn't expect to make something good, and they surprised themselves with their output. Note that I didn't say the *system* surprised them.... Casual creators are most effective when the interactive system doesn't take credit for the finished creative product, but let the user feel ownership of their own creativity (even when its a collaborative effort between the human and the system).



Everyone is creative, and a lot of creative time is shared with others. From quilting bees to game jams, humans communicate and bond through creative activity.

Figure 26.4

Figure 26.5

A bit about creativity...

Casual creators are systems that help people feel creative

Why do we need software to do help us be creative? What's so important about regular people being creative?

- Because creating is fun and meaningful, even when we aren't great artists creating great art
- Because for every portrait in a gallery, there are 100 doodles pinned to a refrigerator
- Because adults convince themselves that they aren't creative (or worth being creative), and they need encouragement and support

So to make systems that assist creativity, we should know a little something about how creativity works. We often talk about creativity like its a magical gift, bestowed on some lucky few geniuses. Its magical, and special, and we can't control it, so why try to make software to help?

But this isn't right: we actually know quite a bit about creativity! All people are creative by nature, even if only in small ways. We also know there are also things that *inhibit* creativity. Instead of talking about *which* people are creative (or "creative enough"), we should talk about *when* people are creative and *what conditions* let creative people be creative. So lets look at a very brief introduction to what we currently know about creativity:

Personal- and Historical- Creativity

We use "creativity" to mean a lot of different things. Consider the following statements:

- Mary Wollstonecraft Shelley was a creative visionary
- You'll never believe these 10 creative uses for old soda bottles
- Creative industries demand increased copyright protections
- Creative art activity can improve mood and health outcomes in seniors

We can see different kinds of creativity:

- creativity that everyone can do (art and craft projects) vs specialist creativity (professionals and famous creators)
- creativity that's about *being clever and suprising and new* vs creativity that is about producing artwork
- "regular" creativity vs famous-forever creativity

Creativity research refers to the last distinction as "P" and "H" creativity, for personal and historical creativity. (Boden, Computer Models of Creativity). Even if we didn't know what causes historical creativity, we actually understand a *lot* about personal creativity.

Flow

Mihaly Csikszentmihalyi is one of our most famous thinkers on how creativity happens. He's most famous for the idea of **Flow**, a pleasurable state where:

- There are clear goals every step of the way
- There is immediate feedback to one's actions
- There is balance between challenges and skills
- Action and awareness are merged
- Distractions are excluded from consciousness
- There is no worry of failure
- Self-consciousness disappears
- The sense of time becomes distorted
- The activity becomes autotelic

Flow isn't specifically about creativity (its used in a lot of video games, too), but it seems to help people feel creative. Other fields, like improv theater, also have lessons about creativity, like how people are more creative when they feel emotionally safe ("yes-and") and have prompts or rules to work inside of. Here are some general conditions that help people feel creative:

- they feel safe: like they won't break anything or make mistakes, and, if they do, it's ok and fixable
- they know what to do next, or, at least, know what their options are (blank slates can be scarier than creative prompts!)
- the people and systems around them acknowledge and respond to their choices

Environmental Creativity

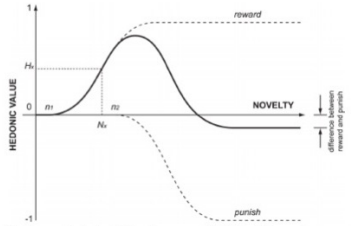
Csikszentmihalyi also talks about *where* creativity happens. In our collective imagination, we often picture geniuses operating alone, isolated in their creative spaces, only emerging to push a finished artwork out the door. But this is not an accurate picture! Almost all creativity happens in an environment of creativity, which he identifies as

- the "domain" (a material/culture with rules),
- the "field" (experts in that domain who are both knowledgeable and listened to by others)
- the individual (our 'lone' genius)

Even when the individual is creating, they are pushed around (and push back on) the rules of their domain. They are aware of their field, and how their work will be received in it. Their work may go on to change the rules of domain or the opinions of their field, but they also work against the backdrop of all the past changes caused from previous artwork.

We make art relative to the rules of our domain: intrinsic rules like how paint can reflect light, or extrinsic rules like what subjects can be painted. We also have a community, whether our friends that we share personal projects with, or the world of professional art critics, and we create art in relationship to them.

One cool project that explored Csikszentmihalyi's domain-field-individual creativity model is Rob Saunder's Digital Clockwork Muse. Saunders created a community of digital artists that could produce virtual "paintings" and share them with each other. Each one had their own style, and they could even self-organize based on their style and what they liked. But what they like was *relative* to the work in their domain. Saunders modelled their preferences with a function called the Hedonic Curve, which says "I like things that are different from what I've seen before but **not too different**."



The "hedonic curve", also called the Wundt curve. We like things that are familiar enough to understand, but not so familiar that they're boring.

Creating as Community

Not only do we create *in* a community, but we also create to *build* communities. Ravelry, Scratch, Tumblr, DeviantArt, and Archive of Our Own are a few examples of spaces where communities develop around creative activity. They all share a lot of features:

- users can browse works, and upvote or comment on them
- users have a persistent identity
- they post new works and comment/upvote with the same identity
- there are self-organization features, like "following" or groups that let users express bonds with each other
- subcommunities form around shared interests and dislikes (that Hedonic curve!)
- ideas, taboos, rules, new genres, tutorials, memes, alliances, and friendships all develop and propagate through the network of social bonds

Of course, creating-as-community is not an internet phenomena; real-world creative community is a continuous thread from quilting-bees to Maker Faire. there's even been suggestion that creativity (and working with your hands) makes people better humans and better citizens (see John Ruskin's Victorian craftmania, craft-kits as a midcentury cure for teenage delinquency, and contemporary research in how crafting makes people happier and healthier)

Creativity is **world-changing genius**, an **idle pastime**, a **social bond**, and a **universal human experience**. This section should have shown you that there is a *lot of complexity* to creativity, but that for all the complexity, we know what makes it work. That means we can write software to support creativity for everyone! To quote philosopher, Jerome Bruner: "Stupid people create for each other as well as benefiting from what comes from afar. So too do slothful and torpid people. I have been speaking of creativity, not of genius."

Terms to remember from this section: **P-H creativity, domain-field-individual, flow, hedonic curve**

Special properties of Casual Creators

Casually-creative people often don't have a *grand plan* of what they want to make. They want to have a good experience while making something, explore their creativity (and the tool) and only secondarily care about getting the final product. Ask any habitual knitter or ceramicist how many extra socks and glazed cups they have lying around! A term for this is *"autotelic"*: doing something for the pleasure of doing it, not for the result.

This difference means that the casual creator tools work very differently than traditional *high-control* professional creativity tools. Here are some interesting properties of casual creators:

Power-vs-control tradeoff

Since users are more flexible with their goals, a casual creator can get away with having fewer options, a simpler UI, and a smaller range of what it can make. But it can use that narrower possibility space to give the user *more support*. Consider the difference between Maya and a hypothetical 3D-printable teapot-making app. In Maya, the user may be making a 3D printable teapot *or* special effects for a film *or* animated characters for a game *or* a VR walkthrough of a house *or* anything else you can make in Maya. We don't know what they're making, so we can't write software to help them, they're on their own to do the right work. For our teapot app, though, we know *lots* about 3D printing and teapots:

Figure 26.6



What makes a good teapot? We know a few things, so we can use that knowledge to build AI assistance. We could make a generative space that contains only good teapots, and no bad, broken, or leaky ones.

This knowledge means that we can automate a lot of the tasks, and have the teapot dynamically correct itself as the user changes it. The user has lost the control to make non-teapot things, or unusual-yet-valid teapots (e.g. multiple spouts), but they gained power to explore a whole space of safe and functional teapot designs with a single gesture.

The most identifiable property of a casual creator is that the user trades some control for assistive (usually generative) power. As a user, your possibilities are limited, but you feel **superpowered** because whatever you can make, you can make faster and better than you could on your own. As you've just read from the Creativity section, that makes users take bigger creative risks, too.

Where does all this power come from? The casual creator needs some kind of system that turns a little action from the user into a *big* or *smart* or *surprising* action from a system. All casual creators have at their hearts a *generative pipeline*. Whether mechanical like the operation of a pottery wheel or digital like stroke-lines from digital paint brush, these create the *power* of a casual creator, and are what makes them feel so magical.

Generativity:

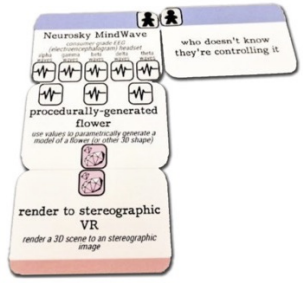
A *generative method* is a function that takes some input (a number, a curve, a photo, a sound wave, etc) and creates something new (often bigger or more complex) from it.
We can chain these functions together, turning some kind of **user input** into some kind of bigger complex generative output. Generative methods can take many kinds of input, and create many kinds of output. For more on the kinds of methods that are available to you, read my zine "Encyclopedia of Generativity" (it's an Ikea catalog of Generative methods!)

In a casual creator, the output is our system's finished **artifact** (like a 3D model or a game level or song or image).

For example, imagine our teapot-maker has a single slider (volume of tea), and it uses that number to generate a teapot of that size. We have a **possibility space** that is one-dimensional (a number from 4-32oz, plus all the teapots generated from those numbers). The **model possiblity space** refers to the data structure used to store/generate (the numbers, in this case). The **artifact possibility space** is the space of output objects (the teapots).

Its often useful to think about these two spaces **separately**. In our teapot-maker, the model-space is linear and continuous: we can build a slider for it, and the user will understand that they can slide the slider back and forth to see the whole possibility space. The artifact-space that these values are translated into *may not* be linear and continuous, though! When the slider is slid to make a very-small teapot, maybe the handle disappears. The size or design of the spout may change in unexpected ways to deal with fluid dynamics, or the lid's size may remain constant as the body-size increases. There may even be hidden surprises generated at certain slider values (the Spore creature creator did this).

There is also unpredictability and non-linearity in how the artifacts are perceived or used. Some teapots fit well in one-hand, or two, or are just right for one person, or for a party. *The algorithm might be able to predict some of this but, often only the user can really judge the shape of artifact-space.*



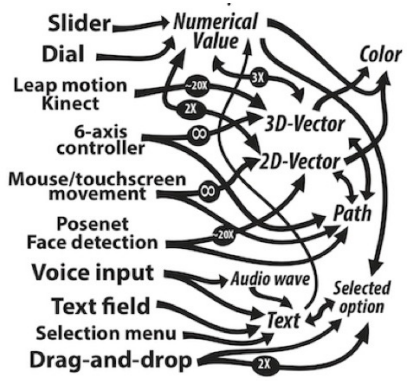
The Generominos (www.galaxykate.com/generominos) are a deck of cards I made to help designers explore new ways to build these pipelines.

Model-spaces can take many different shapes: a two-dimensional space where we set both the volume *and* height of the teapot, or a three-dimensional one where we also set the squareness of the sides. You could display the possibility-space of the two-dimensional model by graphing it on a grid, but its harder to do this with three-or-higher dimensions. Some models contain things other than numbers, like text (write your name on the pot) or selections from a list (decorated with roses, polka dots or stripes) or curves (set the shape of the spout and handle) or any of many other data structures.

Why do different *shapes* of model spaces matter? Casual creators can use generative spaces in different ways, giving users ways to *map* or *navigate* or *annotate*. Here are a few ways to use spaces:

Figure 26.7

- Challenge the user to get to a certain part of the possibility space, ie, "close enough" to some desired point/artifact (Let's Create: Pottery, BECOME A GREAT ARTIST IN JUST 10 SECONDS)
- Give users "achievements" for reaching all the parts of the space (see "Petalz", by Risi et al)
- If there are paths that users often take (from one kind of artifact to another), tell then when they are making common creative moves, challenge them to find an uncommon path ("Dryad" Talton et al)
- Let users name artifacts they discover (No Mans Sky), or share an artifact and watch others remix it (Picbreeder, Sketch-a-bit)
- Use the possibility space as part of a game (Galactic Arms Race) and let users watch the whole game's collective movement through the possibility space.
- Create an evolutionary algorithm by taking two points in the space, and finding points between them. This only works for mostly-linear spaces (and maybe grammar-based spaces if you are very clever)



A few mappings from various input devices to the kinds of output they produce. Some kinds of input can be turned into other kinds of input, but nothing of it can magically appear. When designing a casual creator (or any generative pipeline), consider what kind of data you want to use; that will determine what kind of input controller you can have.

Unfolding control

As a tool-builder, you can also choose how many choices and controls the user has at any time. Sometimes **limiting** the number of choices a user has is a good thing! Choice paralysis is a real phenomenon that can inhibit creativity. Not only do users have to come up with something to make, they also have to guess how to make it. Remember that the Flow definition includes:

- There are clear goals every step of the way
- Distractions are excluded from consciousness

Here are some ways to give the user limited choices *in the moment*, but still allow them to gain access to more control *at their own pace*:

- a **starting artifact** instead of starting with a blank slate
  - a binary choice: "yes" or reroll/erase
- **prompts** or **themes** (can be community driven)
- **challenges**, artificial restrictions, e.g.:
  - "create using only this tool"
  - "make something that fulfills this requirement"
- **unfold controls** by letting users unlock new controls over time
  - can be monetized as microtransactions, or a challenge reward

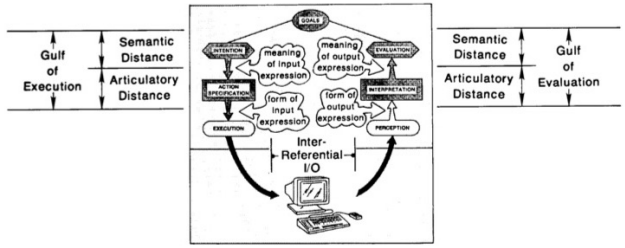
I like to think that there are different *kinds* of movement through the space, depending on how much agency and control the user (or the system) is taking at any moment.

- **control**: traditional artist practice: the user is controls
- **steer**: the user and the system are both working together. The system has a *will* and *agency* to create in a direction, but the user is pulling it in their own direction. Or the user takes an action, and the system elaborates on it.
- **surf**: like surfing the web or surfing a wave, the user lets the system take charge. They can shift themselves to move in different direction, but it feels like the system is a world of content, and they're just angling through it.

Moment-to-moment, users may change their navigation style, or different apps may encourage one style over the other. Picbreeder, for example, only allows steering, while Panoramical is a form of surfing. MS Paint is a canonical example of full control, but sometimes watercolor brushes feel more like steering.

This pairs well with the previous ways of navigating the possibility space. Do you let the user control where you are going? Or are there forces controlling where they can go, or suggesting new directions and paths to take?

Figure 26.8



How do users get from "what we want to do" to "how do we do it" to "did it work?" In the mid-1980s, Ben Shneiderman used a term called "direct manipulation" to describe how good UI could make the distance between the user and the system seem to disappear. Usability guru Don Norman then coined the terms "gulf of execution" and "gulf of evaluation" to identify where that magic "directness" is broken or hard to achieve. (img from his 1986 paper "Cognitive Engineering")

The gulf of execution and the gulf of evaluation

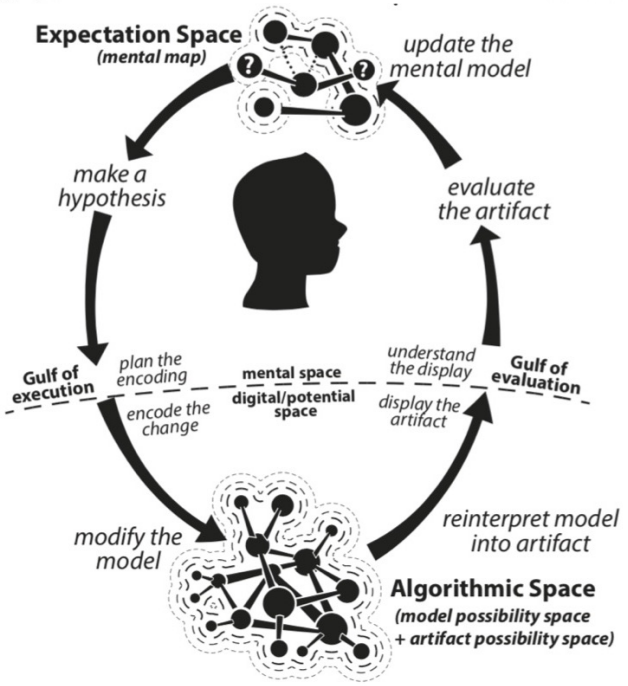
How does the user *change* the artifact (sliders, text fields, gesture-control, a menu)? How long does it take them to change something? Is it obvious *how* to use the UI to make that change? The distance between the user's intentions and their ability to enact them is the **Gulf of Execution** (Don Norman, "Cognitive Engineering").

As a designer, you choose how users can see and explore the possibility space (model or artifact) Do changes in the artifact reappear *right away* after user input? Some generators are fast enough that the artifact can update continuously as the user edits it. Other generators may need seconds *or even hours* to update.

Some kinds of artifacts also take *time* for a human to evaluate. We can tell whether we like a piece of visual art in a fraction of a section, but how long does it take to evaluate a poem or a novel or a song?

What about artifacts that are *experiences*, like a game level or a new type of gameplay? All artifacts have some properties that can be evaluated at a glance, and others that take time (how our teapot looks, vs how well it pours without splashing). The time it takes for the generator to create the new artifact, plus the time it takes for the user to evaluate it (and how accurately they can evaluate it) create the **Gulf of Evaluation**.

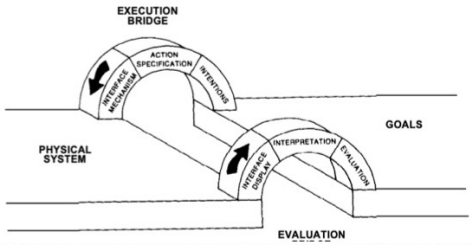
The size of both the gulf of evaluation and the gulf of execution determines how quickly your users can make things, explore the tool, and discover hidden possibilities.



My version of the grokloop. Not only are we trying to understand an interactive system (Shneiderman and Norman), but the system is a front for a vast generative space that we are navigating and mapping as we travel.

I call this the "grokloop", after the Heinlein term "grok": "to understand so thoroughly that the observer becomes a part of the observed". When the Gulf of Execution and Evaluation become so fast that the user is constantly updating their mental model of the system, then their experimentation becomes an invisible mental blur and expertise becomes effortless. We saw this on Spore: novice users experimented so fast and gained such rapid feedback (multiple times a second) that they quickly gained insights on aesthetic creature composition that one would expect from an advanced professional artist.

Figure 26.9



As system designers, we need to build bridges so that users can quickly cross *both* the gulf of execution and the gulf of evaluation. Where is the bottleneck in your loop? Is in in specifying the action, manipulating the controls, recalculating the output, rerendering the artifact, or is it limited by the speed at which the user can "read" the new output? The loop can only get fast once *all* the bottlenecks are solved. (img from "Cognitive engineering")

Dealing with a slow grokloop

Many casual creators can have a very fast grokloop. As users are flexible about the level of control over the creator, a long execution loop can be shortened by replacing complex controls with more automation and generativity (the far end of this process is to only give the user a big red button labeled "make another" than they may bash at any speed) Any casual creator where the user can judge an artifact's *instantly* by looking at the output will have not much of a gulf of evaluation.

"Slow" Creators

But several kinds of desirable artifacts can't be judged at a glance. Things like music, novels, and games must be experienced *over time* to be judged, which as a grokloop can't be shorter than the time needed to experience the artifact, can mean a grokloop cycle of minutes, hours or days!

This has traditionally been the bane of casual game-making apps. No shortage of game designers have wanted to make a game-of-making-games, from 1983's Pinball Construction Set to the Spore Adventure Creator to Super Mario Maker. Yet so many of these end up feeling like grinding, frustrating playtesting work. On the Spore Adventure Creator, most games were published without ever having been played through, so nearly all were buggy or unplayable.

There are a few tricks to dealing with slow grokloop spaces however:

- Quick-to-play games (Warioware, Wevva) can be playtested fast, and iterated on with only a minute or so per grokloop cycle.

- Allowing editing *while* playing can speed up iteration (this is true for professional game dev as well!)
- Well-constrained spaces (like pinball) with fewer emergence gameplay problems are more robust and have fewer bad artifacts to route around
- Partial evaluation (Mike Cook's Danesh, Sentient Sketchbook) can give users *some* information
- When in doubt, **lie**! False evaluations can give users unearned confidence (not good if you care about the output, but fine for an autotelic experience)
    - Entertaining evaluations are evaluations that, even when unhelpful, are interesting and a good experience
    - The Sims, Rollercoaster Tycoon, and BECOME A GREAT ARTIST IN JUST 10 SECONDS all have evaluations by unreliable simulated critics, and yet it's fun to try to please the fickle little AIs!



Google's "Code with Anna and Elsa", and its grandparent, Seymour Papert's Logo

Casual Creator programming languages

These casual creator patterns hold across many kinds of creativity software, from poetry-generators to 3D printing apps. I'm also discovering that they hold true for some kinds of *programming languages*, too. Casual creator programming languages include Logo, Scratch, Joy.JS and my own Tracery. These languages share the previously-mentioned casual creator goals: encouraging users to feel safe and creative exploring a possibility space, while not striving for total control. There are too many other patterns, but I've identified a few properties for programming languages to become "casual-creator-y":

- **bridging the grokloop:** These languages use a variety of ways to bridge and shorten the grokloop. Scratch uses draggable syntax, rather than the user having to remember keywords, though users still have to manually re-run the program and wait for it to execute before they can evaluate the results. Joy.JS (Nicky Case's riff on Logo) has sliders instead of typed numbers for any "magic numbers" in the program, and automatically re-runs the program as the user adjusts these values, for a very short execution/evaluation loop.

Figure 26.10

- **scaffolded:** somehow the language trades power for control. This means that the language may not be *general purpose*, but has a limited possibility space. Logo is *technically* a general purpose language, but almost everyone only ever uses the turtle graphics! Casual creator languages have *constraints*, things they can't do (like Tracery's near-inability to model state), but also *augmentation*, like Joy.JS's re-running, or Logo's turtle-simulator. Even *autocomplete* in a code editor is a casual creator property. One can only autocomplete a language with a limited syntax, but the user gets a boost of power from it!
- **unfoldable:** not every part of the language needs to be known to write a program. In Tracery, many users have never used the modifiers or push/pop syntax, because they can write satisfying programs with out it. The language starts with simple syntax, which the user can expand to more complicated syntax, *as they develop comfort with the language*. "Sugaring" (providing a simplified "shorthand" for common syntax patterns) is common in these languages, with some users never "unfolding" the sugared syntax to get to the higher levels of control
- **little:** a "little language" (coined by Jon Bentley, 1986) is a domain-specific, portable language that plugs in anywhere, like Regular Expressions. One property of little language is that their programs are so constrained that they can sometimes be created *by other programs*! A simple input format means that *anyone* can create an editor, or use existing tools for that format to create a new pipeline. Tracery uses JSON, which means that Cheap Bots Done Quick could safely import Tracery "programs", people can copy-paste them, augment them, or remix them, or even generate them with Tracery itself. A "little language" is **portable, remixable, reusable, & augmentable**.
- **welcoming incompleteness:** A writer never writes anything in order. Any book manuscript (or program) is always littered with "TODO:FILL IN LATER" notes. But many programs will refuse to run unless everything is specified. Often, too, a program encountering a bug will crash, not knowing what to do next. But in a casual creator language, we know enough about the space to "patch over" any missing pieces or bugs. Tracery never crashes or refuses to create output. Instead, it patches in noticeable-placeholders for missing pieces. Expanding a rule like "#animal.a# went to the #place#" with no other symbols or modifiers defined will return "((((animal)).a)) went to the ((place))". **The user can tell that something is wrong, but can keep authoring until they fix it at their leisure.**

"Simple things should be simple, complex things should be possible" - Alan Kay on designing programs (or programming languages)

A few patterns

Throughout this research, I've found dozens, well, **hundreds** of casual creators in the wild! Each one that I look at is unique in some ways, but across the song-makers, avatar-makers, pottery-makers, jewelry-, game-, novel-, and painting-makers, there are a few common patterns that I see. The rate of new patterns is decreasing, so I guess I'm closing in on a "definitive" set, but I anticipate that I will continue to be surprised as more are discovered (and invented!)

- **Instant feedback**: Whenever possible, create instant feedback. If some parts of the pipeline can't render that fast, show what you can. The Spore pipeline couldn't rerender the creature textures, but could show the mesh updates, so users edited an untextured creature. They didn't notice the omission, but they did notice the magical speed.
- **Entertaining evaluations**: If you can't create instant feedback (or even if you can), create evaluations that are fun, entertaining, or surprising. We love to create in a social environment, so when no other humans are around, its fun to please (or be judged by) artificial creative companions. "Yes-and" isn't just for improv, its also critical for a casual creator to make a user feel creatively safe and welcomed.

- **Gestural control**: If your evaluation is fast enough, how can you narrow your gulf of execution? As the popularity of the computer mouse was the defining moment of "direct manipulation", we have a range of motion controls available now: hand tracking, gesture tracking, face tracking, brain sensors and more. Many apps attempt to use these for tradition fine-grained (and often frustrating) control, but they work marvelously well for casual creators. Few things are faster and more fun than waggling your fingers and seeing strange generative sparks emerge
- **Space annotation: users mapping spaces**: How do we understand and move around a space? Kevin Lynch's 1960 Image of the City lays out the mental features people need to understand a physical space (landmarks, paths, nodes, edges, and districts), but it turns out these are also useful in a metaphorical possiblity space. If you can identify regions in the space (Petalz) or paths (Dryad) you can tell users when they are on them, or challenge them to find more. Users can navigate by, or name and discover, landmarks. If you have a way of understanding, valuing, or moving through *physical* space, try in a *possiblity space* and you may discover a new pattern.

Figure 26.11

- **No blank slate**: Creativity is terrifying! After WWII Winston Churchill writes about being afraid to start painting, as any movement could be the wrong one *("...the empty brush hung poised, heavy with destiny, irresolute in the air. My hand seemed arrested by a silent veto.")* We don't deal well with blank slates. Present the user will *something* to push against, a prompt, a half-finished sketch, even a title!
- **No invisible holes (default to always working)**: Whenever possible, have a possibility space with no *broken* content. Crashes, hard failures or syntax errors make users freeze up and become creatively conservative.
- **Self promotion**: Let users self-promote through your tool. Fantastic Contraption let users embed creative solutions as links (which linked back to the app), promoting their cleverness while also spreading the game. Opus Magnum was, well, a magnum opus of this: it created glossy animated gifs of creative solutions, perfectly ripe for posting on Twitter.
- **Group learning**: Learning and increasing master is pleasurable, and its often best done as a collective. Users can self-promote by showing weird tricks they've learned, which then spread and are remixed through the community. Both Spore and Galactic Arms Race had moments where an unknown part of possibility space was opened, and instantly spread.
- **Create channels for users to communicate**: When people are creative, they will want to create structures around that. Spore spontaneously developed "mailboxes" (creations that were the designate message boards for users) since we didn't provide them, as well as contests and creativity themes. By piggybacking on Youtube (and later Twitch), Minecraft became a hit by letting users build and own audiences.
- **Joys of generativity**: There are many "joys" of generative content. Some are just in discovering odd artifacts, and performing surprise (this is very Twitchable). Generative content can also provide seed-material to other creativity (the improv "pull-a-prompt-from-a-hat" model). It often gives the user "permission" ("the computer said to do it!") to try new things (see party games like Twister, Mad Libs or Sex Dice)
- **Repetition**: "Reflect, refract, and apply repetition", the mantra of easy generativity! There is a reason so many generators make flowers: symmetry makes things look good, and its hard for humans to achieve without mechanical aid. Likewise, filtering or distorting is a way to create generative expressiveness out of some user-provided source data (fun-house mirror effect)

- **Adopt and collage signifiers**: Semantically-heavy data (names, photographs, emoji, celebrities, brands) can add meaning and interest to generative pipelines. Like the fun-house mirror effect, there is a strange joy in seeing a familiar symbol manipulated in unfamiliar ways (@glitchlogos)
- **Make it real (3D printing, on-demand prints, sharing)**: There's something magical about seeing a digital artifact in the real world. 3D printers are just one step of a generative pipeline. Also, the real world is where a lot of social interaction happens. Desk tchotckes, fashion, and gifts are all places that casually-created artifacts could become social items.

A little light reading: References

Gauntlett, David. Making is connecting. John Wiley & Sons, 2013.

Boden, Margaret A. "Computer models of creativity." AI Magazine 30.3 (2009): 23.

Saunders, Rob, and John S. Gero. "Curious agents and situated design evaluations." AI EDAM 18.2 (2004): 153-161.

Csikszentmihalyi, M. "Implications of a systems perspective for the study of creativity." Handbook of creativity (1999): 313-335.

Csikszentmihalyi, Mihaly. Flow: the psychology of optimal experience. HarperPerennial, 1990.

Shneiderman, Ben. "Direct manipulation: A step beyond programming languages." Computer 8 (1983): 57-69.

Norman, Donald A. "Cognitive engineering." User centered system design 31 (1986).

Bruner, Jerome S. "The conditions of creativity." Contemporary Approaches to Creative Thinking, 1958, University of Colorado, CO, US; This paper was presented at the aforementioned symposium.. Atherton Press, 1962.

Leder, Helmut, et al. "A model of aesthetic appreciation and aesthetic judgments." British journal of psychology 95.4 (2004): 489-508.

Hickman, Craig. "Kid Pix, the early years." http://red-green-blue.com/kid-pix-the-early-years/.

Now what?

You've finished this zine, and what a long one it was! I hope that it's opened your eyes to casual creators, whether you play them or make them already. It is my fond wish that coining the term "casual creators" will let afficionados of these strange little systems meet and share knowledge, like "video games" does for its community. Perhaps I'll see you at the first meetup!

Figure 26.12

Chapter 27

Conclusion

This dissertation defines and motivates the field of "casual creators", and in doing so, makes the following contributions:

- define what casual creators are (chapter 1)
- identify how casual creativity differs from task-oriented creativity, and why we expect to find different patterns to support it than those that have been identified and studied in professional and virtuosic creativity (chapter 4,chapter 5))
- define a starting set of terms and frameworks (chapter 6, chapter 7 and chapter 8) for understanding casual creators, to establish a shared foundation upon which we can build additional knowledge of casual creators
- lay out the principles that underlie casual creators: interactivity, generativity, possibility spaces, navigation, and social interactions, and examine how they work in interactive or generative systems in general, and also how they combine in casual creator systems (chapter 3)

- identify the shared properties of casual creators across many diverse genres and mediums, from character creators and game-makers, drawing tools and pottery wheels, poetry bots and programming languages (chapter 9, chapter 10, chapter 11, chapter 13, chapter 6, and chapter 14)

- propose a handbook of patterns, a non-exhaustive menu of tools that can be applied to casual creator systems, or systems that seek their properties (chapter 16, chapter 17, chapter 18, chapter 19, chapter 20, chapter 21, chapter 22, chapter 23, chapter 24, and chapter 25)

- motivate the identification, study, and creation of casual creators, and catalyze the formation of an interested new community of research, development, and construction of shared knowledge chapter 2

In this dissertation, I defined a "casual creator" as an interactive generative system that:

- privileges enjoyment of the creative process above productivity

- produces artifacts within a limited-yet-meaningful domain space, enabling automation and support, both passive (encoded into the domain model and system constraints) and active (responding to user actions)

- supports a state of creative flow by restricting choice and preventing hard failures while allowing rapid iteration

- results in the user's feeling of pride and ownership toward the produced artifact, and sense of pride in their own creativity.

Given that definition, I attempted to answer the following questions:

What defines a "casual creator", and what new experiences do casual creators enable?

We can see the definition of a casual creator above, but this dissertation also identified over a hundred casual creators (and systems with properties of casual creators). Several of those systems were built by myself, as explorations of particular properties of interests (entertaining evaluations, 3D printing, gestural interfaces). Most are existing systems that share common experiences like *autotelic* creativity, social functions of creativity, exploration, and complicated feelings of ownership/discovery. These experiences are made possible by the way that casual creators enable safe, rapid navigation through a possibility space, which is itself made possible by the combination of user interaction and a generative pipeline.

How are those experiences predicted or explained by existing theories of creativity?

In chapter 4, we saw a great range of theories of creativity. These theories explained that creativity is not isolated in the mind of a "lone genius", but instead is situated at the intersection of the self, domain, and field, and at the intersection of self and the artifact. We create artifacts with knowledge about the domain, for a field of experts (or colleagues), and we do so through cognitive processes *embodied* in our interactions with, and investigations of, the artifact that is in front of us. We also saw, separate from the theories of creativity, practical world-tested techniques to enhance creativity, by maintaining creatively "safe" spaces, encouraging (or augmenting) divergent thinking, and building computational "creativity support tools".

What frameworks do we need in order to understand and analyze casual creators?

In chapter 6 we explored existing frameworks for generativity, from domains of art, natural history, and games. While each domain had useful and truthful perspectives on "what is generativity", I defined my own framework, the Generative Framework of Generativity (chapter 7) to be able to examine generative systems as *pipelines of generative methods*, where human interaction and evaluation, and algorithmic computation, could occur in *many parts of the pipeline*, flexibly. In chapter 8, we saw perspectives on how people come to understand systems through interacting with them. *Flow theory* explained why some explorations feel effortless, while *direct manipulation* explained why some systems feel sluggish and mysterious to use while others feel direct and obvious. The loop of *reflection-in-action* and the *gulfs of execution and evaluation* gave us language to talk about how this learning takes place in a repeated cycle of hypothesis-action-evaluation, and why that cycle often takes so long. I rolled all of these theories together into the *grokloop*, a name for the hypothesis-action-evaluation cycle when it is used to explore a generative space.

Are there subcategories within casual creators that have shared properties?

Generative possibility spaces (and how we move through them) are the defining features of casual creators. Any given space will have different qualities (full of blandly acceptable artifacts, or perhaps consisting of mostly terrible or broken artifacts with a few unique gems) and a different shape defined by its generative pipeline. Narrow or shallow spaces have shared techniques to make the space (or at least the navigation) seem more interesting. Spaces defined by parametric techniques can be modeled as an N-dimensional cube and can be explored in a variety of useful ways, including by genetic algorithms. Many casual creators use gestural input or drawing metaphors, but are often mediated or extrapolated in

interesting ways. Some domains have a "slow" evaluation step by their nature (novels, games, music) and these require a selection of clever strategies to prevent a slow grokloop from derailing the exploration and learning process. Programming is a creative domain with many difficulties in its gulfs of execution and evaluation, and there have been whole constellations of approaches to solving the issue. Tracery provides one example of a slow grokloop, but also a demonstration of the *situatedness* of casual creativity in an heterogeneous ecosystem of creativity tools and experiments.

What set of design patterns provide prescriptive guidance for designing new successful casual creators?

In addition to all of the patterns that are linked to properties of particular kinds of generators (slow creators, drawing tools, parametric spaces) this dissertation defined ten categories of general-purpose patterns. Some of these are playful re-envisionings of the creativity theories, re-imagining the generative pipeline as an art critic or an improv partner, or creating ridiculous UI framings to encourage creative "safety". A few other patterns observe casual creators that are embedded in contexts of social meaning, where users gather to map a system as a collaborative exercise, or try to outdo each other in pushing against the limitations of the creator. Some reflect on the particular "magic" of generativity, like the ability of computational generators to make perfectly symmetrical copies of wobbly human gesture, or turn a hand-drawn sketch into a colorful model "living" in virtual-space, or turn set of small choices into something that can interface with complex 3D-printing machinery. Finally, we saw ways that generative pipelines can make use of (reinterpret, juxtapose) signifiers that have meaning to us socially or personally.

If these design patterns are used to design new casual creators, do they ease and improve the design process?

Will the framework of "casual creators" help people design and interpret these kinds of systems? While writing this dissertation, it was impossible to keep adding all of the casual creators I've come across: checking *BoingBoing*, *ThisIsColossal* or Twitter in the morning would often lead to two or three new additions each day[1]. Despite not being aware of each other's existence (or being in the same field or domain), these systems often presented remarkably similar sets of patterns. If they *did* know about the patterns available, would they make different choices? Not all casual creators are *good* (*Spore Galactic Adventures*[s217]), so would using this framework allow them to predict and overcome their challenges? I hope so. I evaluated this framework by making a zine and surveying some busy interactive-generative system designers, and they agreed that it seems useful to them and their colleagues. Time will tell if a community will build around the idea, and they begin creating systems using it.

27.1 Future work

Much work remains to be done in casual creators. This dissertation declares dozens of patterns, but none of them have been formally evaluated. What does a "casual creator" evaluation even look like? Perhaps one could go line-by-line in the definition, and create a set of evaluations for "enjoyment above productivity", "limited-yet-meaningful space", "creative flow", "rapid iteration", and "pride and ownership". What would evaluation those tell us about new or existing creators?

[1] If in the past years, I got excited about your system as a "casual creator" example, and it *doesn't* appear in the dissertation, it is only because I have limited space and there are *so many*. Though I did promise to include the slightly-saucy "entertaining evaluation"/improv partner at `http://seoi.net/penint/`, so here it is.

I also hope that this work won't be done by me, but by the other scholars and designers who take up the banner of casual creators. If this dissertation succeeds, then many more people will use the term "casual creator" to describe software. A few of you reading it will find parts of it particular resonant with your interests (casual languages, entertaining evaluations, grokloop learning, slow game makers), and be inspired to advance and explore those parts further.

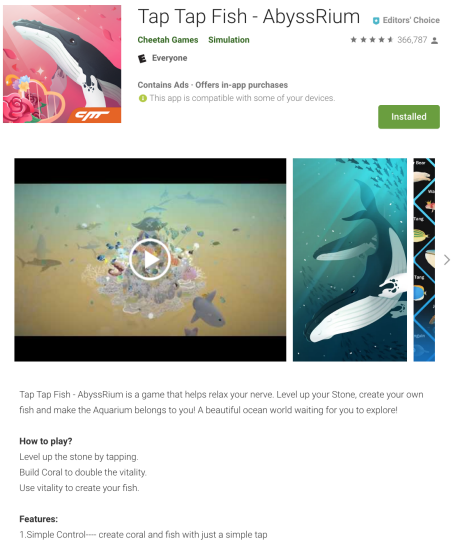
I hope that this dissertation plants many seeds in the minds of its readers. After reading this, perhaps you will look at familiar systems, and see new patterns in them. When you encounter new systems, you can trace their patterns back to a greater heritage of roots. Above all, I hope that you will find others who are also interested in this field, and that you find shared purpose across the boundaries of discipline and domain.

Chapter 28

System Index

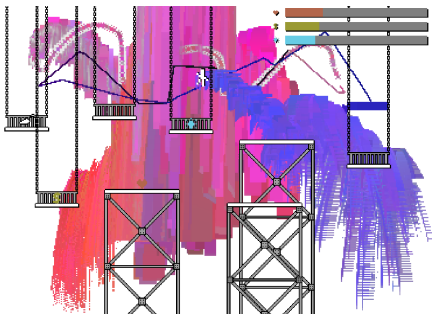
S1: Tap Tap Fish - AbyssRium

Creator: Cheetah Games
Url: `https://play.google.com/store/apps/details?id=com.idleif.abyssrium`
Platform: Android
In chapter: Pattern - Safety
Description: An idle game about curating and building up a fanstastical coral reef. It encourages users to take photos and share them on social media for currency (a Self Promotion pattern).



S2: Action Painting Pro

Creator: Ian MacLarty
Url: `http://ianmaclarty.com/action-painting-pro/`
Platform: web
In chapter: Drawing Tools
Description: A platform game in which the player collects paint and leaves trails behind them.

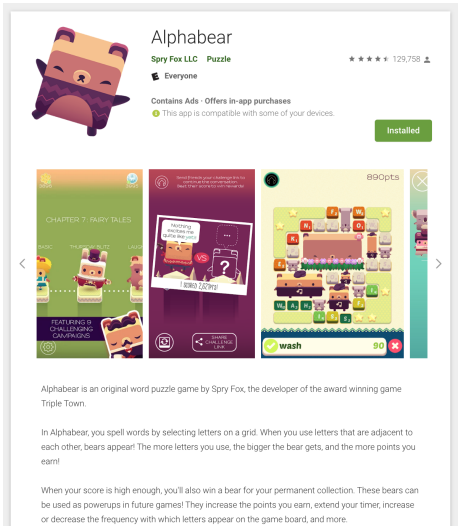


S3: ADA - analog interactive installation / kinetic sculpture / post-digital drawing machine



Creator: Karina Smigla-bobinski
Url: `http://www.smigla-bobinski.com/english/works/ADA/index.html`
Year: 2010
Platform: installation
In chapter: Drawing Tools
Description: A giant inflated ball covered with graphite stubs, which installation-visitors can push around to draw on the gallery walls.

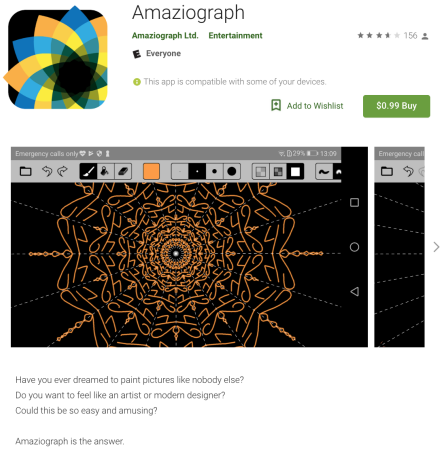
S4: Alphabear



Creator: Spry Fox
Url: `https://play.google.com/store/apps/details?id=com.spryfox.alphabear`
Platform: Android
In chapter: Pattern - Safety
Description: A word-finding puzzle game in which the words that the players find can be socketed into template-generated sentences and shared to social media (a Self Promotion pattern).

S5: Amaziograph

Creator: Amaziograph Ltd.
Url: `https://play.google.com/store/apps/details?id=com.amaziograph2.android&hl=en_US`
Platform: Android
In chapter: Pattern - Repetition, Repetition, Repetition
Description: A mandala-drawing app, and symmetrically generative tool.



S6: Apples to Apples

Creator: Mattel (publisher)
Year: 1999
Platform: boardgame
In chapters: Creativity, Pattern - Entertaining Evaluations
Description: A game in which players create juxtapositions, and decide which is the best.



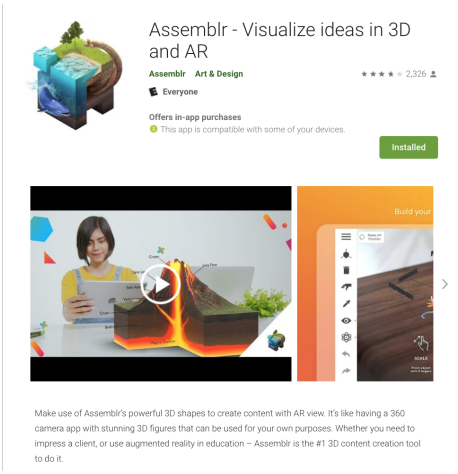
S7: ART SQOOL

Creator: Julian Glander (Glanderco)
Url: `https://artsqool.cool/`
Year: 2019
Platform: OSX
In chapters: Narrow and Shallow Possibility Spaces, Pattern - Entertaining Evaluations
Description: A game about pleasing an impossible AI art professor.



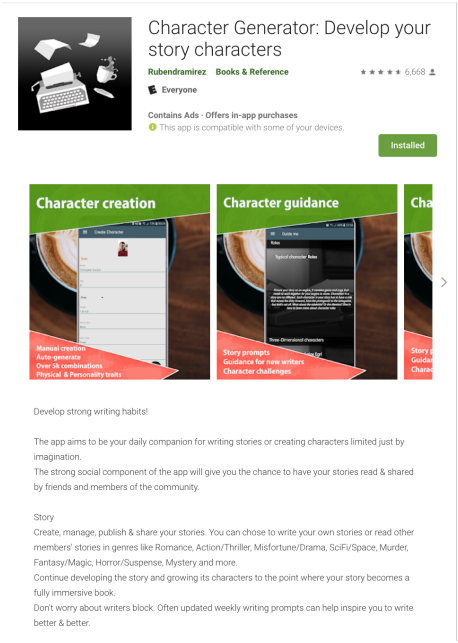
S8: Assemblr - Visualize ideas in 3D and AR

Creator: Assemblr
Url: `https://play.google.com/store/apps/details?id=com.octagonstudio.assemblr&hl=en_US`
Platform: Android
In chapter: Central Themes
Description: A simplified 3D modelling app



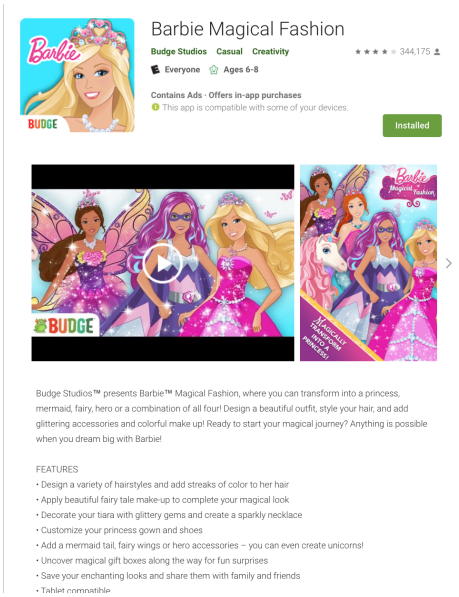
S9: Character Generator: Develop your story characters (formerly Auctor)

Creator: Rubendramirez
Url: `https://play.google.com/store/apps/details?id=com.plotgen.rramirez.plotgenerator&hl=en_US`
Year: Android
In chapter: Slow Creators
Description: A virtual notebook and assistant for describing characters for new stories.



S10: Barbie Magical Fashion

Creator: Budge Studios
Url: `https://play.google.com/store/apps/details?id=com.budgestudios.BarbieMagicalFashion`
Platform: Android
In chapters: A generative framework of generativity, Pattern - Transformation
Description: An outfit-editing app, operating not unlike old-fashioned paper dolls.



S11: The Best Art

Creator: Nicole He
Url: `http://nicole.pizza/the-best-art/`
Year: 2017
Platform: installation
In chapter: Pattern - No Blank Slate
Description: A mixed-initiative project in which a wall-mounted printer generates art assignments (via Tracery)for the artist, who then constructs something for that prompt and posts it.



S12: Betrayal at House on the Hill

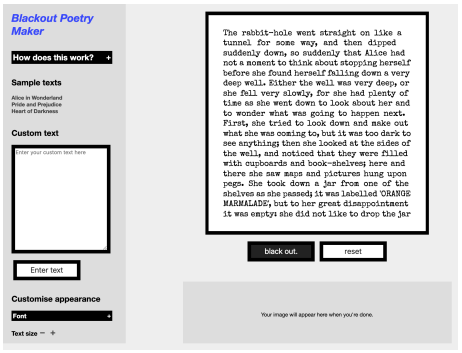
Creator: Avalon Hill (publisher), Bruce Glassco (designer)
Url: `https://boardgamegeek.com/boardgame/10547/betrayal-house-hill`
Year: 2004
Platform: boardgame
In chapters: Existing Frameworks of Generativity, A generative framework of generativity
Description: A boardgame where the tiles are laid out over the course of the game, revealing a new game map each time.

S13: Binary Fission



Creator: Kate Compton, Heather Logas, Joseph C. Osborn, Chandranil Chakrabortti, Kelsey Coffman, Daniel Fava, DylanLederle-Ensign, Zhongpeng Lin, Jo Mazeika, Afshin Mobramaein, Johnathan Pagnutti, Husacar Sanchez, Jim Whitehead, Brenda Laurel
Url: `http://www.galaxykate.com/apps/binaryfission/`
Year: 2016
Platform: web
In chapter: Expert evaluation
Description: A serious game for DARPA, this puzzle game guides user to build efficient decision trees that can sort loop-invariant data.

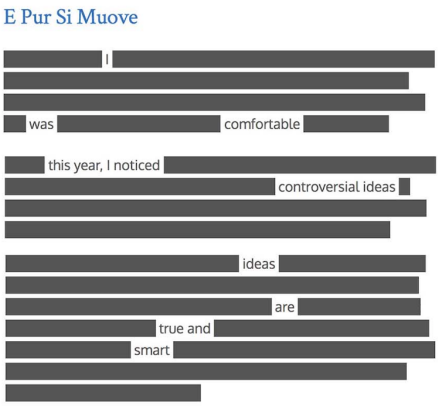
S14: Blackout Poetry Maker



Creator: Emma Winston
Url: `https://blackoutpoetry.glitch.me/`
Year: 2018
Platform: web
In chapter: Slow Creators
Description: A blackout poetry generator in which the user can redact text in a typographically-styled editor.

S15: blackout poetry generator

Creator: Max Kreminski
Url: `https://mkremins.github.io/blackout/`
Year: 2017
Platform: Chrome bookmarklet
In chapter: Slow Creators
Description: A mixed-initiative blackout poetry generator that finds subsets of words in the original text which form sentences.

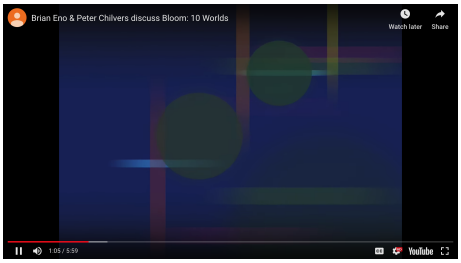


S16: Blade and Soul

Creator: NCSOFT
Url: `https://www.bladeandsoul.com/en/`
Year: 2012
Platform: online game
In chapter: Pattern - Expressing Identity
Description: An online game with a massive and complex character-and-face editor.

S17: Bloom: 10 Worlds

Creator: Brian Eno and Peter Chilvers
Url: `http://generativemusic.com/bloom10worlds.html`
Platform: IOS
In chapter: Slow Creators
Description: An abstract and minimalist creator in which the user's tablet gestures generate music and visual patterns.



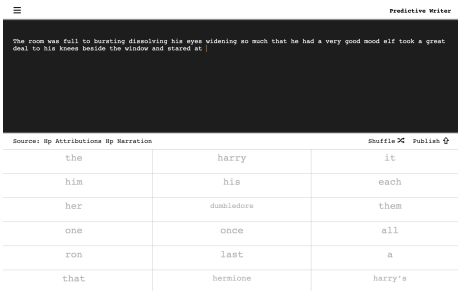
S18: Bluebeard's Bride

Creator: Magpie Games
Url: `https://www.magpiegames.com/bluebeards-bride/`
Year: 2018
Platform: tabletop game
In chapter: Slow Creators
Description: A tabletop game in which players take on the role of internal arguments in the main character's mind.



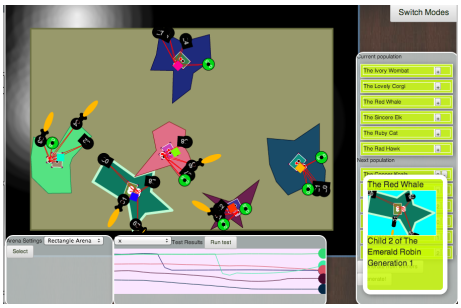
S19: Botnik Writer

Creator: Botnik Studios
Url: `https://botnik.org/apps/writer/`
Platform: web
In chapter: Pattern - Entertaining Evaluations
Description: A mixed initiative tool in which users can autocomplete sentences (like a phone's predictive keyboard), but trained on corpuses like Harry Potter and recipe books. Often misreported as 'a robot wrote Harry Potter'.

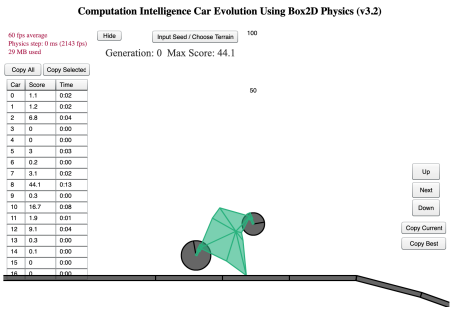


S20: BotPrint

Creator: Kate Compton, Jo Mazeika, Johnathan Pagnutti, Dylan Lederle-Ensign, Afshin Mobramaein, Jim Whitehead
Year: 2014
Platform: research prototype
In chapters:
Description: A research project to make a casual creator for designing a laser-cuttable robot chassis.

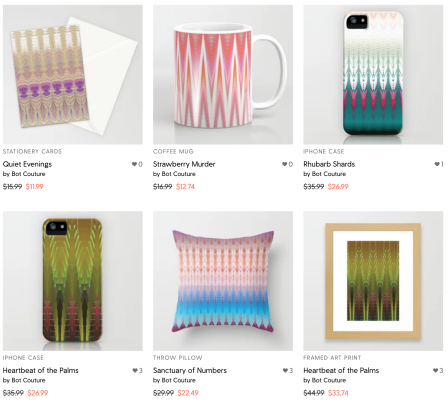


S21: BoxCar2D



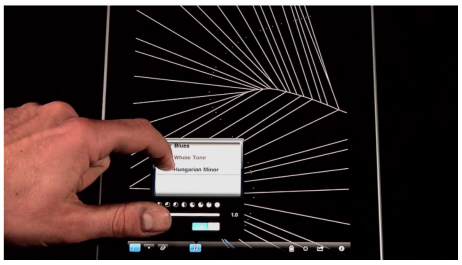
Creator: Ryan Weber
Url: `http://boxcar2d.com/index.html`
Year: 2011
Platform: web
In chapter: Parametric Spaces
Description: Players can watch a generative geometic car evolve to roll across a difficult landscape (note: developed by a UCSC grad?!)

S22: Broderie



Creator: Kate Compton
Url: `http://www.galaxykate.com/apps/unpublic/acaelo/`
Platform: web
In chapter: Pattern - Make It Real
Description: A series of relate projects to generate patterns from particle systems. It has been used as a mixed initiative tool with evolutionary algorithms (in Vector Space), and printed as art prints and scarves (Make It Real), and projected animations.

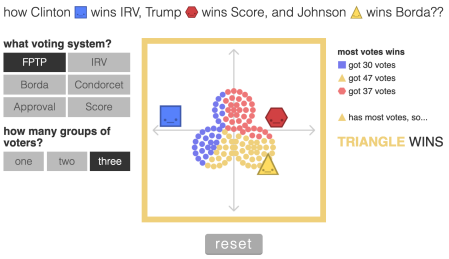
S23: Bubble Harp



Creator: Scott Snibbe
Url: `https://www.snibbe.com/bubbleharp-app`,
Year: 1997,2010
Platform: iPad
In chapters: Narrow and Shallow Possibility Spaces, Slow Creators
Description: An art experience where the user's gesture is interpreted into a Voronoi graph, which serves as an instrument.

S24: To Build a Better Ballot

Creator: Nicky Case
Url: `https://ncase.me/ballot/`
In chapter: Motivation
Description: An explorable explanation to test out different voting schemes.



S25: Cards Against Humanity

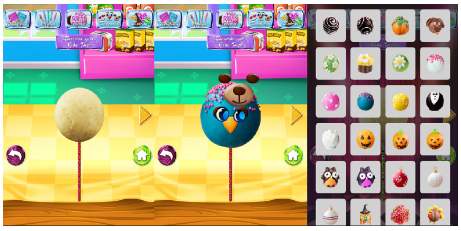
Creator: Ben Hantoot, Max Temkin
Url: `https://cardsagainsthumanity.com/`
Year: 2011
Platform: card game
In chapters: Creativity, Pattern - Entertaining Evaluations
Description: Similar to Apples-to-Apples, but CAH uses maximally offensive signifiers.

S26: Real Cake Maker 3D - Bake, Design & Decorate

Creator: Coco Play By TabTale
Url: `https://play.google.com/store/apps/details?id=com.cocoplay.cakemaker`
Platform: Android
In chapters: Narrow and Shallow Possibility Spaces, Pattern - Signifiers Add Power
Description: One of many child-targeted editors for baked goods.

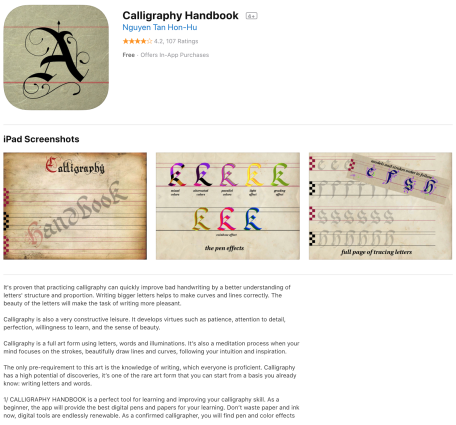


S27: Cake Pop Maker - Cooking Games



Creator: Bean Sprites
Url: `https://play.google.com/store/apps/details?id=com.beansprites.cakepopscooking`
Platform: Android
In chapters: Narrow and Shallow Possibility Spaces, Pattern - Signifiers Add Power
Description: One of many similar apps marketed toward children. Sequences of tasks contain rote tasks (open the oven) and somewhat creative tasks (place the candies), alternating with watching ads.

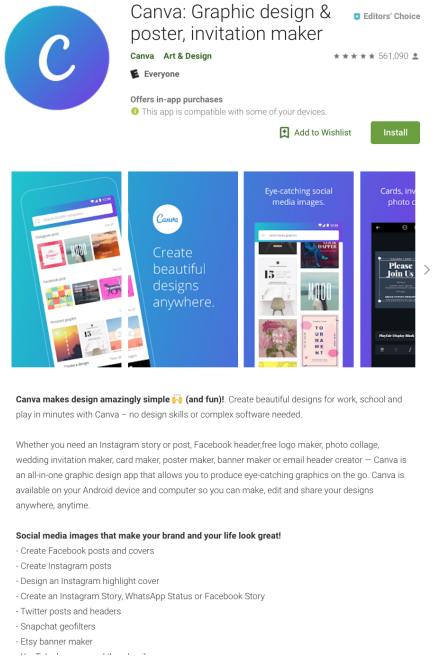
S28: Calligraphy Handbook



Creator: Nguyen Tan Hon-Hu
Url: `https://itunes.apple.com/us/app/calligraphy-handbook/id830584075?mt=8`
Platform: IOS
In chapter: Drawing Tools
Description: An IOS app to trace calligraphic tutorials, as well as test out calligraphy patterns and colors.

S29: Canva: Graphic Design & Logo, Flyer, Poster maker

Creator: Canva
Url: `https://play.google.com/store/apps/details?id=com.canva.editor`
Platform: Android
In chapter: Central Themes
Description: A casual creator for making simple graphic design layouts.



S30: Caves of Qud

Creator: Freehold Games
Url: `http://www.cavesofqud.com/`
Year: 2010
Platform: PC
In chapter: Possibility Spaces
Description: A disaster-simulator rendered in ASCII, the complex systems in this game create strange situations that players delight in retelling to each other.



S31: Cheap Bots, Done Quick!



Creator: George Buckenham (v21)
Url: `https://cheapbotsdonequick.com/`
Platform: web
In chapter: Tracery and Chancery
Description: A site to host users' Tracery grammars and Twitter credentials, which are used to regularly post generated output, for easy-to-make Twitterbots. Hosts six thousand (or so) active bots as of spring 2018.

S32: Cheap Markovs, Traced Quick!

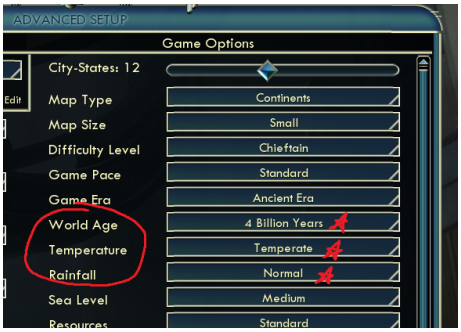
Creator: Serin Delaunay
Url: `https://github.com/serin-delaunay/cheapmarkovstracedquick`
Platform: python module
In chapter: Casual Programming
Description: A Python tool to generate Tracery grammars from plain text corpora.

S33: Chibify



Creator: Chibify
Url: `https://www.chibify.com/`
Platform: web + 3D printing
In chapter: Pattern - Make It Real
Description: An online site to make simple doll-like figures, which can be 3D printed (Make It Real)

S34: Civilization V (map generator)



Creator: Firaxis Games
Url: `https://gaming.stackexchange.com/questions/19389/civilization-v-map-generator`
Year: 2010
Platform: OSX
In chapter: A generative framework of generativity
Description: A generator for playable game levels that has many complex settings for input.

S35: Coffee: A Misunderstanding



Creator: Dietrich Squinkifer
Url: `http://squinky.me/2014/06/23/coffee-a-misunderstanding/`
Year: 2014
Platform: live performance with iPod Touches
In chapter: Pattern - Safety
Description: An interactive theater game in which two actors are fed lines by two puppeteers. This piece skillfully diffuses the awkwardness of using audience members in uncomfortable dramatic situations.

S36: Cookie Clicker



Creator: Orteil
Url: `http://orteil.dashnet.org/cookieclicker/`
Platform: web
In chapter: Narrow and Shallow Possibility Spaces
Description: An idle game that begins with a simple task (click a cookie) and unfolds into a strange narrative about greed.

S37: Coyote Now



Creator: Ryan Feddersen
Url: `http://goodtimes.sc/santa-cruz-arts-entertainment/santa-cruz-mah-coyote-now/`
Year: 2018
Platform: installation
In chapter: Drawing Tools
Description: A room-sized immersive coloring book that can be colored by museum visitors.

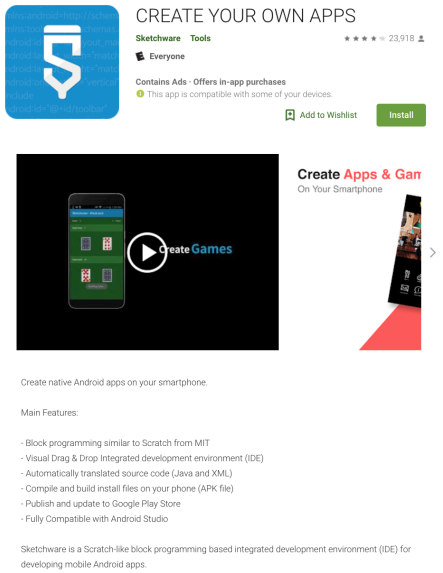
S38: Cranium



Creator: Hasbro
Url: `https://shop.hasbro.com/en-us/product/cranium-game:84DD09AD-5056-9047-F51B-702BA83401F7`
Year: 1998
Platform: board game
In chapter: Pattern - Safety
Description: A 90s board game with many intentionally impossible creative tasks (like drawing with one's eyes closed).

S39: Creator Your Own Apps

Creator: Sketchware
Url: `https://play.google.com/store/apps/details?id=com.besome.sketch`
Platform: Android
In chapter: Central Themes
Description: A Scratch-using game editor, but targeted at wannabe professional app-makers (including built-in ad serving and monetization).

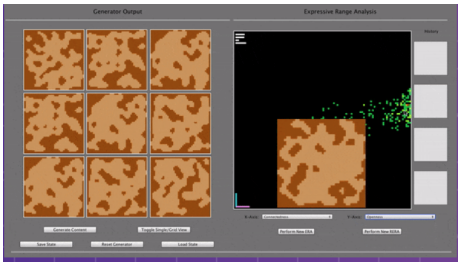


S40: Create Your Own Unicorn

Creator: Semisphere Apps
Url: `https://play.google.com/store/apps/details?id=com.Create.Your.Own.Unicorn.Dr`
Platform: Android
In chapter: A generative framework of generativity
Description: Select from several choice in horns, wings, and mane to create a trippy 3D unicorn.

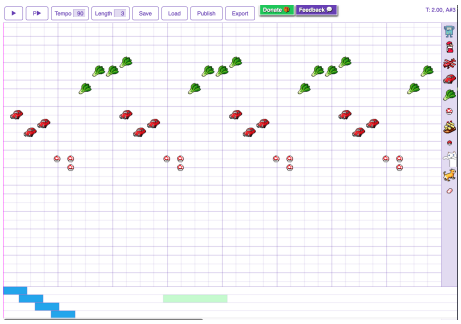


S41: Danesh



Creator: Michael Cook
Url: `http://www.danesh.procjam.com/`
Platform: A tool to visualize and navigate the possiblity space of a generator. (Cook does not consider this a casual creator)
In chapters: Possibility Spaces, Parametric Spaces, Pattern - Possibility Space Annotation
Description: 'Danesh is a tool to help people explore, explain and experiment with procedural generators'

S42: Mario Paint Music Composer



Creator: DanielX
Url: `https://danielx.net/composer/`
In chapter: Slow Creators
Description: A reimplementation of Mario Paint's music maker, with some new generative features not in the original.

S43: Digital Clockwork Muse

Creator: Rob Saunders
Url: `https://cs.gmu.edu/~jgero/publications/2001/01SaundersGeroAISB.pdf`
Year: 2001
Platform: unreleased software
In chapters: Creativity, Pattern - Entertaining Evaluations
Description: An academic project to simulate a community of artists and their evolving opinions about each other's work. A major influence on Techne.

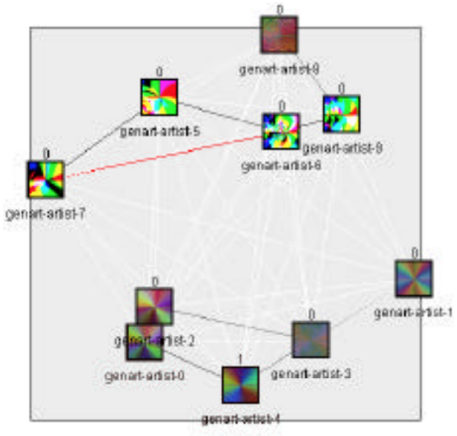
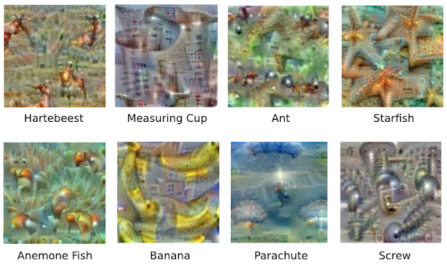


Figure 10: A screenshot of the simulation clearly showing the two cliques. The squares represent agents. The images show the currently selected genetic artwork for each agent. The number above each square shows the agent's attributed creativity. The dark lines between agents indicate the communication of credit.

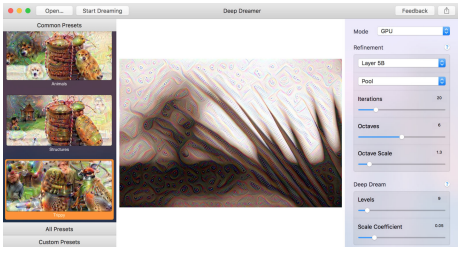
S44: Deepdreem

Creator: Google, Alexander Mordvintsev. Michael Tyka, Christopher Olah
Url: `https://github.com/google/deepdream`
Year: 2015
Platform: IPython Notebook
In chapter: Pattern - Transformation
Description: An algorithmic approach to generative images with generative adversarial networks. Kicked off a revolution in AI-created art.



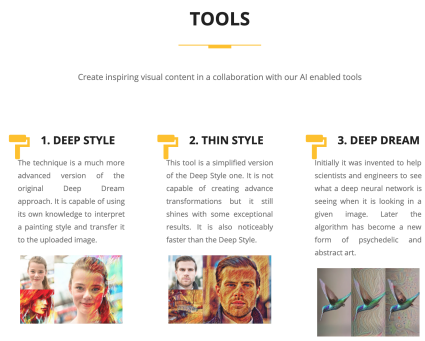
S45: Deep Dreamer

Url: `https://www.71squared.com/deepdreamer`
Platform: OSX application
In chapter: Pattern - Transformation
Description: A desktop tool for making Deep Dream artworks with user-provided images.



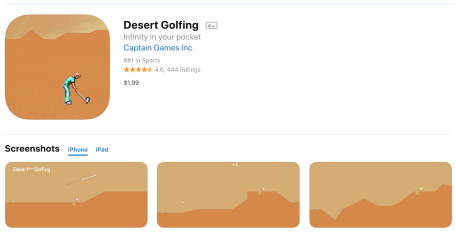
S46: Deep Dream Generator

Url: `https://deepdreamgenerator.com/`
Platform: web
In chapter: Pattern - Transformation
Description: A web tool for making Deep Dream artworks with user-provided images.



S47: Desert Golfing

Creator: Justin Smith, Captain Games
Url: `https://itunes.apple.com/us/app/desert-golfing/id902062673?mt=8`
Year: 2017
Platform: IOS
In chapter: Possibility Spaces
Description: A game of impossibly-many generative golf levels. Became a focus of collaborative internet effort to catalog and beat the levels.



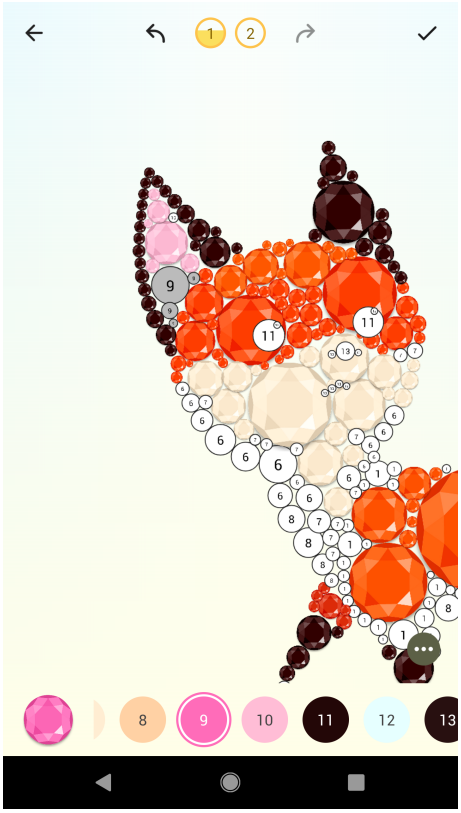
S48: Diablo II

Creator: Blizzard North
Url: `http://us.blizzard.com/en-us/games/d2/`
Year: 2000
Platform: PC
In chapters: A generative framework of generativity, Possibility Spaces, Pattern - Safety
Description: A very popular game with simple tile-based generative dungeons.



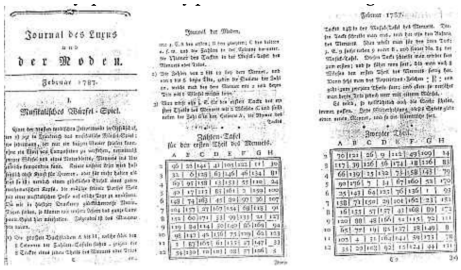
S49: Diamond art - Diamond coloring and Easy Relaxing

Creator: incolor
Url: `https://play.google.com/store/apps/details?id=com.diamond.art.color.by.number`
Platform: Android
In chapter: Narrow and Shallow Possibility Spaces
Description: An app to place virtual rhinestones to complete a picture.



S50: Dice music (Musikalisches Wurfelspiel)

Year: 1757
Platform: sheet music
In chapter: Slow Creators
Description: A genre of generative combinatorial sheet music in the 1700s.



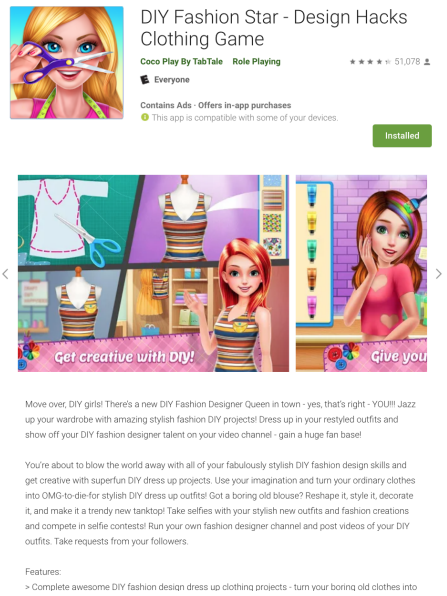
S51: Live Texturing of Augmented Reality Characters from Colored Drawings

Creator: Disney Research
Url: `https://www.youtube.com/watch?v=SWzurBQ81CM`
Year: 2015
Platform: research project on iPad
In chapter: Pattern - Transformation
Description: A research project to texture and color 3D models from scans of children's coloring-book pages.



S52: DIY Fashion Star - Design Hacks Clothing Game

Creator: Coco Play by TabTale
Url: `https://play.google.com/store/apps/details?id=com.cocoplay.diy.fashion`
Platform: Android
In chapters: A generative framework of generativity, Pattern - Entertaining Evaluations
Description: A game about cutting up and reconfiguring clothing, with a virtual in-game Instagram where the outfits get rated with virtual likes

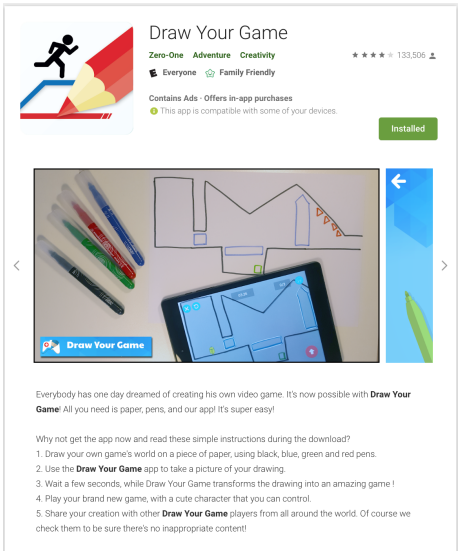


S53: Drawing Machine 1



Creator: Josph L Griffith
Url: `http://josephlgriffiths.com/drawingmachine1.html`
Year: 2009
Platform: installation
In chapter: Drawing Tools
Description: A bicycle attached to a wall-mounted compass with markers, so that pedalling the bicycle draws erratic circles on the wall.

S54: Draw Your Game

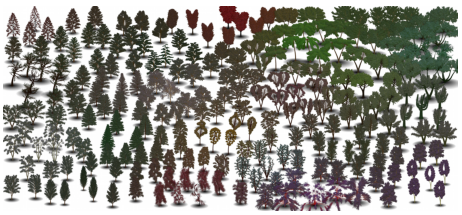


Creator: Zero-one
Url: `https://play.google.com/store/apps/details?id=com.korrisoft.draw.your.game`
Platform: Android
In chapters: Central Themes, Drawing Tools
Description: An app to let users scan new platformer levels from physically-drawn schematics. Pen colors indicate different kinds of surfaces.

S55: Dread

Creator: Epidiah Ravachol
Url: `https://dreadthegame.wordpress.com/about-dread-the-game/`
Platform: 2005
In chapter: Slow Creators
Description: A horror-themed game that uses Jenga as a dramatic tool. Its character creation has been extended with a casual creator by Ian Horswill.

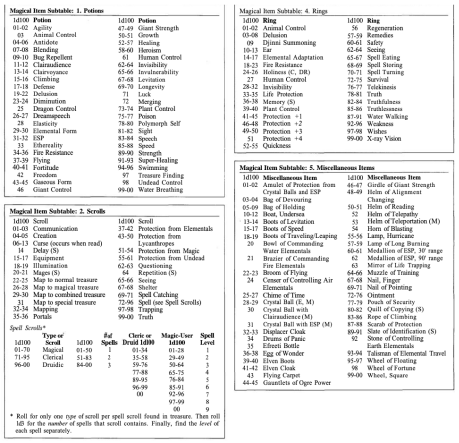
S56: Dryad



Creator: Stanford Virtual Worlds Group
Url: `https://news.stanford.edu/news/2008/january9/dryad-010908.html,https://web.archive.org/web/20080824061151/http://dryad.stanford.edu/`
Year: 2007
Platform: PC
In chapters: Possibility Spaces, Parametric Spaces, Pattern - Possibility Space Annotation
Description: A system to explore a generative space of L-system trees.

S57: Dungeons & Dragons

Creator: TSR
Year: 1978
Platform: table-top role playing
In chapter: A generative framework of generativity
Description: A storytelling game system with several generative components (Image from https://issuu.com/mfrances73/docs/dndrc/226



Magical Item Subtable: 1. Potions

1d100	Potion	1d100	Potion
01-02	Agility	47-49	Giant Strength
03	Animal Control	50-51	Growth
04-06	Antidote	52-57	Healing
07-08	Blending	58-60	Heroism
09-10	Bug Repellent	61	Human Control
11-12	Clairaudience	62-64	Invisibility
13-14	Clairvoyance	65-66	Invulnerability
15-16	Climbing	67-68	Levitation
17-18	Defense	69-70	Longevity
19-22	Delusion	71	Luck
23-24	Diminution	72	Merging
25	Dragon Control	73-74	Plant Control
26-27	Dreamspeech	75-77	Poison
28	Elasticity	78-80	Polymorph Self
29-30	Elemental Form	81-82	Sight
31-32	ESP	83-84	Speech
33	Ethereality	85-88	Speed
34-36	Fire Resistance	89-90	Strength
37-39	Flying	91-93	Super-Healing
40-41	Fortitude	94-96	Swimming
42	Freedom	97	Treasure Finding
43-45	Gaseous Form	98	Undead Control
46	Giant Control	99-00	Water Breathing

Magical Item Subtable: 2. Scrolls

1d100	Scroll	1d100	Scroll
01-03	Communication	37-42	Protection from Elementals
04-05	Creation	43-50	Protection from Lycanthropes
06-13	Curse (occurs when read)	51-54	Protection from Magic
14	Delay (S)	55-61	Protection from Undead
15-17	Equipment	62-63	Questioning
18-19	Illumination	64	Repetition (S)
20-21	Magic (S)	65-66	Seeing
22-25	Map to normal treasure	67-68	Shelter
26-28	Map to magical treasure	69-71	Spell Catching
29-30	Map to combined treasure	72-96	Spell (see Spell Scrolls)
31	Map to special treasure	97-98	Trapping
32-34	Mapping	99-00	Truth
35-36	Portals		

*Spell Scrolls:**

1d100	Type or/ Scroll	1d100	# of Spells	Cleric or Druid	Magic-User 1d100	Spell Level
01-70	Magical	01-50	1		01-34	1
71-95	Clerical	51-83	2		35-58	2
96-00	Druidic	84-00	3		59-76	3
					77-88	4
					89-95	5
					96-99	6
					00	7
						8
						9

* Roll for only one *type* of scroll per spell scroll found in treasure. Then roll 1d5 for the *number of* spells that scroll contains. Finally, find the *level* of each spell separately.

Magical Item Subtable: 4. Rings

1d100	Ring	1d100	Ring
01-02	Animal Control	56	Regeneration
03-08	Delusion	57-59	Remedies
09	Djinni Summoning	60-61	Safety
10-13	Ear	62-64	Seeing
14-17	Elemental Adaptation	65-67	Spell Eating
18-23	Fire Resistance	68-69	Spell Storing
24-26	Holiness (C, DR)	70-71	Spell Turning
27	Human Control	72-75	Survival
28-32	Invisibility	76-77	Telekinesis
33-35	Life Protection	78-81	Truth
36-38	Memory (S)	82-84	Truthfulness
39-40	Plant Control	85-86	Truthlessness
41-45	Protection +1	87-91	Water Walking
46-48	Protection +2	92-96	Weakness
49-50	Protection +3	97-98	Wishes
51	Protection +4	99-00	X-ray Vision
52-55	Quickness		

Magical Item Subtable: 5. Miscellaneous Items

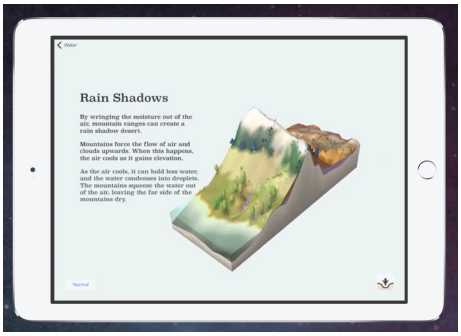
1d100	Miscellaneous Item	1d100	Miscellaneous Item
01-02	Amulet of Protection from Crystal Balls and ESP	46-47	Girdle of Giant Strength
03-04	Bag of Devouring	48-49	Helm of Alignment Changing
05-09	Bag of Holding	50-51	Helm of Reading
10-12	Boat, Undersea	52	Helm of Telepathy
13-14	Boots of Levitation	53	Helm of Teleportation (M)
15-17	Boots of Speed	54	Horn of Blasting
18-19	Boots of Traveling/Leaping	55-56	Lamp, Hurricane
20	Bowl of Commanding Water Elementals	57-59	Lamp of Long Burning
21	Brazier of Commanding Fire Elementals	60-61	Medallion of ESP, 30' range
22-23	Broom of Flying	62	Medallion of ESP, 90' range
24	Censer of Controlling Air Elementals	63	Mirror of Life Trapping
25-27	Chime of Time	64-66	Muzzle of Training
28-29	Crystal Ball (E, M)	67-68	Nail, Finger
30	Crystal Ball with Clairaudience (M)	69-71	Nail of Pointing
31	Crystal Ball with ESP (M)	72-76	Ointment
32-33	Displacer Cloak	77-79	Pouch of Security
34	Drums of Panic	80-82	Quill of Copying (S)
35	Efreeti Bottle	83-86	Rope of Climbing
36-38	Egg of Wonder	87-88	Scarab of Protection
39-40	Elven Boots	89-91	Slate of Identification (S)
41-42	Elven Cloak	92	Stone of Controlling Earth Elementals
43	Flying Carpet	93-94	Talisman of Elemental Travel
44-45	Gauntlets of Ogre Power	95-97	Wheel of Floating
		98	Wheel of Fortune
		99-00	Wheel, Square

S58: Dwarf Fortress

Creator: Bay 12 Games
Url: `http://www.bay12games.com/dwarves/`
Year: 2006
Platform: PC
In chapters: A generative framework of generativity, Possibility Spaces, Pattern - Self-promotion
Description: An immensely complex simulation of thousands of years of history. Spawned story telling commmunities and tools like the Legends Viewer.

S59: Earth Primer

Creator: Chaim Gingold
Url: `http://www.earthprimer.com/`
Year: 2015
Platform: iPad
In chapter: Motivation
Description: A combination of virtual sandbox and textbook to let users experiment with how meteorology and geology work.



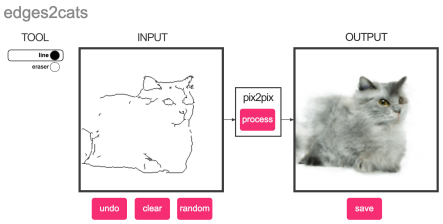
S60: Eastshade

Creator: Eastshade Studios
Year: 2019
Platform: PC
In chapter: Introduction
Description: In this lavishly decorative game about fulfilling quests, the player can paint pictures of the scenery, which can be sold or gifted to the NPCs.

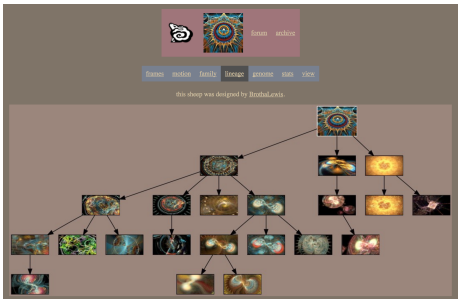


S61: Image-to-Image Demo (Edges2Cats)

Creator: Christopher Hesse
Url: `https://affinelayer.com/pixsrv/`
Year: 2017
Platform: web with machine learning
In chapters: Drawing Tools, Pattern - Transformation
Description: A clever use of GANs to fill user-drawn outlines with an approximation of cats.



S62: Electric Sheep



Creator: Scott Draves
Url: `https://electricsheep.org/`
Year: 1999
Platform: desktop screensaver
In chapter: Parametric Spaces
Description: A long-running project where community members render, vote on, and create generative screensaves.

S63: Elite Dangerous (database)



Creator: Frontier Developments plc
Url: `https://eddb.io/`
Year: 2014
Platform: Steam game, with online database
In chapter: Pattern - Possibility Space Annotation
Description: A space trading game notable in this dissertation for its community-driven documentation

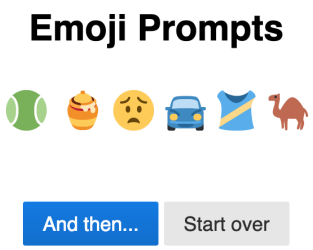
S64: ELIZA



Creator: Joseph Weizenbaum
Url: `https://www.masswerk.at/elizabot/`
Year: 1966
Platform: IBM 7094, MAD-SLIP.
In chapter: A generative framework of generativity
Description: The famous template-based chatbot and virtual therapist.

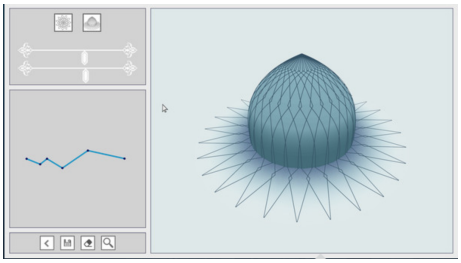
S65: Emoji Prompts

Creator: Byrdseed
Url: `http://byrdseed.com/emoji/`
Platform: web
In chapter: Pattern - Signifiers Add Power
Description: A simple generator of emoji sequences, for use as writing prompts



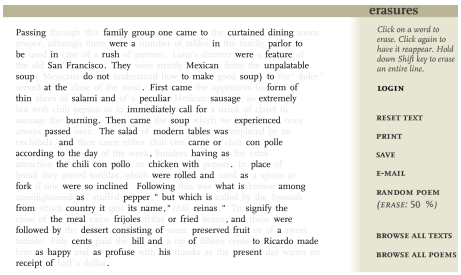
S66: Engare

Creator: Mahdi Bahrami
Url: `http://www.engare.design/`
Year: 2017
Platform: desktop
In chapter: Pattern - Repetition, Repetition, Repetition
Description: A game and casual creator for geometrically-generated patterns.



S67: ERASURES

Creator: Joshua Beckman and David Hirmes
Url: `https://erasures.wavepoetry.com/`
Platform: web
In chapter: Slow Creators
Description: A web-based blackout poetry tool



S68: Ethereal Dialpad

Creator: Adam Smith
Url: `(Nolongeravailable)`
Year: 2010
Platform: Android
In chapter: Slow Creators
Description: A touch-based tool to generate harmonious sound doodlings with the swipe of a finger.

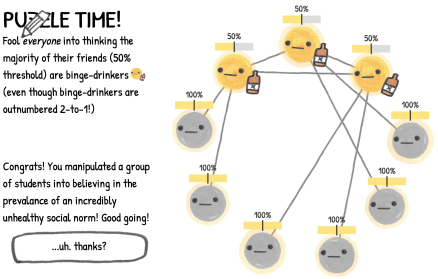
S69: Every Rat in NYC (That People Complained About in 2016)

Creator: Lynn Cherny
Url: `http://bl.ocks.org/arnicas/raw/d210166c6d302da09397/`
Year: 2016
Platform: web
In chapter: Tracery and Chancery
Description: A project combining Flickr, Google Maps, Tracery and the NYC public health rat-sightings data feed, to make up fictional lives for rats seen in NYC in 2016.



S70: The Evolution of Trust

Creator: Nicky Case
Url: `https://ncase.me/trust/`
Platform: web
In chapter: Motivation
Description: An explorable explanation of how different betting strategies fare against each other.

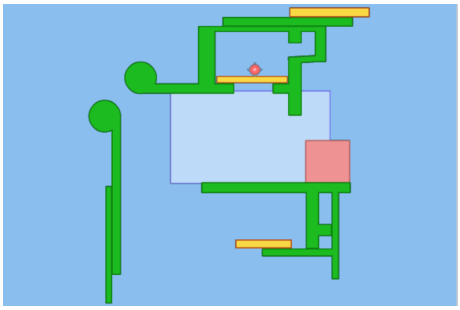


S71: Exonumia

Creator: Kate Compton
Year: 2012
Platform: web
In chapter: Pattern - Make It Real
Description: An unfinished casual creator for making 3D printed exonumia (commemorative coins and medals).



S72: Fantastic Contraption



Creator: Northway Games, Colin Northway
Url: `https://www.kongregate.com/games/inxile_ent/fantastic-contraption`
Year: 2008
Platform: Flash
In chapter: Pattern - Self-promotion
Description: A game about building simulated 2D mechanical machines. Had a very robust solution-sharing framework.

S73: Fiasco

Creator: Jason Morningstar, Bully Pulpit Games
Url: `https://bullypulpitgames.com/games/fiasco/`
Year: 2009
In chapters: Creativity Support Frameworks, Slow Creators
Description: A storytelling game with numerous randomly-generated constraints to provide structure and inspiration.

S74: Firewatch

Creator: Campo Santo Games
Year: 2016
Platform: PC
In chapter: Introduction
Description: In this meditative 3D story game, players can take photos with an in-game disposable camera, which can be sent to Campo Santo to be developed and sent back as vintage-style physical prints, in an example of the Make-It-Real pattern.

S75: Microsoft FreeCell

Creator: Microsoft, Jim Horne
Year: 1995
Platform: Windows 95
In chapters: Possibility Spaces, Pattern - Possibility Space Annotation
Description: A game that shipped with many versions of Windows, and had a community dedicated to cataloging all possible deals.

S76: Funky Ikebana

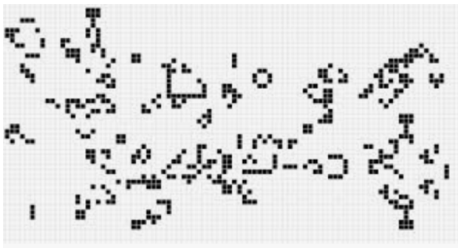


Creator: Kate Compton
Url: `http://www.galaxykate.com/apps/flowers/`
Year: 2014
Platform: web,processing.js
In chapters: A generative framework of generativity, Parametric Spaces
Description: An ambient art toy for exploring a space of generative L-System flowers.

S77: GameMaker Studio

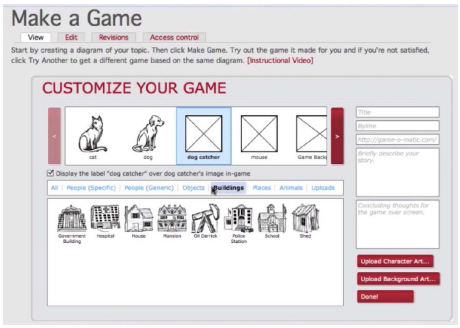
Creator: Yoyo Games
Url: `https://www.yoyogames.com/gamemaker`
Year: 1999
Platform: desktop app
In chapter: Expert evaluation
Description: A more-accessible-than-Unity software tool for making games.

S78: Conway's Game of Life



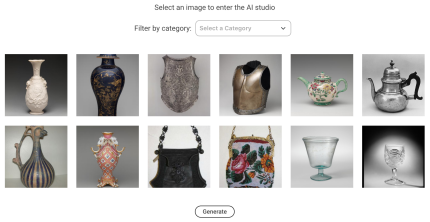
Creator: John Conway
Year: 1970
Platform: mathematic rule set
In chapter: A generative framework of generativity
Description: A set of rules that, when applied repeatedly to a set of black and white tiles, cause interestingly emergent behavior.

S79: Game-O-Matic



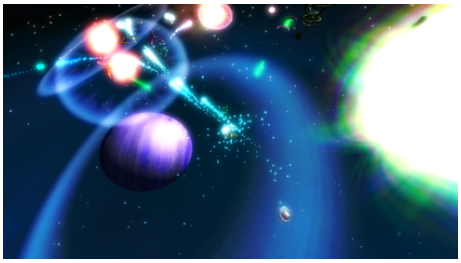
Creator: Mike Treanor, Brian Blackford, Michael Mateas, Ian Bogost
Url: `https://games.soe.ucsc.edu/game-o-matic-generating-videogames-represent-ideas`
Year: 2012
Platform: web (flash)
In chapter: Slow Creators
Description: A casual creator (with a big logic/AI backend) for making editorial games.

S80: Ganbreeder



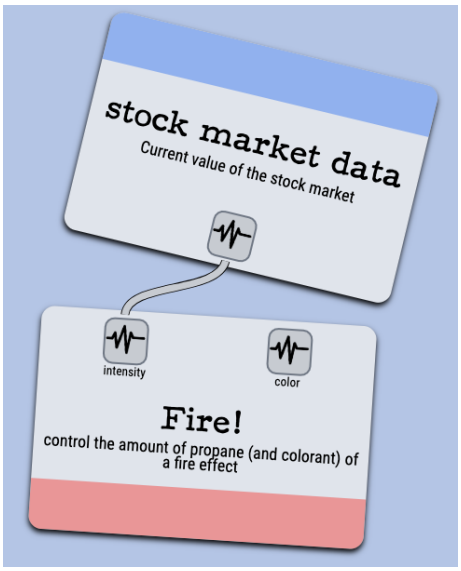
Creator: Joel Simon
Url: `https://ganbreeder.app/`
Year: 2019
Platform: web,tensorflow
In chapters: Parametric Spaces, Expert evaluation
Description: A site where anyone can try to explore the generative art created by combining different concepts (like dog and burrito). Has spawned a lot of delighted and horrified posts as people recount what they find.

S81: Galactic Arms Race



Creator: Evolutionary Games, Evolutionary Complexity Research Group at UCF
Url: `https://galacticarmsrace.blogspot.com/`,
Year: 2010
Platform: Windows
In chapters: A generative framework of generativity, Narrow and Shallow Possibility Spaces
Description: A shmup MMO where all the particle weapons are generated from a system that tracks their popularity.

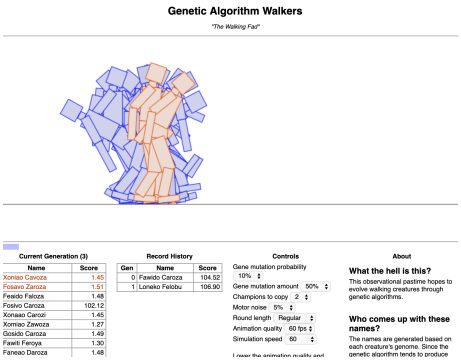
S82: Generominos



Creator: Kate Compton
Url: `http://www.galaxykate.com/generominos/`,`http://www.galaxykate.com/generominos/editor-dev/`
Year: 2018
Platform: card game
In chapters: Creativity Support Frameworks, Expert evaluation
Description: A set of Kickstarted cards to demonstrate some principles of generativity and generate new pipeline ideas.

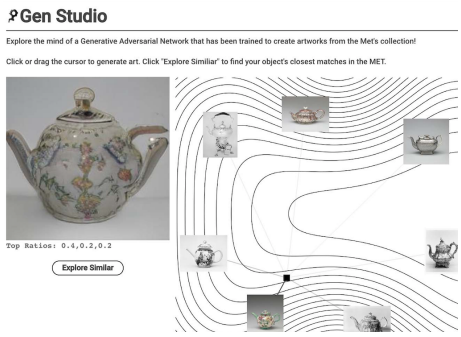
S83: Genetic Algorithm Walkers

Creator: Rafael Matsunaga
Url: `https://rednuht.org/genetic_walkers/`
Year: 2015
Platform: web
In chapter: Parametric Spaces
Description: A web app in which users can watch a genetic algorithm learning to pilot a walking humanoid figure.



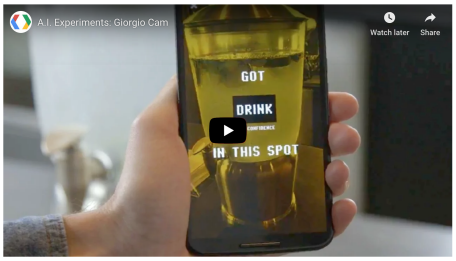
S84: Gen Studio

Creator: The Met, Microsoft, and MIT
Url: `https://gen.studio/`
Year: 2018
Platform: web, generative adversarial networks
In chapter: Parametric Spaces
Description: A tool to explore generative adversarial network-created variants on antiques from the Met Museum's collection.



S85: Giorgio Cam

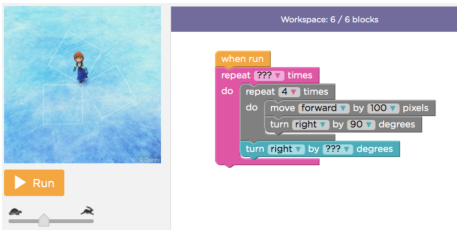
Creator: Eric Rosenbaum, Yotam Mann, Google Creative Lab
Url: `https://experiments.withgoogle.com/giorgio-cam`
Year: 2017
Platform: web, Google Cloud Vision API
In chapters: A generative framework of generativity, Slow Creators
Description: A mixed initiative tool that turns objects on a webcam into bad rap lyrics.



S86: Goat Simulator

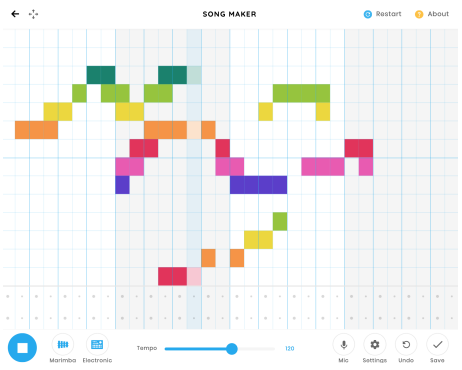
Creator: Coffee Stain Studios
Url: `http://www.goat-simulator.com/`
Year: 2014
Platform: Windows
In chapter: Motivation
Description: A bizarre game about playing a goat in a physics simulation, beloved by streamers for its emergent moments.

S87: Code with Anna and Elsa



Creator: Google
Url: `https://studio.code.org/s/frozen/stage/1/puzzle/1`
Year: 2014
Platform: web, Scratch
In chapter: Casual Programming
Description: An Hour of Code exercise for making Logo-style geometric patterns with virtual figure skaters and Scratch instructions.

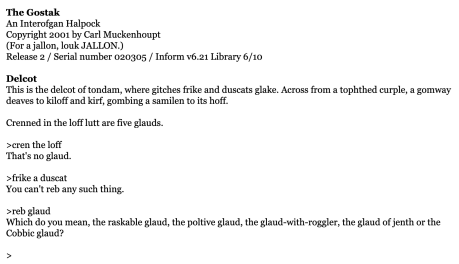
S88: Chrome Music Lab: Song-Maker



Creator: Google Creative Lab, Use All Five, Yotam Mann.
Url: `https://musiclab.chromeexperiments.com/Song-Maker`
Year: 2018
Platform: web
In chapters: Motivation, A generative framework of generativity, Slow Creators
Description: One of many grid-based music generation tools.

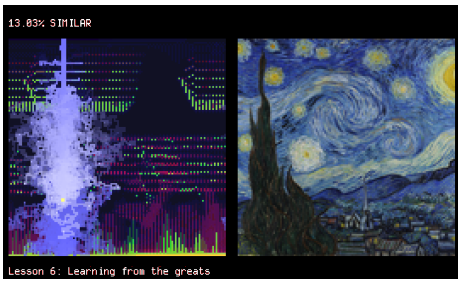
S89: The Gostak

Creator: Carl Muckenhoupt
Url: `https://ifdb.tads.org/viewgame?id=w5s3sv43s3p98v45`
Year: 2001
Platform: Z5
In chapter: Frameworks of Interactivity
Description: A text adventure in which the player must slowly translate a fictional language to be able to type commands.



S90: BECOME A GREAT ARTIST IN JUST 10 SECONDS

Creator: Michael Brough, Andi McClure
Url: `http://www.igf.com/become-great-artist-just-10-seconds`, `http://ludumdare.com/compo/ludum-dare-27/?action=preview&uid=4987`
Year: 2013
Platform: Mac
In chapters: Frameworks of Interactivity, Narrow and Shallow Possibility Spaces, Pattern - Entertaining Evaluations
Description: An award-winning glitch art experiment where each key is mapped to a different impossible-to-control pixel manipulation.

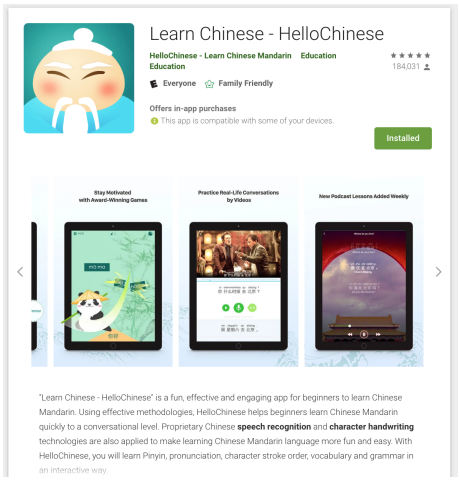


S91: GTA

Creator: Rockstar Games
Year: 2013
Platform: PC
In chapter: Introduction
Description: Primarily a game about driving fast and shooting people, recent games in this series (like GTA V) have developed a culture of personal expression and storytelling, using the limited creative tools provided by the game system.

S92: Learn Chinese - HelloChinese

Creator: HelloChinese - Learn Chinese Mandarin
Url: `https://play.google.com/store/apps/details?id=com.hellochinese`
Platform: Android
In chapter: Drawing Tools
Description: A language-learning app in which the user can trace characters for practice. Very little creative range.



S93: HipChef

Creator: Kate Compton
Year: 2015
Platform: HTML/Web
In chapters:
Description: An idle game about running a restaurant, with Tracery-generated recipes and patrons. Players curate a menu of generated content to fit their own preferences and the current hip trends.



S94: Human Free Photobooth

Creator: Kate Compton
Year: 2019
Platform: installation with machine learning
In chapter: Pattern - Entertaining Evaluations
Description: A AI-based photobooth that refuses to take pictures of humans, so the people in front of it must contort themselves until the algorithm can only label them as household objects.



S95: I Ching

Creator: unknown
Year: 9th century BCE
Platform: yarrow stalks
In chapters: Creativity Support Frameworks, Existing Frameworks of Generativity
Description: An ancient technique of interpreting randomly-generated phenomena as stories.

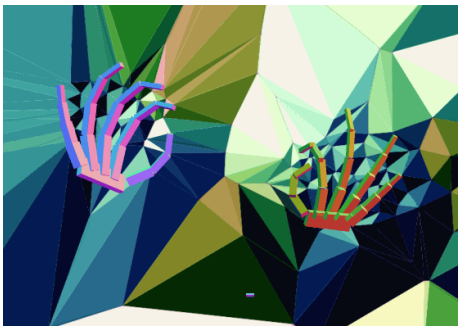
S96: Idle Game Maker

Creator: Orteil
Url: `http://orteil.dashnet.org/igm/`
Platform: web (hosted)
In chapter: Slow Creators
Description: An online editor for making simple idle games, by the maker of Cookie Clicker.



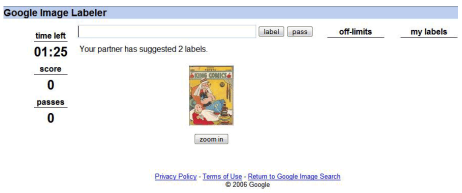
S97: Idle Hands

Creator: Kate Compton
Url: `http://www.galaxykate.com/apps/idlehands/`
Year: 2017
Platform: web,THREE.js,processing.js
In chapters: A generative framework of generativity, Expert evaluation
Description: An art toy in which the user's LeapMotion-tracked hands are continuously interpreted into vivid geometry.



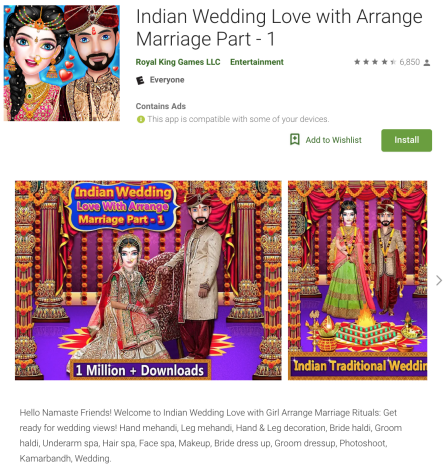
S98: Google Image Labeler

Creator: Google
Url: `https://images.google.com/imagelabeler/`
Year: 2006
Platform: web (multiplayer)
In chapter: Pattern - Entertaining Evaluations
Description: A two player game in which two strangers attempt to come up with matching labels for an image.



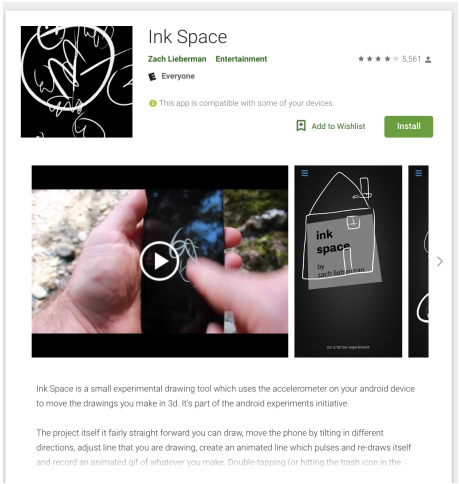
S99: Indian Wedding Love with Arrange Marriage Part - 1

Creator: Royal King Games LLC
Url: `https://www.amazon.com/Indian-Wedding-Love-Arrange-Marriage/dp/B07BDRNP59/`
Platform: Amazon app
In chapter: Pattern - Signifiers Add Power
Description: One of many apps that simulate all of the steps in preparing for a traditional Indian wedding, including a few creative choices.



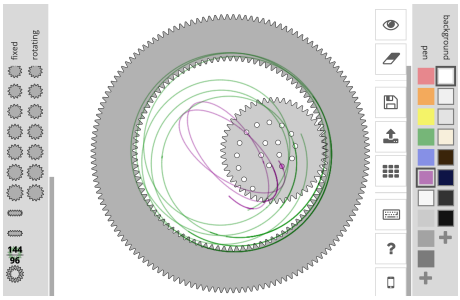
S100: Inkspace

Creator: Zach Lieberman
Url: `https://experiments.withgoogle.com/ink-space`
Year: 2015
In chapters:
Description: A 3D augmented-reality drawing program.



S101: Inspirograph

Creator: Nathan Friend
Url: `https://nathanfriend.io/inspirograph/`
Platform: web
In chapter: Possibility Spaces
Description: A web-based interactive simulator replicating the experience of using a physical Spirograph.



S102: @InstrumentBot

Creator: Cam Larios
Url: `https://twitter.com/instrumentbot`
Year: 2016
Platform: Twitterbot
In chapter: Pattern - No Blank Slate
Description: A Tracery-based Twitterbot designed by a sound artist to inspire new ideas.



S103: Interruption Junction

Creator: Dietrich Squinkifer
Url: `http://squinky.me/2015/01/18/interruption-junction/`
Year: 2015
Platform: web game with Tracery
In chapter: Tracery and Chancery
Description: A game where the user must mash the spacebar to socially participate in an inane Tracery-generated conversation.

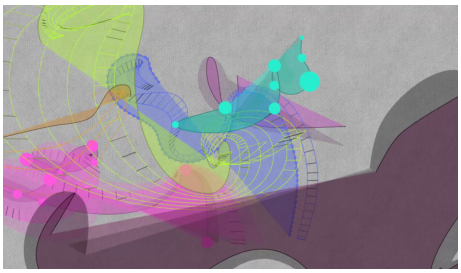


S104: I/O Brush



Creator: Kimiko Ryokai, Stefan Marti, Professor Hiroshi Ishii
Url: `https://tangible.media.mit.edu/project/io-brush/`
Year: 2004
Platform: installation
In chapters: A generative framework of generativity, Drawing Tools
Description: Users sample textures with a brush-shaped webcam, then paint them into generative brush strokes.

S105: Jazz Composition



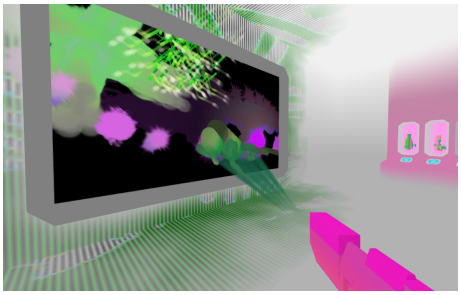
Creator: Kate Compton
Url: `http://www.galaxykate.com/apps/jazzcomposition/`
Year: 2018
Platform: web,processing.js
In chapters: A generative framework of generativity, Drawing Tools
Description: Gestures (from mouse, touch or LeapMotion) are drawn into complex generative lines which then oscillate on the screen.

S106: The Jellies Experience



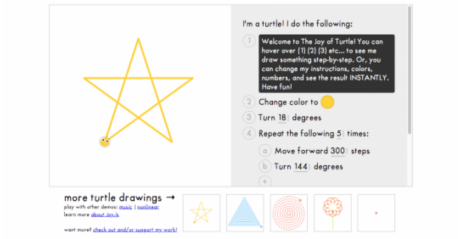
Creator: Monterey Bay Aquarium
Url: `https://www.youtube.com/watch?v=pE0hJWNuNas`
Year: 2012
Platform: installation
In chapter: Pattern - Transformation
Description: An aquarium installation in which children can color jellies on touchscreens, then watch them animate on a wall.

S107: Joy Exhibition



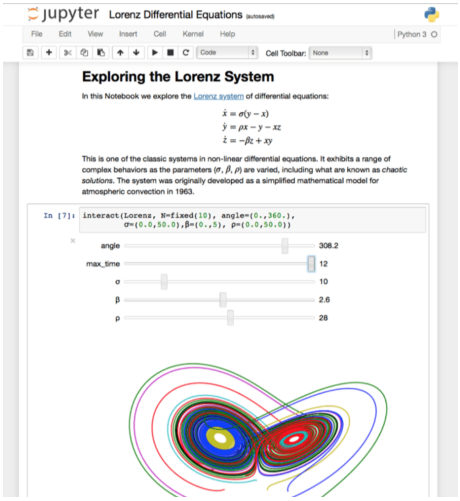
Url: `https://strangethink.itch.io/joy-exhibition`
Year: 2017
Platform: OSX
In chapters: Narrow and Shallow Possibility Spaces, Drawing Tools
Description: 'A downloadable UFO Art Zone', players pick up virtual guns and use them to spray unpredicable paint streams at a canvas.

S108: Joy.JS



Creator: Nicky Case
Url: `https://ncase.me/joy/`
Year: 2017
Platform: web
In chapter: Casual Programming
Description: A riff on Logo, this programming IDE allows the user to scroll variable values and see them continuously updated.

S109: Jupyter Notebook



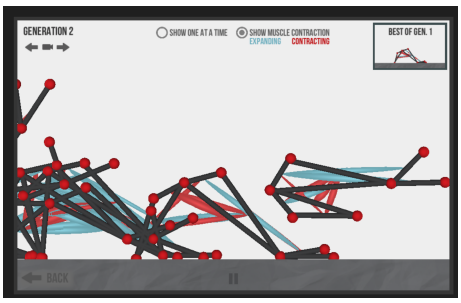
Creator: Project Jupyter
Url: `https://jupyter.org/`
Year: 2014
Platform: interactive Python platform
In chapter: Casual Programming
Description: A popular tool for combining interactive Python code and comments into one 'page'

S110: Kaleidoscope

Creator: Sir David Brewster
Year: 1817
Platform: physical toy
In chapters: Introduction, Pattern - Repetition, Repetition, Repetition, Expert evaluation
Description: From the patent:'The Kaleidoscope for exhibiting and creating beautiful Forms and Patterns of great use in all the ornamental Arts'.

S111: Evolution by Keiwan

Creator: Keiwan
Url: `https://keiwan.itch.io/evolution`
Platform: web
In chapter: Parametric Spaces
Description: An editor, simulator, and evolver for 2D tensegrity creatures.

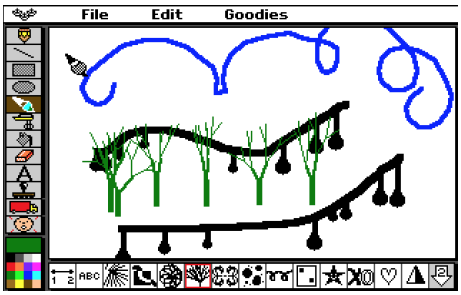


S112: KeyForge

Creator: Richard Garfield
Url: `https://www.fantasyflightgames.com/en/products/keyforge/`
Year: 2018
Platform: card game
In chapters: Possibility Spaces, Pattern - Possibility Space Annotation
Description: A collectible card game with (inadvisably) generative deck names.

S113: Kid Pix

Creator: Craig Hickman
Url: `https://archive.org/details/KID_PIX_DOS`
Year: 1989
Platform: Macintosh, Windows, emulator
In chapters: Introduction, Motivation, Narrow and Shallow Possibility Spaces, Drawing Tools, Pattern - Transformation
Description: A beloved piece of classic software for drawing surprising generative doodles.

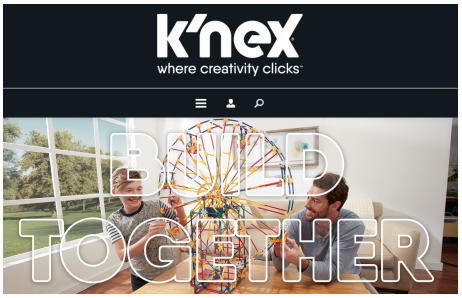


S114: Kirby: Canvas Curse



Creator: HAL Laboratory
Url: `https://kirby.fandom.com/wiki/Kirby:_Canvas_Curse`
Year: 2005
Platform: Nintendo DS
In chapter: Drawing Tools
Description: A game in which the user draws paths (with a stylus) for the animated character.

S115: k'nex



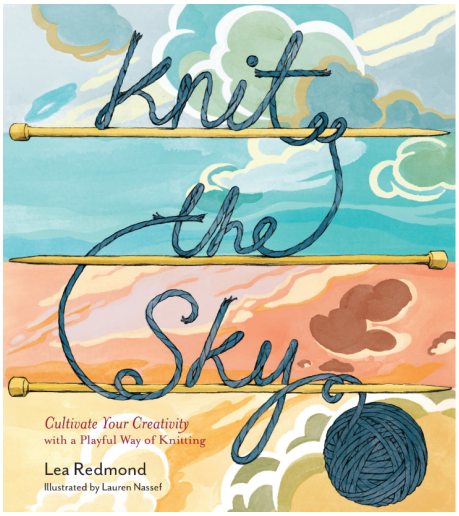
Creator: Joel Glickman
Url: `https://www.knex.com/`
Year: 1992
Platform: physical toy
In chapter: Pattern - Signifiers Add Power
Description: A set of snappable joints, struts, and flexors which can be combined to create strong and flexible structures.

S116: Knit Magic

Creator: Mattel
Year: 1974
Platform: physical toy
In chapter: Motivation
Description: A mechanized version of the Knitting Nancy tool, where turning a crank created a knitted tube.

S117: Knit the Sky: Cultivate Your Creativity with a Playful Way of Knitting

Creator: Lea Redmond
Url: `https://www.leafcutterdesigns.com/knitthesky/`
Year: 2015
Platform: book of knitting exercises
In chapter: Pattern - Make It Real
Description: A book of experiential knitting instructions, including one in which the user knits one row (matching the sky color) each day.



S118: Lasers & Feelings

Creator: One Seven Design (John Harper)
Url: `http://www.onesevendesign.com/laserfeelings/`
Year: 2013
Platform: one-page paper roleplaying instructions
In chapter: Slow Creators
Description: A spoof on Star Trek fanfic where all problems are solved by lasers or feelings, based on a dice roll.

GM: CREATE A SPACE ADVENTURE

Roll or choose on the tables below.

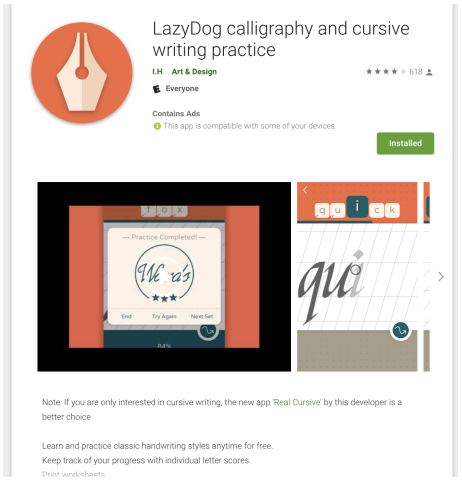
A THREAT...	
1. Zorgon the Conqueror	4. Space Pirates
2. The Hive Armada	5. Cyber Zombies
3. Rogue Captain	6. Alien Brain Worms
WANTS TO...	
1. Destroy / Corrupt	4. Protect / Empower
2. Steal / Capture	5. Build / Synthesize
3. Bond with	6. Pacify / Occupy

S119: The Laughing Room



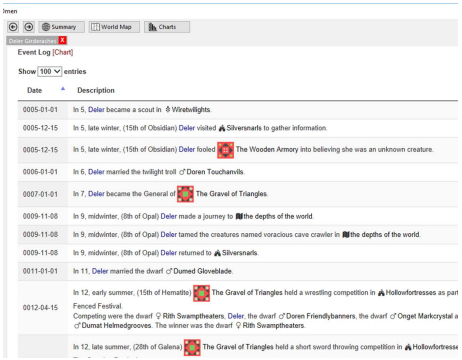
Creator: Jonny Sun,Hannah Davis, Christopher Sun,Nikhil Dharmaraj
Url: `https://metalabharvard.github.io/projects/laughingroom/`
Year: 2018
Platform: installation
In chapter: Pattern - Entertaining Evaluations
Description: Humans sitting in living room setting try to tell jokes. If the AI hears them and matches them to known humor, it produces a laugh track.

S120: LazyDog calligraphy and cursive writing practice



Creator: I.H
Url: `https://play.google.com/store/apps/details?id=info.bppacomics.lazydog`
Platform: Android
In chapter: Drawing Tools
Description: A calligraphy practice app where the user is judged on accuracy. No creativity/freeplay mode.

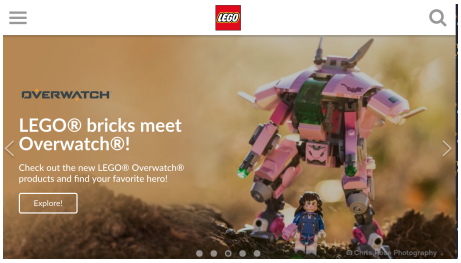
S121: Legends Viewer for Dwarf Fortress



Creator: Parker147
Url: `http://www.bay12forums.com/smf/index.php?topic=72702.0`
Year: 2011
Platform: desktop view for game-data
In chapter: Pattern - Self-promotion
Description: A fan-made tool to decode and browse the historical events of a Dwarf Fortress playthrough.

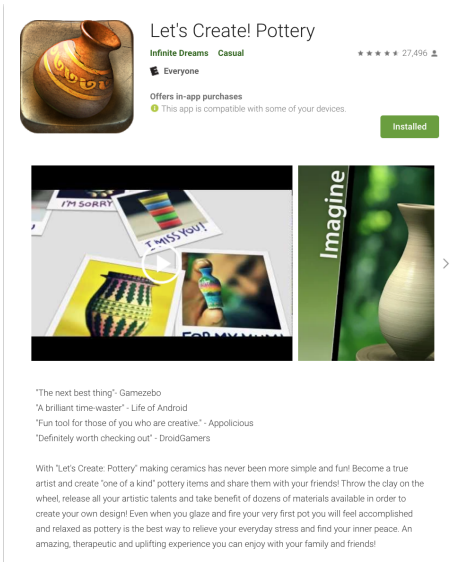
S122: Lego

Creator: Ole Kirk Christiansen
Url: `https://www.lego.com/en-us`
Year: 1949
Platform: physical toy
In chapter: Pattern - Signifiers Add Power
Description: The famous toy of interlocking bricks, Lego has spent decades also investing in sets that combine the structural possibilities of regular bricks with pieces that reference cultural signifiers like Harry Potter and the Avengers.

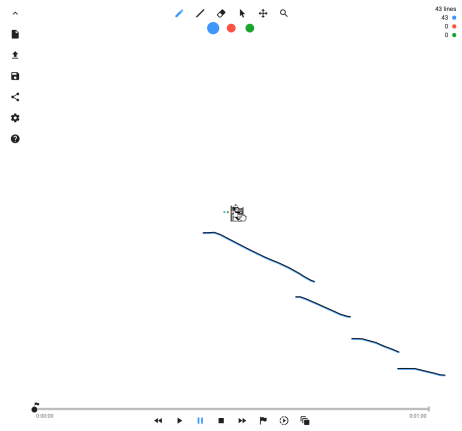


S123: Let's Create: Pottery

Creator: Infinite Dreams
Url: `https://play.google.com/store/apps/details?id=pl.idreams.pottery`
Year: 2012
Platform: Android
In chapters: Introduction, Motivation, Central Themes, Narrow and Shallow Possibility Spaces, Parametric Spaces, Pattern - Make It Real
Description: A virtual pottery wheel with many casual creator patterns, such as unlockable tools and the ability to 3D print your pot.



S124: Line Rider



Creator: Boštjan Čadež
Url: `https://www.linerider.com/`
Year: 2006
Platform: web
In chapter: Drawing Tools
Description: A early breakout hit of Flash, users can draw lines for a simulated sled-rider to slide across. Virtuosic users like to show off levels that synchonize with music.

S125: Logo language

Creator: Seymour Papert
Url: `http://el.media.mit.edu/logo-foundation/what_is_logo/logo_and_learning.html`
Year: 1967
In chapter: Casual Programming
Description: A language developed to give children a visual and interesting environment to learn to code.

S126: Ludi

Creator: Cameron Browne
Url: `http://cambolbro.com/games/yavalath/`
Year: 2007
Platform: evolutionary game generator
In chapter: Slow Creators
Description: A generator made by an AI researcher to create and evolve new Go-like mechanics. Created the commercially-successful game Yavalath.

S127: LUMINAI

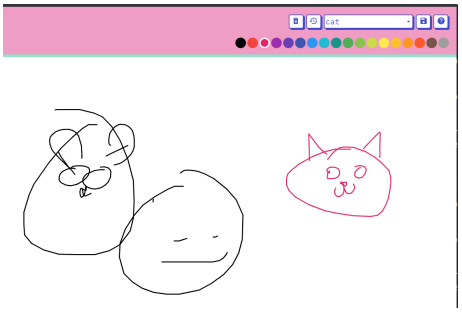


Creator: Expressive Machinery Lab
Url: `https://expressivemachinery.gatech.edu/projects/luminai/`
Year: 2013
Platform: installation
In chapter: A generative framework of generativity
Description: A virtual dance partner attempts to repeat and improvise on the interactor's Kinect-sensed moves.

S128: Mad Libs

Creator: Leonard Stern, Roger Price
Url: `http://www.madlibs.com/history/`
Year: 1953
Platform: pencil-and-paper template
In chapters: Creativity Support Frameworks, Existing Frameworks of Generativity, A generative framework of generativity, Slow Creators
Description: A surprising and generative template-filling game devised by two televisions writers to come up with new ideas.

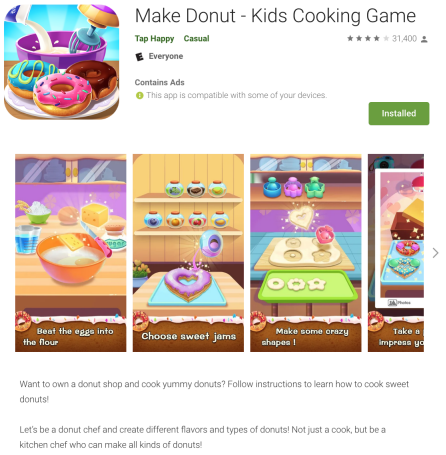
S129: Magic Sketchpad



Creator: Monica Dinculescu
Url: `https://magic-sketchpad.glitch.me/`
Year: 2018
Platform: web with Magenta's SketchRNN
In chapter: Pattern - Safety
Description: A mixed-initiative tool that finishes user drawings based on machine-learned patterns from Google's Quickdraw data.

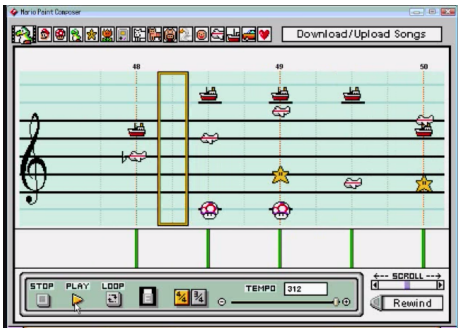
S130: Make Donut - Kids Cooking Game

Creator: Tap Happy
Url: `https://play.google.com/store/apps/details?id=com.k3games.donuthero.free`
Platform: Android
In chapters: Narrow and Shallow Possibility Spaces, Pattern - Signifiers Add Power
Description: A simple task-based app for making virtual donuts.



S131: Mario Paint

Creator: Nintendo (designer: Hirofumi Matsuoka)
Url: `http://emulator.online/snes/mario-paint/`
Year: 1992
Platform: SNES
In chapters: Motivation, Slow Creators
Description: A sub-creator within a creativity game, one of many grid-based music tools.



S132: Autodesk Maya

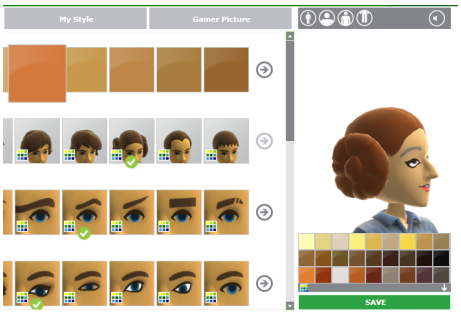
Creator: Alias Systems Corporation
Year: 1998
Platform: Windows
In chapters: Central Themes, Pattern - Safety
Description: A complex, difficult and powerful 3D CAD program.

S133: Mendel



Creator: Owen Bell
Url: `https://gamecenter.nyu.edu/projects/mendel/`
Year: 2016
Platform: desktop app
In chapter: Parametric Spaces
Description: A game in which the user collects, breeds, and catalogs slow-growing trees (from genetic algorithms).

S134: Miitomo

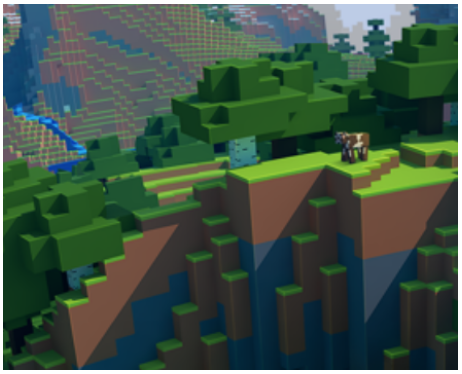


Creator: Nintendo EPD
Url: `https://miitomo.com/`
Year: 2016
In chapter: Motivation
Description: Users created digital versions of themselves, which could be used to play social games or visit each other.

S135: millefiore

Creator: 6th century Sweden, 15th Murano
Url: `https://en.wikipedia.org/wiki/Millefiori`
Year: 6th century
Platform: glass-working technique
In chapter: Pattern - Repetition, Repetition, Repetition
Description: A traditional glassmaking technique where one pattern of glass is stretched, cut, and rebundled, repeatedly, to generate dozens of tiny perfect duplicates.

S136: Minecraft

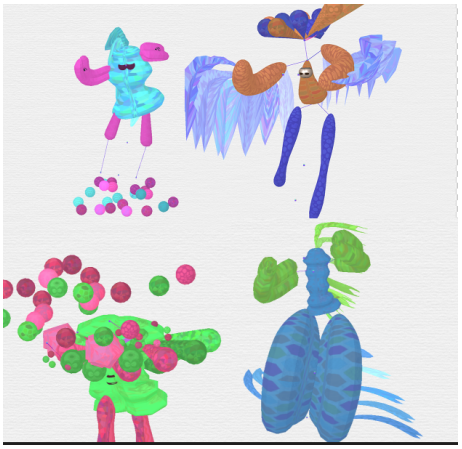


Creator: Mojang, Markus Persson
Url: `https://minecraft.net/en-us/`
Year: 2011
Platform: Windows/OSX
In chapters: Motivation, Creativity Support Frameworks, A generative framework of generativity, Frameworks of Interactivity, Pattern - Possibility Space Annotation, Pattern - Self-promotion
Description: A voxel-based mining game which also became a massive platform for user-based creativity and fan-made tools.

S137: Monster Hunter: World

Creator: Capcom
Url: `http://www.monsterhunterworld.com/`
Year: 2018
Platform: console game
In chapter: Pattern - Expressing Identity
Description: A game with an impressively detailed face-editing system.

S138: Monster Mirror



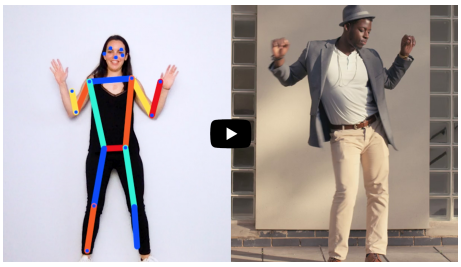
Creator: Kate Compton
Year: 2018
Platform: web,THREE.js, Posenet
In chapter: A generative framework of generativity
Description: A commissioned piece by Google, it turns user pose data into a virtual generative puppet.

S139: Monument Valley 2

Creator: ustwo games
Url: `https://play.google.com/store/apps/details?id=com.ustwo.monumentvalley2&hl=en_US`
Year: 2017
In chapter: Pattern - Repetition, Repetition, Repetition
Description: Embedded at the end of each section of this game is a casual creator where the player draws a mandala to represent that level.

S140: Move Mirror

Creator: Google Creative Lab
Url: `https://experiments.withgoogle.com/move-mirror`
Year: 2018
Platform: web,THREE.js, Posenet
In chapter: A generative framework of generativity
Description: Demonstrates Posenet by matching the user's detected pose with photos from a corpus.



S141: Microsoft Paint

Creator: Microsoft
Url: `https://jspaint.app/#local:4459f0f035f77`
Year: 1985
Platform: Windows 1.0 and on
In chapters: Central Themes, Drawing Tools, Pattern - Entertaining Evaluations, Pattern - Transformation
Description: A simple pixel-based drawing tool included with many versions of Windows.



S142: Magic: The Gathering

Creator: Wizards of the Coast
Url: `https://magic.wizards.com/en`
Year: 1993
Platform: card game
In chapters: Slow Creators, Pattern - Possibility Space Annotation
Description: A collectible card game with a large community devoted to documentation.

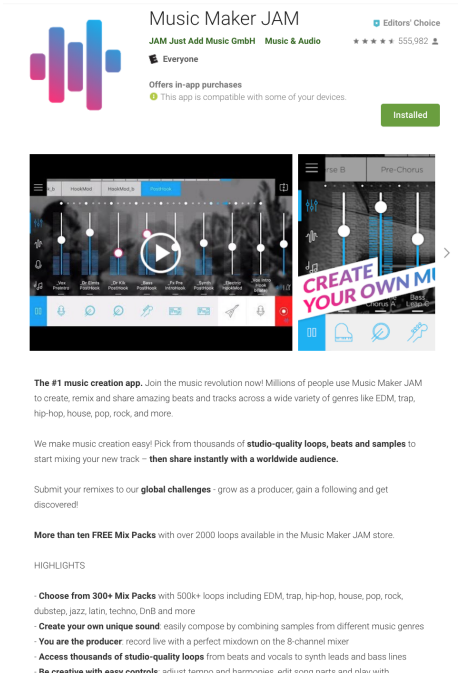
S143: Will Harvey's Music Construction Set

Creator: Will Harvey
Url: `https://archive.org/details/a2_Music_Construction_Set_1983_Electronic_Arts_cr_Midwest_Pirates_Guild`
Year: 1984
Platform: Apple II
In chapter: Slow Creators
Description: An early form of music-creation software, notable because it could both play and edit scores.



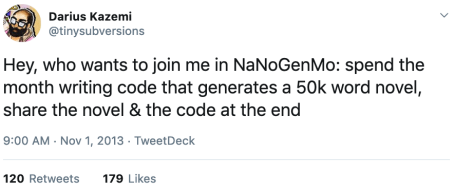
S144: Music Maker JAM

Creator: JAM Just Add Music GmbH
Url: `https://play.google.com/store/apps/details?id=com.magix.android.mmjam&hl=en_US`
Platform: Android
In chapter: Central Themes
Description: A dance music mixology app.



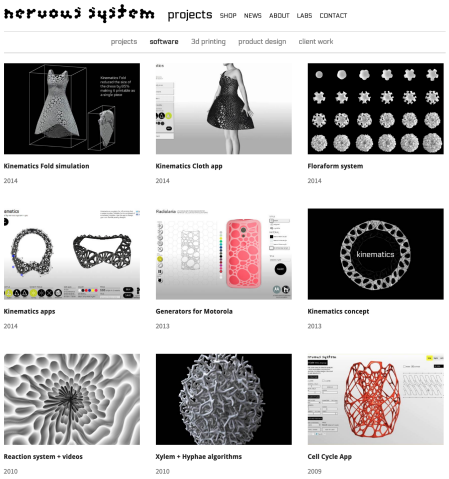
S145: NaNoGenMo (National Novel Generating Month)

Creator: Darius Kazemi
Url: `https://nanogenmo.github.io/`
Year: 2013
Platform: annual design challenge
In chapters: Slow Creators, Pattern - No Blank Slate
Description: A puckish reflection on National Novel Writing Month, participants must write an algorithm that can generate a 50,000 word novel.



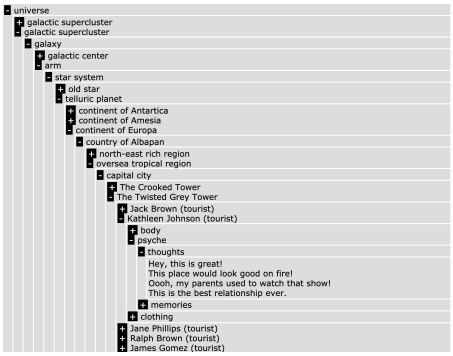
S146: Kinematics Apps

Creator: Nervous System
Url: `https://n-e-r-v-o-u-s.com/projects/software/`
Year: 2014
Platform: web apps for 3D printing
In chapters: Possibility Spaces, Pattern - Make It Real
Description: A suite of many web apps which can generate 3D printable Voronoi geometry for necklaces, rings, and mesh-based dresses.

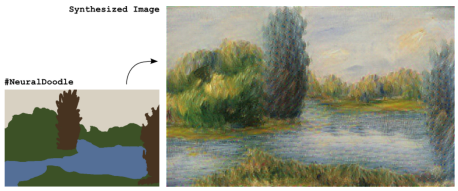


S147: Nested

Creator: Orteil
Url: `http://orteil.dashnet.org/nested`
Year: 2011
In chapters: A generative framework of generativity, Pattern - Possibility Space Annotation
Description: Appears at first to be a list of universes, but each entity can be recursively expanded into its components: planets, cities, people, atoms.



S148: Neural Doodle



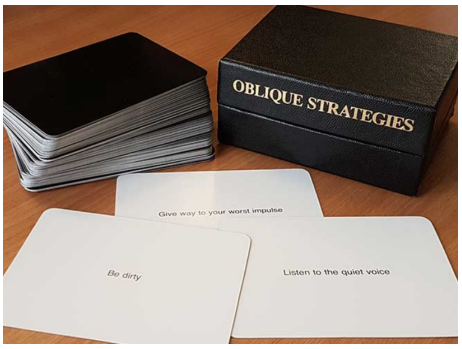
Creator: Alex Champandard
Url: `https://github.com/alexjc/neural-doodle`
Year: 2016
Platform: Python app
In chapters: Drawing Tools, Pattern - Transformation
Description: A generative adversarial network for filling in simple sketches with textures from impressionist paintings.

S149: No Man's Sky



Creator: Hello Games
Url: `https://www.nomanssky.com/`
Year: 2016
Platform: PS4, Windows, OSX
In chapter: A generative framework of generativity
Description: A game about exploring a massive generative cosmos. Players enjoy documenting the generative planets and creatures they find.

S150: Oblique Strategies



Creator: Brian Eno, Peter Schmidt
Url: `http://www.rtqe.net/ObliqueStrategies/`
Year: 1979
Platform: card deck
In chapter: Creativity Support Frameworks
Description: A deck of cards by ambient musician Brian Eno to break out of ruts in creative thinking.

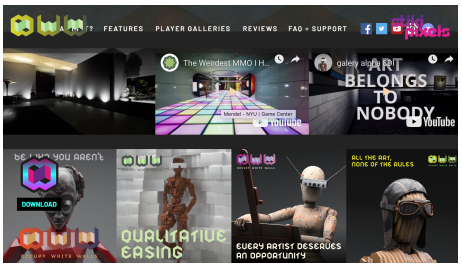
S151: obliteration room

Creator: Yayoi Kusama
Url: `https://play.qagoma.qld.gov.au/looknowseeforever/works/obliteration_room/`
Year: 2002
Platform: installation
In chapter: Drawing Tools
Description: A gallery show in which visitors can cover a blank white room in colored dot stickers.



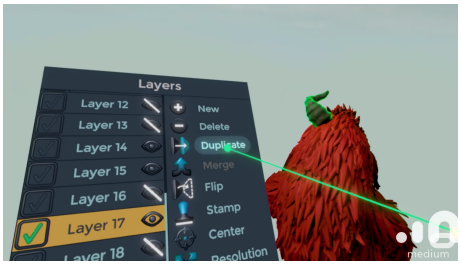
S152: Occupy White Walls

Creator: Stikipixels
Url: `https://www.oww.io/`
Year: 2018
Platform: Windows app
In chapter: Pattern - Entertaining Evaluations
Description: An unusual MMO in which players create virtual art galleries (with a combination of contemporary internet art and 18th c. classics) to be visited by humans and virtual agents.



S153: Oculus Medium

Creator: Oculus VR
Url: `https://www.oculus.com/medium/`
Year: 2016
Platform: Oculus Rift VR
In chapters: Introduction, Drawing Tools
Description: A voxel-based sculpting tool for VR



S154: Okami



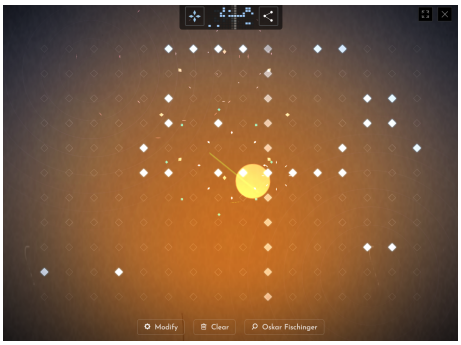
Creator: Clover Studio
Url: `http://www.okami-game.com/`
Year: 2006
Platform: Wii
In chapter: Drawing Tools
Description: The player uses the Wiimote to draw patterns which invoke spells.

S155: Opus Magnum



Creator: Zachtronics
Url: `http://www.zachtronics.com/opus-magnum/`
Year: 2017
Platform: OSX
In chapter: Pattern - Self-promotion
Description: Players create complex alchemical machines, which can be downloaded as easily-shared animated GIFs.

S156: Oskar Fischinger's 117th Birthday



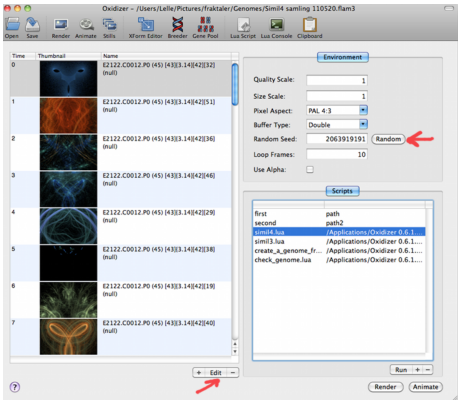
Creator: Google Doodle Team
Url: `https://www.google.com/doodles/oskar-fischingers-117th-birthday`
Year: 2017
Platform: web
In chapters: Motivation, Slow Creators
Description: An interactive interpretation of Fischinger's animations, in which the user draws a score for the simulator to follow.

S157: Ouija board



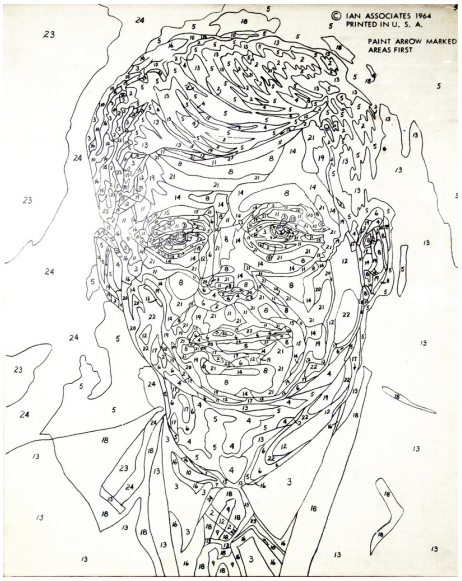
Creator: Elijah Bond
Year: 1890
Platform: board game
In chapter: Existing Frameworks of Generativity
Description: A parlor game for contacting spirits by reading the emergent pattern of the pointer's movement.

S158: Oxidizer



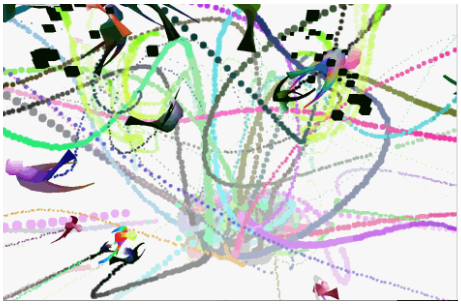
Creator: Rampant Mac
Url: `http://rampant-mac.com/wp/`
Year: 2006
Platform: desktop app
In chapter: Parametric Spaces
Description: An editor and viewer for Electic Sheep patterns.

S159: Paint-by-numbers



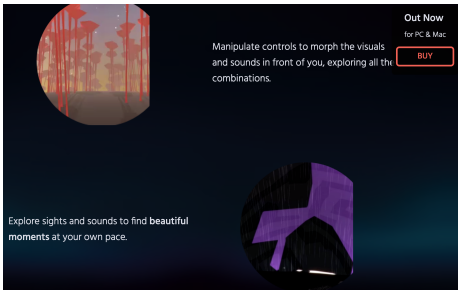
Creator: Max S. Klein, Dan Robbins
Url: `https://en.wikipedia.org/wiki/Paint_by_number`
Year: 1950
In chapters: Motivation, Narrow and Shallow Possibility Spaces, Drawing Tools, Pattern - Transformation
Description: A common pastime kit with a set of paints and numbered spaces to fill in. Now has many digital variants as well.

S160: PaintThrush



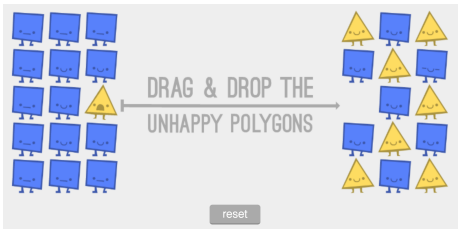
Creator: Kate Compton
Url: `https://galaxykate.itch.io/paintthrush`
Year: 2015
Platform: web, THREE.js, processing.js, Leap Motion
In chapters: A generative framework of generativity, Drawing Tools
Description: An art toy where the user conducts virtual birds (using the LeapMotion) that leave behind watercolor trails.

S161: Panoramical



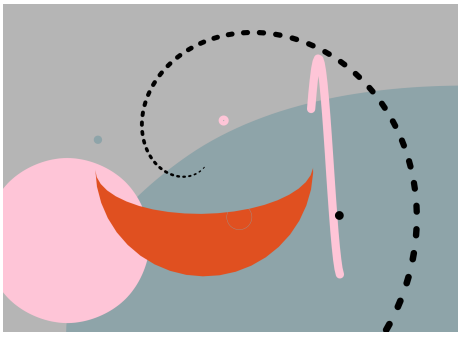
Creator: Fernando Ramallo, David Kanaga
Url: `http://www.panoramic.al/`
Year: 2015
Platform: desktop
In chapters: Slow Creators, Tracery and Chancery
Description: A meditative on-rails experience where the user changes the parameters of a sound and visualization show.

S162: Parable of the Polygons



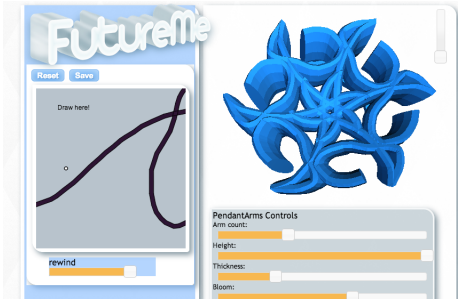
Creator: Vi Hart, Nicky Case
Url: `https://ncase.me/polygons/`
Year: 2014
Platform: web
In chapter: A generative framework of generativity
Description: An explorable explanation of the rules behind segregation and how they can be undone.

S163: Patatap



Creator: Jono Brandel and Lullatone
Url: `https://patatap.com/`
Year: 2014
Platform: web
In chapter: Slow Creators
Description: A web-based music-and-graphics synthesizer where any key pressed creates a different shape and tone. Pairs well with unusual non-keyboard inputs like the MakeyMakey.

S164: Pendant Maker



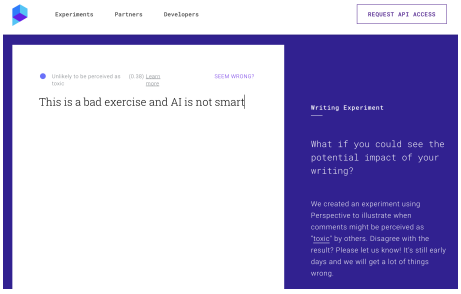
Creator: Kate Compton
Url: `http://studiobreak.com/highlight-episode-17-the-3d-printer-experience/`
Year: 2013
In chapters: Drawing Tools, Pattern - Make It Real
Description: A casual creator commissioned by a 3D printing startup as a way for non-artists to create interesting 3D printable mandala/flower pendants.

S165: Pentameter pentominoes



Creator: UVa's Puzzle Poetry group
Url: `https://bpasanek.github.io/puzzlepoesis/`
Year: 2017
Platform: physical toy
In chapter: Slow Creators
Description: Brad Pasanek's lasercut poetry puzzle pieces, which can be assembled into the original sonnet, or recombined into a new one.

S166: Perspective API



Creator: Google
Url: `https://www.perspectiveapi.com/#/`
Year: 2017
Platform: API
In chapter: Pattern - Entertaining Evaluations
Description: A tool from Google that purports to detect toxicity in text. Comes with an online test tool which was helpful in determining how much to trust that sales pitch.

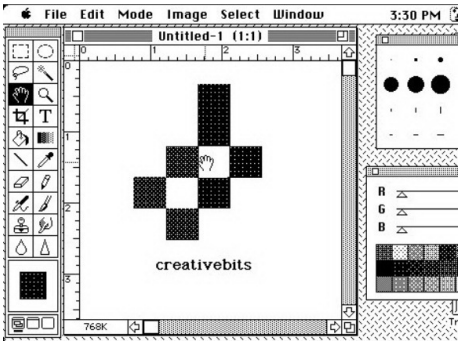
S167: Petalz



Creator: Sebastian Risi, Joel Lehman, David B. D'Ambrosio, Ryan Hall and Kenneth O. Stanley
Url: `http://www.aigameresearch.org/portfolio-item/petalz/`
Year: 2012
Platform: Facebook game
In chapters: A generative framework of generativity, Possibility Spaces, Pattern - Safety, Pattern - Possibility Space Annotation, Pattern - Self-promotion, Pattern - Make It Real
Description: A social game about breeding virtual flowers. For an academic project, it had a user community and many casual creator social patterns.

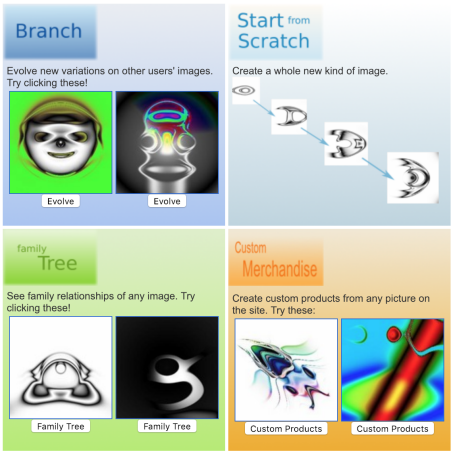
S168: Adobe

Creator: Adobe, Thomas Knoll, John Knoll
Year: 1988
Platform: Windows, OSX
In chapters: Drawing Tools, Pattern - No Blank Slate, Pattern - Safety, Pattern - Transformation
Description: Famous photomanipulation software, contains a few casual creator elements, like the Mosaic filter.



S169: Picbreeder

Creator: Evolutionary Complexity Research Group at UCF
Url: `http://picbreeder.org/`
Year: 2007
Platform: web
In chapters: A generative framework of generativity, Parametric Spaces
Description: An academic project to evolve pictures, allows users to breed their own and receive credit for them.

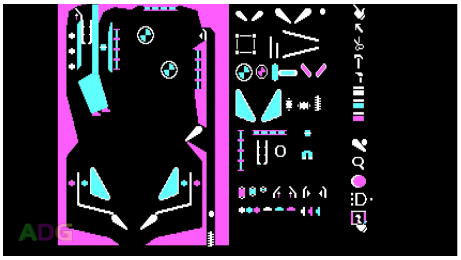


S170: Pictionary

Creator: Robert Angel and Gary Everson
Url: `https://en.wikipedia.org/wiki/Pictionary`
Year: 1985
Platform: party game
In chapter: Pattern - Safety
Description: A classic game in which one player must draw a difficult subject for their team members to guess.

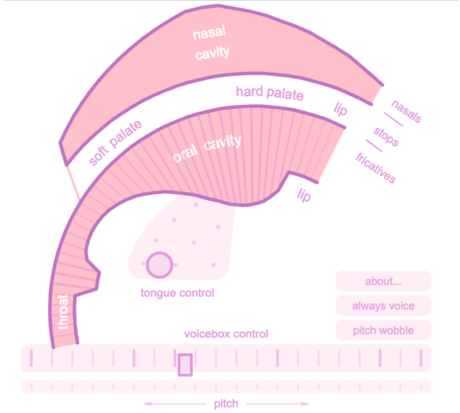


S171: Pinball Construction Set



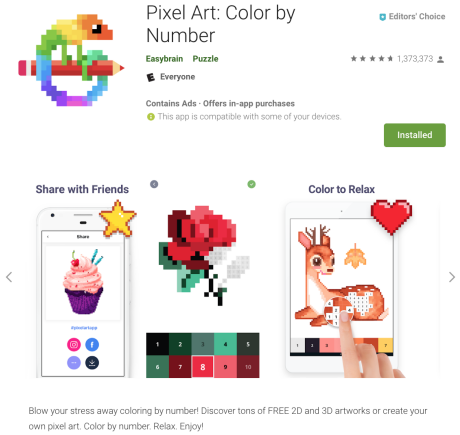
Creator: Bill Budge
Url: `https://archive.org/details/a2_asimov_pinball_construction_set`
Year: 1983
Platform: Apple II
In chapters: Slow Creators, Casual Programming
Description: An editor and simulator for designing virtual pinball tables.

S172: Pink Trombone



Creator: Neil Thapen
Url: `https://www.dood.al/pinktrombone/`
Year: 2017
Platform: web
In chapter: Motivation
Description: An explorable explanation and sound simulator for how the human throat produces sound.

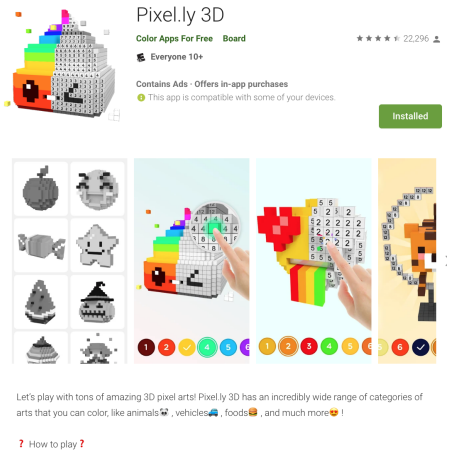
S173: Pixel Art: Color by Number



Creator: Easybrain
Url: `https://play.google.com/store/apps/details?id=com.europosit.pixelcoloring`
Platform: Android
In chapter: Narrow and Shallow Possibility Spaces
Description: One of many similar tap-to-color pixel art apps.

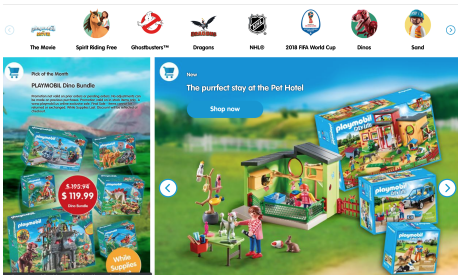
S174: Pixel.ly 3D

Creator: Color Apps For Free
Url: `https://play.google.com/store/apps/details?id=com.pixel.coloring.color.by.number`
Platform: Android
In chapters: Narrow and Shallow Possibility Spaces, Pattern - Safety
Description: One of many similar tap-to-color 3D voxel art apps.



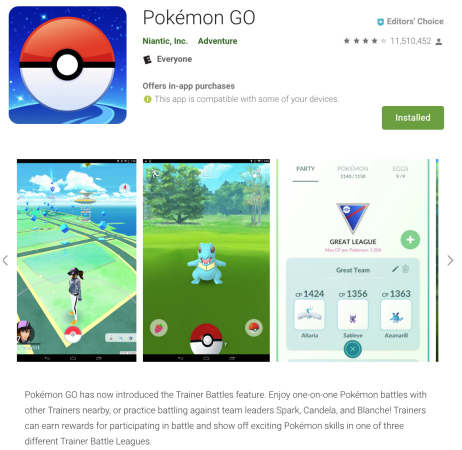
S175: Playmobil

Creator: Hans Beck
Url: `https://www.playmobil.us/`
Year: 1974
Platform: physical toy
In chapter: Pattern - Signifiers Add Power
Description: A toy line with many different sets for many cultural signifiers, including a TSA checkpoint, Vermeer's 'Milkmaid', and a beer-brewing monk.



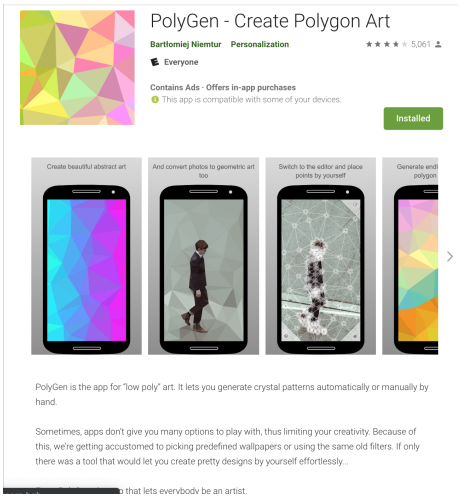
S176: Pokémon Go

Creator: Niantic
Url: `https://www.pokemongo.com/en-us/`
Year: 2016
Platform: Android
In chapter: A generative framework of generativity
Description: An augmented-reality game about collecting monsters in real world locations.



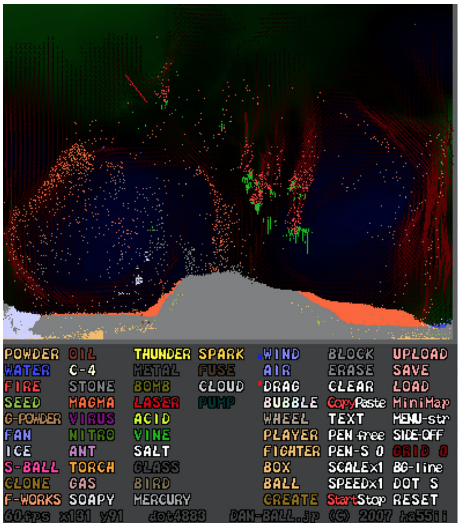
S177: PolyGen - Create Polygon Art

Creator: Bartłomiej Niemtur
Url: `https://play.google.com/store/apps/details?id=com.bitbotany.polygen`
Platform: Android
In chapter: Central Themes
Description: A generator to make pretty geometric abstract triangulations



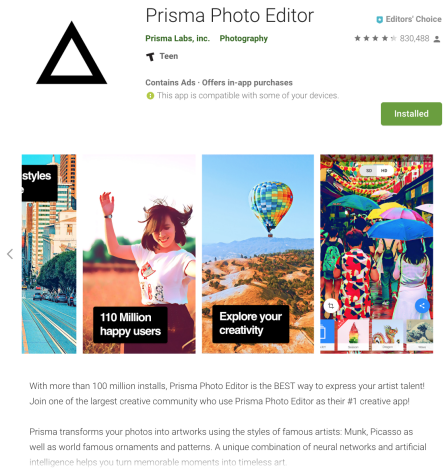
S178: The Powder Game

Creator: Campo Santo Games
Year: 2016
Platform: PC
In chapter: A generative framework of generativity
Description: A riff on the classic Conway's Game of Life, this game uses cellular automata to simulate scenarios drawn by the user.



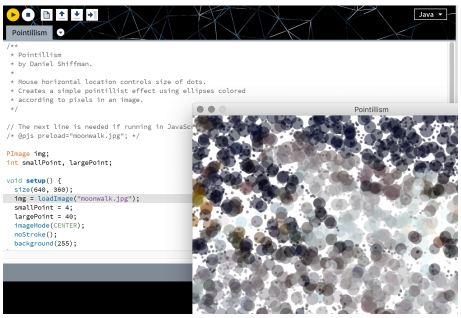
S179: Prisma Photo Editor

Creator: Prisma labs
Url: `https://prisma-ai.com/`
Platform: Android
In chapter: Pattern - Transformation
Description: A photo-kaleidoscope app with a plethora of different algorithmic manipulations for user-provided photos.



S180: Processing

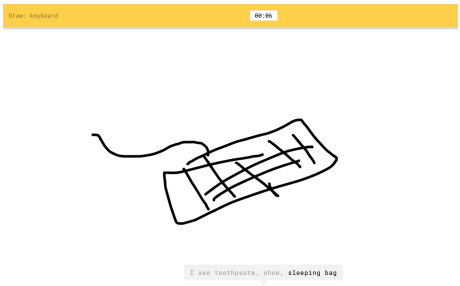
Creator: Casey Reas, Ben Fry
Url: `https://processing.org/`
Year: 2001
Platform: Java IDE
In chapter: Casual Programming
Description: A combination IDE, Java extension, and launcher, so that novice users can quickly build visual interactive programs.



S181: One Hundred Thousand Billion Sonnets (Cent Mille Milliards de poemes)

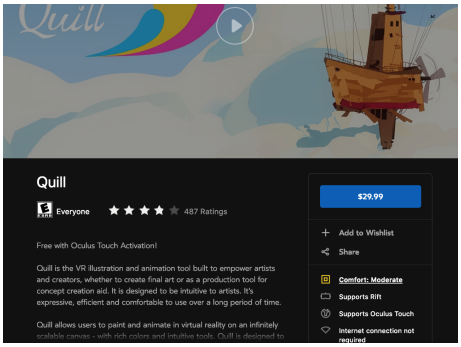
Creator: Raymond Queneau (interactive web version by Gordon Dow)
Url: `http://www.growndodo.com/wordplay/oulipo/10%5E14sonnets.html`
Year: 1961
Platform: rearrangeable poem
In chapter: Slow Creators
Description: A poem that demands to be cut into individual lines and reassembled.

S182: Quick, Draw!



Creator: Google
Url: `https://quickdraw.withgoogle.com/`
Year: 2016
Platform: web, Tensorflow
In chapters: A generative framework of generativity, Pattern - Entertaining Evaluations, Pattern - Safety
Description: Google's adversarial AI-training game where the user draws something and the AI must guess what it is. They later released all the drawing data, which spawned other projects.

S183: Quill



Creator: Facebook
Url: `https://www.oculus.com/experiences/rift/1118609381580656/`
Year: 2016
Platform: VR
In chapter: Drawing Tools
Description: A professional-targeted drawing tool for VR.

S184: Rainbow Loom



Creator: Cheong Choon Ng
Url: `https://www.rainbowloom.com/`
Year: 2010
Platform: physical toy
In chapter: Motivation
Description: A creativity toy for weaving together colorful loops of plastic.

S185: Recipe Dice

Creator: Lea Redmond
Url: `https://www.leafcutterdesigns.com/gifts/recipe-dice/`
Platform: physical toy
In chapter: A generative framework of generativity
Description: A set of dice with ingredients on each side, to create new chili or chai recipes.



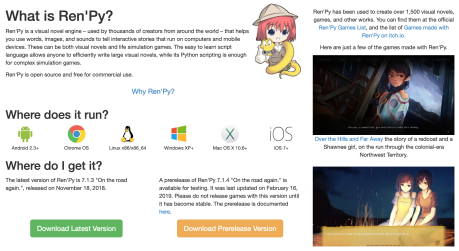
S186: Magnetic Poetry

Creator: Dave Kapell
Url: `https://magneticpoetry.com/`
Year: 1993
Platform: physical poetry aid
In chapter: Slow Creators
Description: A set of words on magnets for combinatorial poetry by novice (and distracted) people.

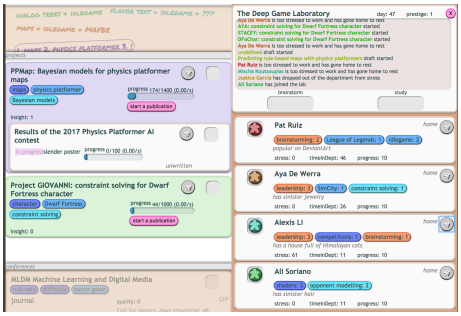


S187: Ren'Py

Creator: Tom 'PyTom' Rothamel
Url: `https://www.renpy.org/`
Year: 2004
Platform: desktop app
In chapter: Slow Creators
Description: An editor for making visual novels



S188: Revisions

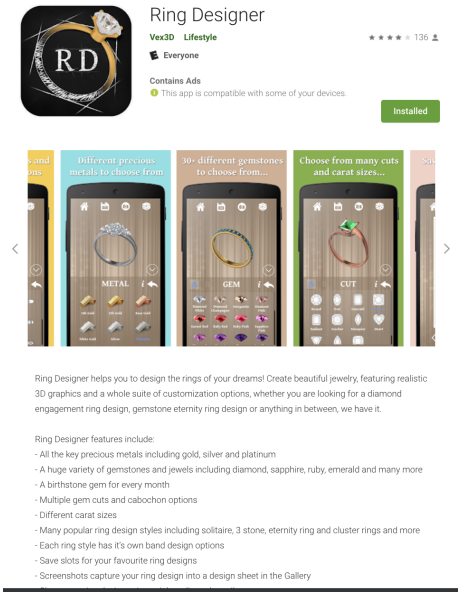


Creator: Kate Compton
Url: `http://www.galaxykate.com/revisions/`
Year: 2017
Platform: HTML/Web
In chapters:
Description: An idle game about running an AI reseach lab, where players curate machine-generated whiteboard ideas (via Tracery) into virtual research projects and papers.

S189: RimWorld

Creator: Ludeon Studios
Url: `https://rimworldgame.com/`
Year: 2013
Platform: Windows, OSX, Linux
In chapter: Pattern - Possibility Space Annotation
Description: A survival horror base-builder game, where complex generated characters drive the events and create emerent narrative situations.

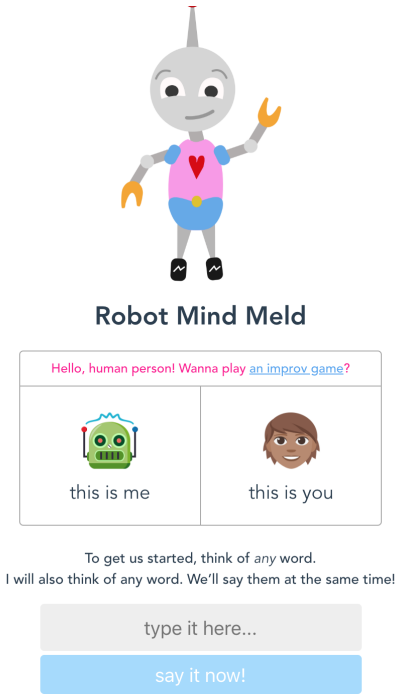
S190: Ring Designer



Creator: Vex3d
Url: `https://play.google.com/store/apps/details?id=com.Vex3d.RingDesigner`
Platform: Android
In chapters: A generative framework of generativity, Parametric Spaces
Description: A grammar/socket-based generator for designing jewelry

S191: Robot Mind Meld

Creator: Alex Lew
Url: `http://robotmindmeld.com/`
In chapter: Parametric Spaces
Description: A game in which the human and computer try to agree on a word by traversing ConceptNet.



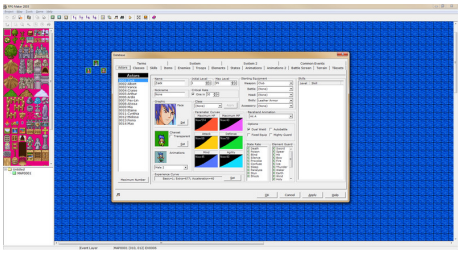
S192: RollerCoaster Tycoon

Creator: Chris Sawyer
Url: `http://www.rollercoastertycoon.com/rollercoaster-tycoon-classic/`
Year: 1999
Platform: Windows
In chapter: Pattern - Entertaining Evaluations
Description: A game about designing roller coasters and laying out a theme park for maximum profit (or disaster).



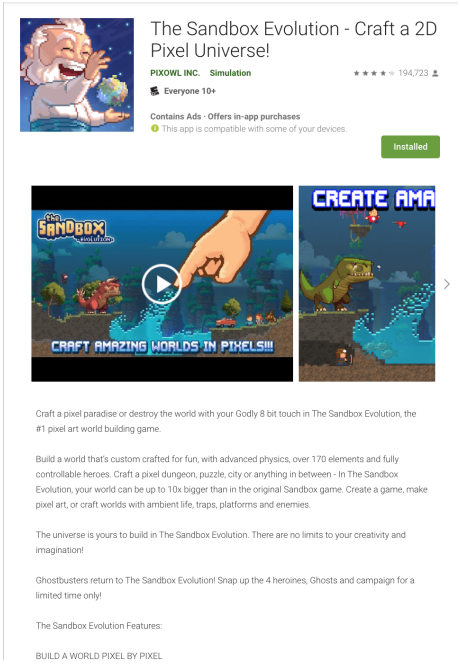
S193: RPGMaker

Creator: ASCII/Enterbrain
Url: `https://www.rpgmakerweb.com/`
Year: 1992
Platform: desktop app
In chapter: Slow Creators
Description: A long-running series of software for making classic Final Fantasy-style RPGs.



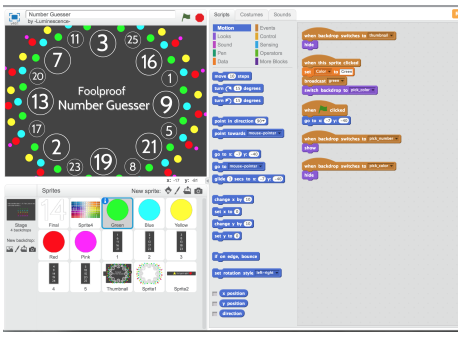
S194: The Sandbox Evolution - Craft a 2D Pixel Universe

Creator: PIXOWL Inc
Url: `https://play.google.com/store/apps/details?id=com.pixowl.tsb2`
Platform: Android
In chapter: A generative framework of generativity
Description: A more commercial and game-like reinterpretation of the Powder/Dust game.

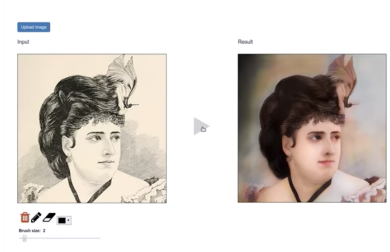


S195: Scratch

Creator: MIT Media Lab
Url: `https://scratch.mit.edu/`
Year: 2007
Platform: web, Windows, OSX
In chapters: Motivation, Casual Programming, Expert evaluation
Description: A language, IDE, and sharing site for children to make, share, and modify simple programs.

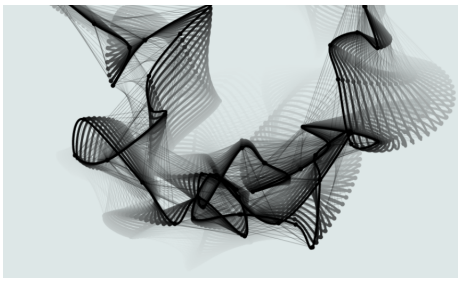


S196: Project Scribbler



Creator: Sangkloy et all
Url: `https://adoberesearch.ctlprojects.com/project/scribbler-controlling-deep-image-synthesis-with-sketch-and-color/`
Year: 2017
Platform: research app
In chapter: Pattern - Transformation
Description: An algorithm to colorize and shade line drawings.

S197: Scribbles

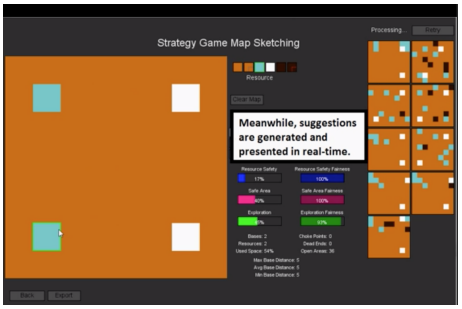


Creator: Kate Compton
Url: `http://www.galaxykate.com/apps/Prototypes/Scribbles/`
Year: 2005/2015
Platform: web
In chapter: Drawing Tools
Description: A drawing tool where the user's drawn lines wriggle to life. Several versions.

S198: Second Life

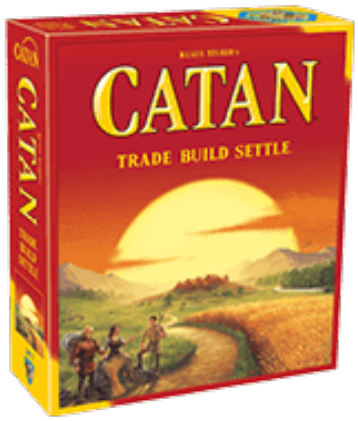
Creator: Linden Lab
Url: `https://secondlife.com/`
Year: 2003
Platform: multiplayer desktop
In chapter: Pattern - Self-promotion
Description: A vast MMO where users build complex architecture, social systems, and economies.

S199: Sentient Sketchbook: Computer-Assisted Game Level Authoring



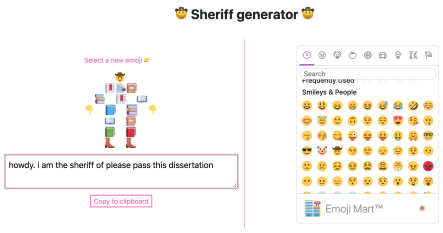
Creator: Antonios Liapis
Url: `http://www.sentientsketchbook.com/`
Year: 2014
Platform: desktop research app
In chapters: Slow Creators, Expert evaluation
Description: An academic project to help users quickly navigate the possibility space of strategy-game maps.

S200: Settlers of Catan



Creator: Rio Grande games
Url: `https://www.catan.com/`
Year: 1995
Platform: board game
In chapters: Existing Frameworks of Generativity, A generative framework of generativity
Description: A common board game with a reconfigurable randomly-generated map.

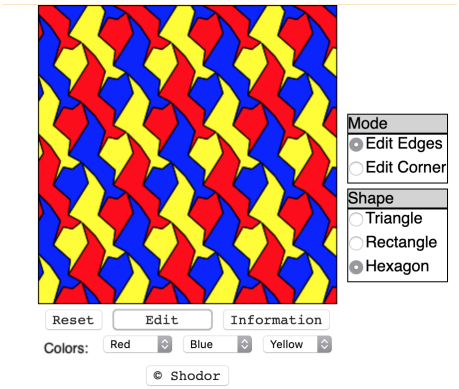
S201: Sheriff generator



Creator: Pete Nykänen
Url: `https://howdy.petetnt.xyz/`
Platform: web
In chapter: Pattern - Signifiers Add Power
Description: A web-based generator for the common Twitter meme of 'Howdy I'm the Sheriff' emoji memes.

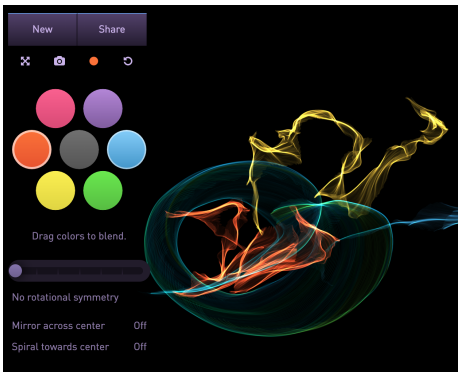
S202: Tesselate

Creator: Shodor
Url: `http://www.shodor.org/interactivate/activities/Tessellate/`
Platform: web
In chapter: Pattern - Repetition, Repetition, Repetition
Description: An online editor and visualizer for M.C. Escher style tessellations.



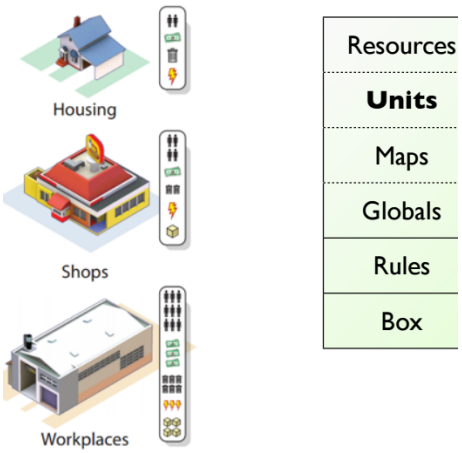
S203: Silk - Interactive Generative Art

Creator: Yuri Vishnevsky
Url: `http://weavesilk.com/`
Platform: web, mobile
In chapter: Pattern - Repetition, Repetition, Repetition
Description: An editor for gesture-based layering of translucent generative and symmetric linework.



S204: SimCity

Creator: Maxis
Url: `https://www.ea.com/games/simcity`
Year: 2012
Platform: desktop
In chapters: A generative framework of generativity, Pattern - Entertaining Evaluations
Description: A series of city-building games, where the user's city-building creativity is tempered by the simulations pushback.



S205: Simpsonize me

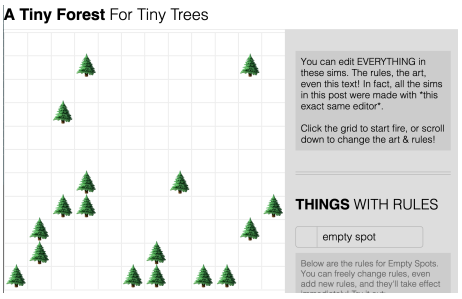


Creator: Fox
Url: `http://www.simpsonizeme.com/`
Year: 2007
Platform: web
In chapter: Pattern - Make It Real
Description: A now-defunct system let users see themselves as Simpsons characters in time for the 2007 movie (Signifiers).

S206: The Sims

Creator: Maxis
Url: `https://www.ea.com/games/the-sims`
Year: 2000
Platform: Windows, OSX
In chapters: Pattern - Entertaining Evaluations, Pattern - No Blank Slate, Pattern - Self-promotion, Pattern - Signifiers Add Power
Description: An immensely succesful dollhouse simulator, and the platform for many creative fan-made projects.

S207: Simulating the World in Emoji



Creator: Nicky Case
Url: `http://ncase.me/simulating/`
Year: 2015
In chapters: A generative framework of generativity, Pattern - Signifiers Add Power
Description: A creator for making cellular automata simulations: users select emoji for each tile, and write the rules for them.

S208: Sketch Aquarium and Sketch Town



Creator: teamLab
Url: `https://www.teamlab.art/w/aquarium`
Year: 2013
Platform: installation
In chapter: Pattern - Transformation
Description: A pair of experiences in which children can color and scan artwork to see it appear in a simulated town or ocean.

S209: Sketch Nation Create



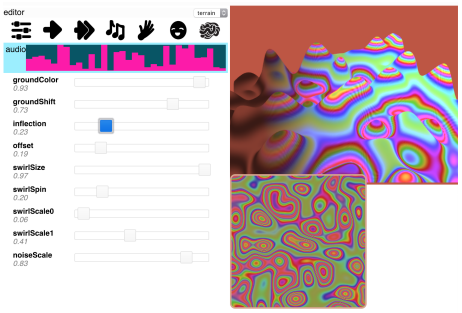
Creator: Sketch Nation Inc
Url: `https://play.google.com/store/apps/details?id=com.engineousgames.Sketch_Nation`
Platform: Android
In chapter: Slow Creators
Description: An app to edit simple game templates (or make more complex modifications).

S210: Sketchup

Creator: @last Software, Google, Trimble
Url: `https://www.sketchup.com/`
Year: 2000
Platform: Windows, OSX
In chapter: Frameworks of Interactivity
Description: CAD software that attempts to be simpler and more intuitive than usual.

S211: Slidertoy

Creator: Kate Compton
Url: `https://github.com/galaxykate/slidertoy`
Year: 2018
Platform: JS app template
In chapter: Parametric Spaces
Description: A framework made to demonstrate the way that vector-based generative content can be navigated.



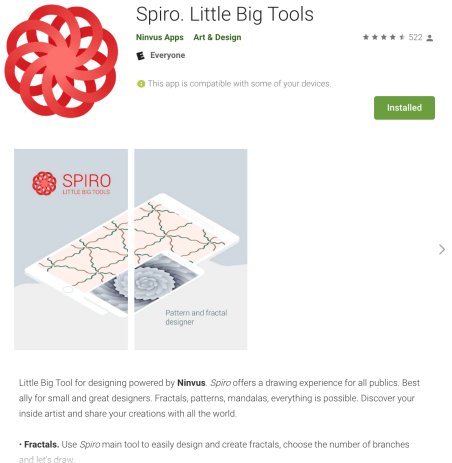
S212: Spelunky

Creator: Derek Yu
Url: `https://www.spelunkyworld.com/`
Year: 2008
Platform: desktop game
In chapter: Possibility Spaces
Description: A game with generative platformer levels.

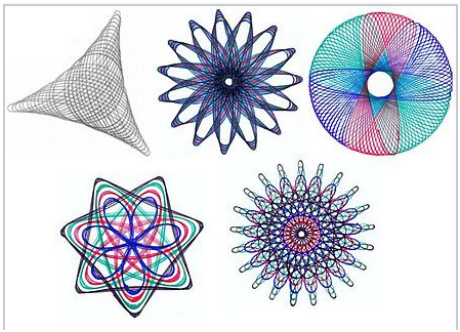


S213: Spiro. Little Big Tools

Creator: Ninvus Apps
Url: `https://play.google.com/store/apps/details?id=com.ninvus.spiro&hl=en_US`
Platform: Android
In chapter: Pattern - Repetition, Repetition, Repetition
Description: A generative drawing tool where touch-drawn lines are reflected symmetrically to create complex patterns.

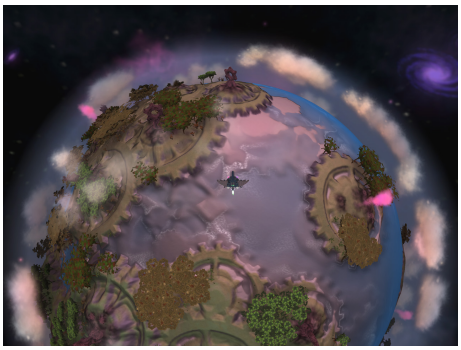


S214: Spirograph



Creator: Denys Fisher, Hasbro
Year: 1965
Platform: physical toy
In chapters: Introduction, Existing Frameworks of Generativity, Possibility Spaces, Drawing Tools
Description: A gear-based drawing toy with possible antecedents going back to the 1800s.

S215: Spore



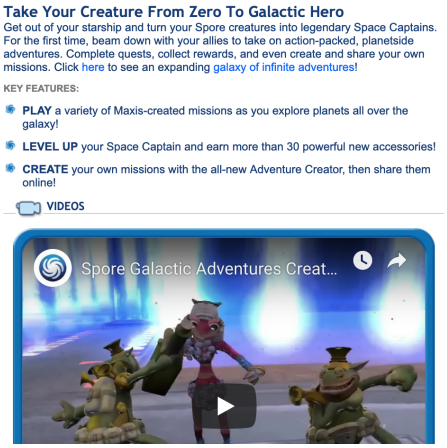
Creator: Maxis
Url: `https://www.spore.com/`
Year: 2008
Platform: indows/OSX
In chapters: Motivation, A generative framework of generativity, Possibility Spaces, Narrow and Shallow Possibility Spaces, Slow Creators, Pattern - No Blank Slate, Pattern - Safety, Pattern - Self-promotion, Pattern - Make It Real
Description: A suite of casual creators with a game wrapped around them. Had a robust sharing site with millions of uploaded creations and vigorous community creativity.

S216: Spore Creature Creator



Creator: Maxis
Url: `https://www.spore.com/trial`
Year: 2008
Platform: Windows/OSX
In chapters: Motivation, A generative framework of generativity, Frameworks of Interactivity, Narrow and Shallow Possibility Spaces, Slow Creators, Pattern - Entertaining Evaluations, Pattern - No Blank Slate, Pattern - Safety, Pattern - Repetition, Repetition, Repetition, Pattern - Signifiers Add Power
Description: The inspiration for this dissertation and one of the most uniformly succesful casual creators.

S217: Spore: Galactic Adventures



Creator: Maxis
Url: `http://www.spore.com/what/ga`
Year: 2009
In chapters: Slow Creators, Conclusion
Description: An expansion to Spore where users could design simple levels and planet geography.

S218: Starbound

Creator: Chucklefish
Url: `https://playstarbound.com/`
Year: 2016
In chapter: Pattern - Possibility Space Annotation
Description: A space exploration game with lively mapping and annotation of its generative worlds.

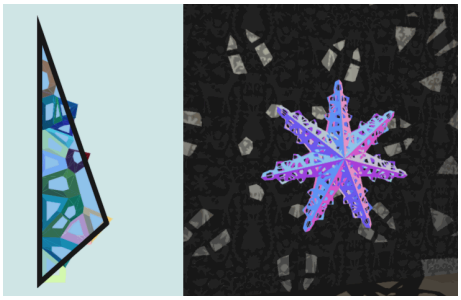
S219: Stardew Valley

Creator: Chucklefish
Url: `https://www.stardewvalley.net/`
Year: 2016
Platform: OSX
In chapter: Motivation
Description: Farming simulator with a simple character creator



S220: Stjärna

Creator: Kate Compton
Year: 2019
Platform: web
In chapter: A generative framework of generativity
Description: A creator for Swedish holiday star lanterns, using Voronoi patterns to create non-overlapping patterns of holes.



S221: Storyteller

Creator: Daniel Benmergui
Url: `http://www.storyteller-game.com/,https://www.kongregate.com/games/danielben/storyteller`
Year: 2008
Platform: web
In chapters: A generative framework of generativity, Slow Creators
Description: A long-development virtual comic book, where the events of the end panels are determined by the user's placement of the characters in the early panels.



S222: style2paints

Creator: lllyasviel
Url: `https://style2paints.github.io/`
Platform: web app and software repo
In chapter: Pattern - Transformation
Description: A pipeline to create beautifully-shaded art from line drawings, trained from a database of anime illustrations.

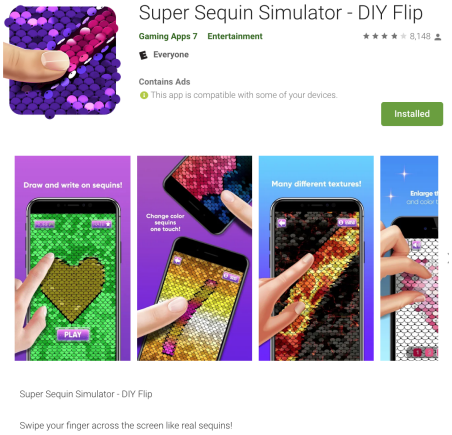


S223: Sublime Text

Creator: Jon Skinner, Will Bond
Url: `https://www.sublimetext.com/`
Year: 2008
In chapter: Casual Programming
Description: A popular text editor with many automation-focused plugins and features.

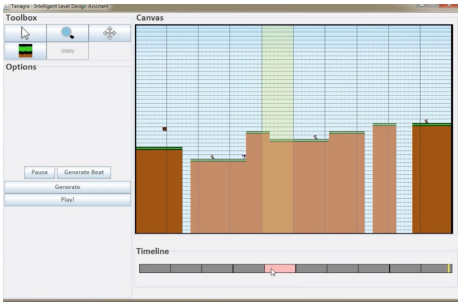
S224: Super Sequin Simulator - DIY Flip

Creator: Gaming Apps 7
Url: `https://play.google.com/store/apps/details?id=ru.gamingapps7.supersequinsimulatordiyflip`
Platform: Android
In chapter: Narrow and Shallow Possibility Spaces
Description: A realistic simulator for drawing on so-called magic sequin fabric.



S225: Tanagra: Mixed-Initiative Level Design for 2D Platformers

Creator: Gillian Smith
Url: `http://sokath.com/main/blog/2011/12/01/tanagra-mixed-initiative-level-design-for-2d-platformers/`
Year: 2011
Platform: research prototype
In chapter: Slow Creators
Description: An academic project to assist users in creating platformer levels that adhere to constraints.



S226: Tarot

Creator: unknown
Year: 18th c
Platform: card deck
In chapter: Existing Frameworks of Generativity
Description: Each card has implicit meanings. When several randomly-selected cards juxtaposed in a reading, they can sketch out a story (for the perceptive reader).

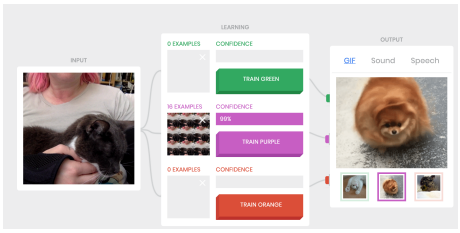


S227: Tay



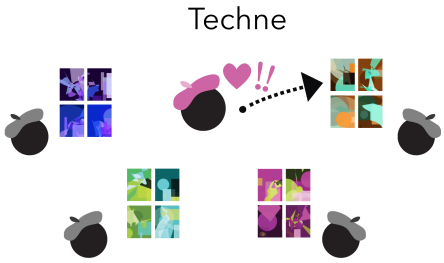
Creator: Microsoft
Url: `https://en.wikipedia.org/wiki/Tay_(bot)`
Year: 2016
Platform: Twitterbot
In chapter: Possibility Spaces
Description: A Twitter-based chatbot that could learn from input (with disastrous results).

S228: Teachable Machine



Creator: Google
Url: `https://teachablemachine.withgoogle.com/`
Year: 2017
Platform: web, Tensorflow
In chapter: A generative framework of generativity
Description: A Google-created tool to show novice users how to train a very simple classifier from webcam data.

S229: Techne



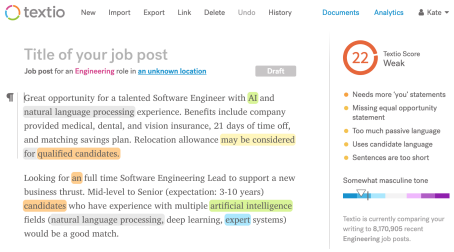
Creator: Kate Compton, Johnathan Pagnutti, and others
Url: `https://archives.nucl.ai/recording/techne-an-art-commune-by-bots-for-bots/`
Year: 2016
Platform: research prototype
In chapters: Tracery and Chancery, Pattern - Entertaining Evaluations
Description: An academic project to develop a community of artbots that could learn from each other.

S230: TEE K. O.



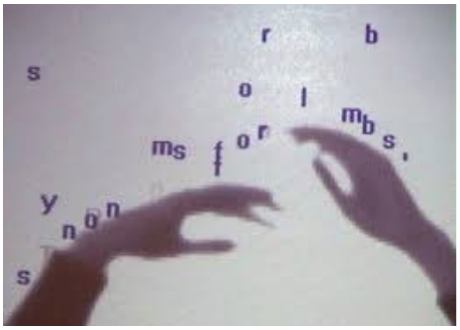
Creator: Jackbox Games
Url: `https://jackboxgames.com/tag/tee-k-o/`
Year: 2016
Platform: console plus phones
In chapter: Pattern - Safety
Description: A party game where players create drawings, captions, and combine them into t-shirt designs to be voted on.

S231: TextIO



Url: `https://textio.com/`
Platform: web
In chapter: Pattern - Entertaining Evaluations
Description: An automated writing annotation tool that analyzes tone and content while you write.

S232: Text Rain



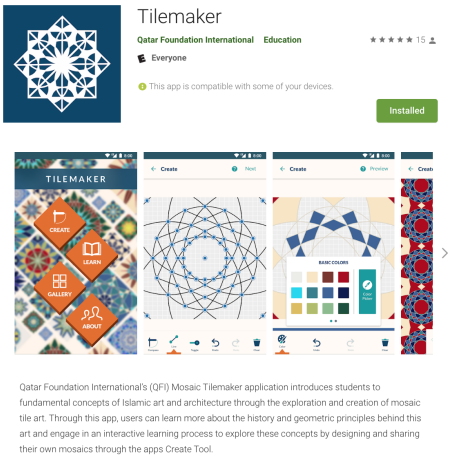
Creator: Romy Achituv, Camille Utterback
Url: `http://camilleutterback.com/projects/text-rain/`
Year: 1999
Platform: installation
In chapter: A generative framework of generativity
Description: Visitors' shadows are used as colliders with particle-based letters dropping from the top of the wall.

S233: Threadsteading



Creator: Gillian Smith, April Grow, Chenxi Liu, Lea Albaugh, Jen Mankoff and Jim McCann
Url: `http://sokath.com/main/blog/2016/12/01/threadsteading/`
Year: 2016
Platform: CNC sewing machine
In chapter: Pattern - Make It Real
Description: A game using a computer-controlled sewing machine to embroider players' moves onto a specially-printed fabric.

S234: Tilemaker



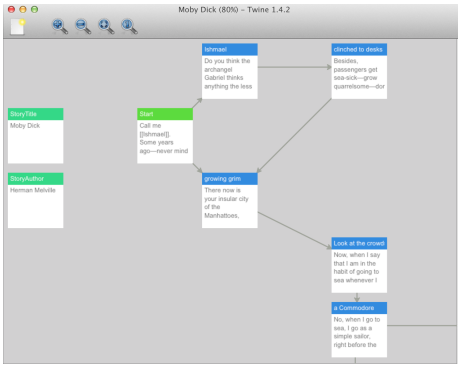
Creator: Qatar Foundation International
Url: `https://play.google.com/store/apps/details?id=com.qfi.tilemaker`
Platform: Android
In chapter: Pattern - Repetition, Repetition, Repetition
Description: A governmented-funded app to explore traditional tile patterns.

S235: Tilt Brush



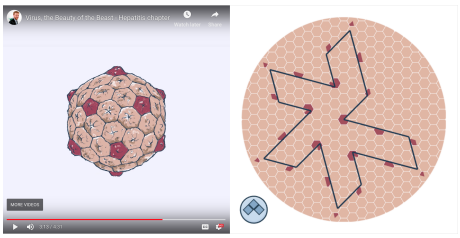
Creator: Google
Url: `https://www.tiltbrush.com/`
Year: 2016
Platform: VR
In chapters: Introduction, Motivation, A generative framework of generativity, Drawing Tools
Description: A stroke-based VR drawing tool

S236: Twine



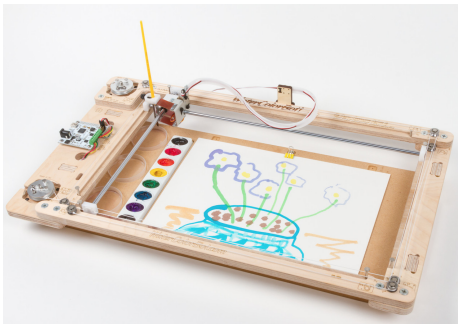
Creator: Chris Klimas
Url: `https://twinery.org/`
Year: 2009
Platform: web
In chapters: Creativity Support Frameworks, Slow Creators
Description: A popular tool to create hypertext narratives, which made the genre accessible to more writers and non-coders.

S237: Virus: the Beauty of the Beast



Creator: Hamish Todd
Url: `http://viruspatterns.com/`
Year: 2017
Platform: youtube video + webapp
In chapter: Narrow and Shallow Possibility Spaces
Description: An unusual combination of casual creator synchronized to a video lecture.

S238: WaterColorBot



Creator: Evil Mad Scientist Laboratories and Super Awesome Sylvia
Url: `https://watercolorbot.com/`
Year: 2013
Platform: CNC controlled watercolor kit
In chapter: Pattern - Make It Real
Description: A robotic arm that can dip into watercolor paints and replicate the pattern drawn by the user of a desktop app.

S239: Weaving

Creator: unknown
Url: `https://en.wikipedia.org/wiki/Weaving`
Year: 27,000 years ago
Platform: physical craft
In chapter: Existing Frameworks of Generativity
Description: An ancient and modern human craft of overlapping thread. There are countless culturally-unique algorithms for it which create emergent aesthetics.

S240: WebGL Lathe Workshop

Creator: Einar Öberg
Url: `http://inear.se/lathe/`
Platform: web
In chapter: Narrow and Shallow Possibility Spaces
Description: A soothing virtual 3D lathe.



S241: Wevva

Creator: Metamakers
Url: `http://www.wevvagame.com/`
Year: 2017
Platform: IOS
In chapters: Slow Creators, Expert evaluation
Description: A casual creator to make and play collision-mechanic games.

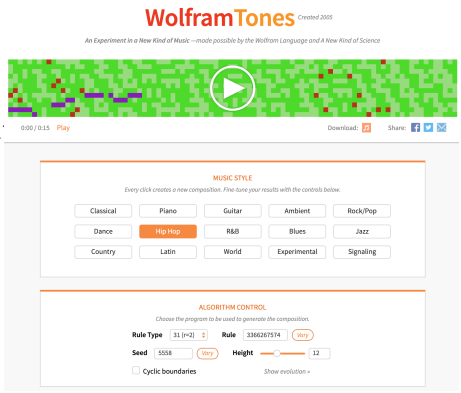


S242: What-Dog.net



Creator: Microsoft
Url: `https://www.what-dog.net/`
Year: 2016
Platform: web
In chapter: Motivation
Description: What breed of dog is that? Upload a photo and Microsoft will try to guess. Works great with non-dog photos.

S243: WolframTones



Creator: Wolfram Technology
Url: `http://tones.wolfram.com/generate/G1LtwWZyBnUE9SXiWc6beeV7qYFl8s0VSesNazi`
Year: 2005
Platform: web
In chapters: Motivation, A generative framework of generativity, Slow Creators
Description: There are many grid-based music tools, this one cellular automata rather than direct user control.

S244: World of Warcraft

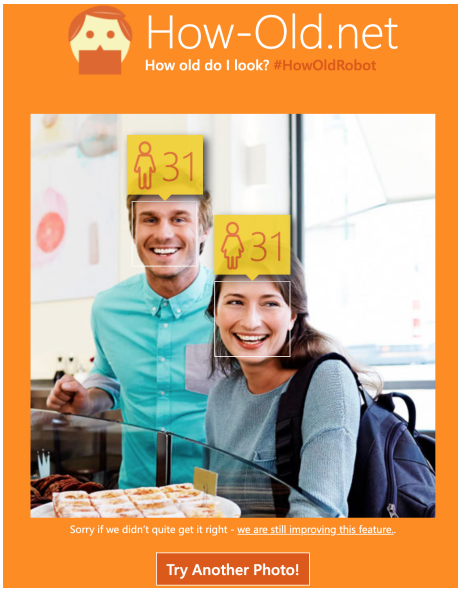
Creator: Blizzard
Url: `https://worldofwarcraft.com/en-us/`
Year: 2004
Platform: Windows, OSX
In chapters: Motivation, Possibility Spaces, Pattern - Expressing Identity
Description: A massively popular MMO, with a very simple character creator.

S245: I Write Like

Creator: Coding Robots
Url: `https://iwl.me/`
Platform: web
In chapter: Motivation
Description: This site purports to tell you which famous author a piece of text is most like. Inaccurate but fun.

S246: How-Old.net

Creator: Microsoft
Url: `https://www.how-old.net/`
Year: 2015
Platform: web
In chapter: Motivation
Description: How old are the people in this photo? This site identifies faces in a photo and guesses their age.



S247: Zentangle

Creator: Zentangle, Inc
Url: `https://zentangle.com/`
Year: 2014
Platform: drawing method
In chapters: Drawing Tools, Pattern - Transformation
Description: A casual yet algorithmic approach to structured doodling.



Bibliography

[1] DEEP DREAM GENERATOR. (https://deepdreamgenerator.com/).

[2] List of Magic slang. (https://mtg.gamepedia.com/List_of_Magic_slang).

[3] Syntactic Sugar. (https://en.wikipedia.org/wiki/Syntactic_sugar).

[4] "Reviews (Pinball Construction Set and Music Construction Set)". *Ahoy*, 5, 1984. (https://archive.org/stream/Ahoy_Issue_05_1984-05_Ion_International_US#page/n51/mode/2up).

[5] 5ive Bullet Games. Tracery. (https://www.assetstore.unity3d.com/en/#!/content/100911).

[6] Gregory D Abowd and Alan J Dix. Giving undo attention. *Interacting with Computers*, 4(3):317–342, 1992.

[7] Fabien Accominotti. Creativity from interaction: Artistic movements and the creativity careers of modern painters. *Poetics*, 37(3):267–294, 2009.

[8] Benzi Ahamed. (Swift)Tracery: Powerful extensible content generation library inspired by Tracery.io. (https://github.com/BenziAhamed/Tracery).

[9] Derek Ahmedzai. Source for trumptaxbot. (https://cheapbotsdonequick.com/source/trumptaxbot).

[10] Lea Albaugh, April Grow, Chenxi Liu, James McCann, Gillian Smith, and Jennifer Mankoff. Threadsteading: Playful Interaction for Textile Fabrication Devices. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 285–288. ACM, 2016.

[11] JL Alty. Navigating through compositional space: The creativity corridor. *Leonardo*, 28(3):215–219, 1995.

[12] Teresa M Amabile, Regina Conti, Heather Coon, Jeffrey Lazenby, and Michael Herron. Assessing the work environment for creativity. *Academy of management journal*, 39(5):1154–1184, 1996.

[13] Joceline Andersen. Now you've got the shiveries: Affect, intimacy, and the ASMR whisper community. *Television & New Media*, 16(8):683–700, 2015.

[14] Sebastian Anthony. 3D printing: a replicator and teleporter in every home. (`https://www.extremetech.com/extreme/92042-3d-printing-a-replicator-and-teleporter-in-every-home`) Aug 5, 2011.

[15] Chris Argyris. Theories of action that inhibit individual learning. *American Psychologist*, 31(9):638, 1976.

[16] Melinos Averkiou, Vladimir G Kim, Youyi Zheng, and Niloy J Mitra. Shapesynth: Parameterizing model collections for coupled shape exploration and synthesis. In *Computer Graphics Forum*, volume 33, pages 125–134. Wiley Online Library, 2014.

[17] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[18] Dustin Bailey. The artist behind this Minecraft sculpture makes a living building stuff for the game. (`https://www.pcgamesn.com/minecraft/minecraft-sculpture`).

[19] Almas Baimagambetov. Grammy (This is a JVM port of Kate Compton's story-grammar generator - Tracery). (`https://github.com/AlmasB/grammy`).

[20] Philip Ball and Neil R Borley. *The self-made tapestry: pattern formation in nature*, volume 198. Oxford University Press Oxford, 1999.

[21] Matthew R.F. Balousek. Twinecery: A tool for authoring and implementing tracery grammars in Twine. (`https://github.com/mrfb/twinecery`).

[22] Francesco Barbieri, Francesco Ronzano, and Horacio Saggion. What does this Emoji Mean? A Vector Space Skip-Gram Model for Twitter Emojis. In *LREC*, 2016.

[23] Pippin Barr. It is as if you were playing chess, 2016. (`http://www.pippinbarr.com/games/itisasifyouwereplayingchess/`).

[24] Robert Barry. Emojis Are Art in This Twitter Bot ASCII Gallery. (https://motherboard.vice.com/en_us/article/aekvqp/tiny-gallery-emojis-are-art-in-this-twitter-bot-ascii-gallery) Feb 16, 2016.

[25] Richard Bartle. Hearts, Clubs, Diamonds, Spades: Players Who Suit Muds, 1996. (https://mud.co.uk/richard/hcds.htm).

[26] Joe Baxter-Webb. A Strange Voyager's Earliest Voyages. (https://joebaxterwebb.wordpress.com/2017/01/23/a-strange-voyagers-guide-to-starting-a-twitter-prose-bot/) Jan 23, 2017.

[27] Suzanne Bearne. Art and wine: Learning to paint while drinking. (https://www.bbc.com/news/business-42251261).

[28] Matt Bell. Creative projects. (https://mattishere.wordpress.com/creative-projects/).

[29] Mathias Benedek, Tanja Könen, and Aljoscha C Neubauer. Associative abilities underlying creativity. *Psychology of Aesthetics, Creativity, and the Arts*, 6(3):273, 2012.

[30] Jon Louis Bentley. Little languages. *Commun. ACM*, 29(8):711–721, 1986.

[31] Daniel E Berlyne. Novelty, complexity, and hedonic value. *Perception & Psychophysics*, 8(5):279–286, 1970.

[32] Srinjita Bhaduri, Jesús G Ortiz Tovar, and Shaun K Kane. Fabrication Games: Using 3D Printers to Explore New Interactions for Tabletop Games. In *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition*, pages 51–62. ACM, 2017.

[33] Noah Blier. Stories of AI Failure and How to Avoid Similar AI Fails in 2019. (https://www.lexalytics.com/lexablog/stories-ai-failure-avoid-ai-fails-2019) Jan 3, 2019.

[34] Margaret Boden. Computer Models of Creativity. *The Cambridge handbook of creativity*, pages 351–372, 1999.

[35] Margaret A Boden and Ernest A Edmonds. What is generative art? *Digital Creativity*, 20(1-2):21–46, 2009.

[36] Ian Bogost. *Unit operations: An approach to videogame criticism*. MIT press, 2008.

[37] Brian Boucher. Visit What May Be the Smallest Gallery in the World. (https://news.artnet.com/market/tiny-gallery-emoji-artworks-emma-winston-434894) Feb 26, 2016.

[38] Eli Brody. Tracery: a story-grammar generation library for Ruby. (https://github.com/elib/tracery).

[39] Cameron Browne and Frederic Maire. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):1–16, 2010.

[40] Jerome Seymour Bruner. *On knowing: Essays for the left hand*. Harvard University Press, 1979.

[41] George Buckenham. A port of Kate Compton's Tracery to C#/Unity3d. (https://github.com/v21/unity-tracery).

[42] George Buckenham. personal correspondence with George Buckenham, Jan 3, 2018. 2018.

[43] George Buckenham. Tracery: a story-grammar generation library for javascript. (https://github.com/v21/tracery).

[44] Glen Bull, Hossein Haj-Hariri, Rosa Atkins, and Pam Moran. An educational framework for digital manufacturing in schools. *3D Printing and Additive Manufacturing*, 2(2):42–49, 2015.

[45] Angelika C Bullinger, Anne-Katrin Neyer, Matthias Rass, and Kathrin M Moeslein. Community-based innovation contests: Where competition meets cooperation. *Creativity and innovation management*, 19(3):290–303, 2010.

[46] DL Cade. Adobe's Project 'Deep Fill' is an incredible, AI-powered Content Aware Fill. (https://www.dpreview.com/news/2758068086/adobe-s-project-deep-fill-is-an-incredible-ai-powered-content-aware-fill) Oct 21, 2017.

[47] Roger Caillois. *Man, play, and games*. University of Illinois Press, 2001.

[48] Italo Calvino. Prose and Anticombinatorics, 1986.

[49] Nicky Case. Loopy. (http://ncase.me/loopy/).

[50] Nicky Case. Neurotic Neurons. (http://ncase.me/neurons/).

[51] Nicky Case. Simulating the World in Emoji. (http://ncase.me/simulating/).

[52] Nicky Case. The Wisdom and/or Madness of Crowds. (http://ncase.me/crowds/).

[53] Nicky Case and Vi Hart. Parable of the Polygons. (http://ncase.me/polygons/).

[54] Tim Challies. ""No Man's Sky" and 10,000 Bowls of Plain Oatmeal". (https://www.challies.com/articles/no-mans-sky-and-10000-bowls-of-plain-oatmeal/).

[55] Alex J Champandard. Deep Forger. (https://deepforger.com/).

[56] Alex J Champandard. Neural Doodles: Workflows for the Next Generation of Artists. accessed at https://web.archive.org/web/20160503202056/https://nucl.ai/blog/neural-doodles/.

[57] Alex J Champandard. Semantic style transfer and turning two-bit doodles into fine artworks. *arXiv preprint arXiv:1603.01768*, 2016.

[58] David Chan. The philosophy of The Sims. Technical report, Stanford University Working paper, http://library. stanford. edu/depts/hasrg . . . , 2003.

[59] Jenova Chen. Flow in games (and everything else). *Communications of the ACM*, 50(4):31–34, 2007.

[60] Winston S Churchill. *Painting as a Pastime*. Rosetta Books, 1965.

[61] Travis Clark. The 11 most popular gaming YouTube stars — some of whom made over $10 million last year. (https://www.businessinsider.com/most-popular-gaming-youtube-stars-ranked-2018-2).

[62] Simon Colton, Jakob Halskov, Dan Ventura, Ian Gouldstone, Michael Cook, and Blanca Pérez Ferrer. The Painting Fool Sees! New Projects with the Automated Painter. In *ICCC*, pages 189–196, 2015.

[63] Simon Colton, Mark Nelson, Edward Powley, Swen Gaudl, Rob Saunders, Blanca Perez Ferrer, Peter Ivey, and Michael Cook. A Parameter-Space Design Methodology for Casual Creators. *Proceedings of International Conference on Computational Creativity 2018*, pages 264–271, 2018.

[64] Slashdot community (simoniker). Universities Step Up Videogame Studies. (https://games.slashdot.org/story/03/11/13/2017200/universities-step-up-videogame-studies) Nov 13, 2003.

[65] Kate Compton. Chancery. (http://www.tracery.io/chancery/).

[66] Kate Compton. (JS Fiddle version of the Roman Comedy Generator that became Tracery). (https://jsfiddle.net/galaxykate/SECmW/).

[67] Kate Compton. PaintThrush. (https://galaxykate.itch.io/paintthrush).

[68] Kate Compton. Scribbles. (http://www.galaxykate.com/apps/Prototypes/Scribbles/).

[69] Kate Compton. Tracery dev editor. (http://brightspiral.com/dev/) Oct 25, 2018.

[70] Kate Compton. Tracery dev editor and visualization experiment. (http://tracery.io/editor/).

[71] Kate Compton. Tracery (Tech Toolbox GDC 2016). (https://www.gdcvault.com/play/1023377/Tech).

[72] Kate Compton. Let It Grow: Practical Procedural Generation From The Ground Up, 2015. PROCJAM 2015.

[73] Kate Compton. So You Want To Build a Generator, 2016. (http://galaxykate0.tumblr.com/post/139774965871/so-you-want-to-build-a-generator).

[74] Kate Compton. Bot Poetics, 2017. (https://vimeo.com/225566776).

[75] Kate Compton. Encyclopedia of Generativity (Zine), 2017. (http://www.galaxykate.com/pdfs/galaxykate-zine-encyclopediaofgenerativity.pdf).

[76] Kate Compton. generative art toys. In Tanya X Short and Tarn Adams, editors, *Procedural Generation in Game Design*. AK Peters/CRC Press, 2017.

[77] Kate Compton. https://www.youtube.com/watch?v=3RwRv_CIK4A, 2017. ASYNC.

[78] Kate Compton. Practical Procedural Generation for Everyone, 2017. (https://www.youtube.com/watch?v=WumyfLEa6bU).

[79] Kate Compton. Oops, I made an open-source art tool! (Zine), 2018. (http://www.galaxykate.com/pdfs/galaxykate-zine-opensourcetoolsforthearts.pdf).

[80] Kate Compton, Benjamin Filstrup, et al. Tracery: Approachable story grammar authoring for casual users. In *Seventh Intelligent Narrative Technologies Workshop*, 2014.

[81] Kate Compton, James Grieve, Ed Goldman, Ocean Quigley, Christian Stratton, Eric Todd, and Andrew Willmott. Creating spherical worlds. In *SIGGRAPH Sketches*, page 82, 2007.

[82] Kate Compton, Ben Kybartas, and Michael Mateas. Tracery: an author-focused generative text tool. In *International Conference on Interactive Digital Storytelling*, pages 154–161. Springer, 2015.

[83] Kate Compton, Heather Logas, Joseph C Osborn, Chandranil Chakrabortti, Kelsey Coffman, Daniel Fava, Dylan Lederle-Ensign, Zhongpeng Lin, Jo Mazeika, Afshin Mobramaein, et al. Design Lessons From Binary Fission: A Crowd Sourced Game for Precondition Discovery. In *DiGRA/FDG*, 2016.

[84] Kate Compton and Michael Mateas. Procedural Level Design for Platform Games. pages 109–111, 2006.

[85] Kate Compton and Michael Mateas. A Different Kind of Physics: Interactive evolution of expressive dancers and choreography. *Computational Creativity and Games Workshop, ICCC*, 2015.

[86] Kate Compton and Michael Mateas. A generative framework of generativity. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[87] Kate Compton and Michael Mateas. Freedom of Movement: Generative Responses to Motion Control. *Experimental AI in Games, AIIDE*, 2018.

[88] Kate Compton, Edward Melcer, and Michael Mateas. Generominos: Ideation Cards for Interactive Generativity. *Experimental AI in Games Workshop, AIIDE*, 2017.

[89] Kate Compton, Joseph C Osborn, and Michael Mateas. Generative methods. In *The Fourth Procedural Content Generation in Games workshop, PCG*, volume 1, 2013.

[90] Reddit contributor (account deleted). 18 Quintillion Planets? (`https://www.reddit.com/r/NoMansSkyTheGame/comments/3femfv/18_quintillion_planets/`).

[91] Reddit contributor (account deleted). Just got Legend Viewer, blew my mind. (`https://www.reddit.com/r/dwarffortress/comments/3z2nzq/just_got_legend_viewer_blew_my_mind/`).

[92] Reddit contributor (JoinTheBattle). The Matrix used multiple sets of twins in the scene with the woman in red to create the feeling of a copy and paste world. (`https://www.reddit.com/r/MovieDetails/comments/6kgaph/the_matrix_used_multiple_sets_of_twins_in_the/`).

[93] Fandom contributors. Celestial Brush techniques. (https://okami.fandom.com/wiki/Celestial_Brush_techniques).

[94] Fandom contributors. Powder Game Timeline. (https://danball.fandom.com/wiki/Powder_Game_Timeline).

[95] Fandom contributors. Spore Bot Parts Pack. (https://spore.fandom.com/wiki/Spore_Bot_Parts_Pack).

[96] Fandom contributors. Spore Creepy & Cute. (https://spore.fandom.com/wiki/Spore_Creepy_%26_Cute).

[97] Gamepedia contributors. Lamb's Classification Method. (https://nomanssky.gamepedia.com/Lamb%27s_Classification_Method).

[98] Reddit contributors. For Sharing Awesome Rimworld Seeds. (https://www.reddit.com/r/RimWorldSeeds/).

[99] Reddit contributors. Reddit - paintcomposer. (https://www.reddit.com/r/paintcomposer/).

[100] Reddit contributors. Sugar Plum Fairy [Line Rider] [100 Line Challenge contest] (Changed the ending. (https://www.reddit.com/r/linerider/comments/8k5qv5/sugar_plum_fairy_line_rider_100_line_challenge/).

[101] Know Your Meme contributors (Don and Y.F.). Edges2Cats. (https://knowyourmeme.com/memes/sites/edges2cats).

[102] Michael Cook. Alien Languages: How We Talk About Procedural Generation. (https://www.gamasutra.com/blogs/MichaelCook/20160825/279938/Alien_Languages_How_We_Talk_About_Procedural_Generation.php) Aug 25, 2016.

[103] Michael Cook. Danesh. (http://www.danesh.procjam.com/).

[104] Michael Cook. Seeds: The PROCJAM Zine. (http://www.procjam.com/seeds/).

[105] Michael Cook, Jeremy Gow, and Simon Colton. Danesh: Helping bridge the gap between procedural generators and their output. In *"Proceedings of the 7th Workshop on Procedural Content Generation"*, 2016.

[106] Ronny Cook, Judy Kay, Greg Ryan, and Richard C Thomas. A toolkit for appraising the long-term usability of a text editor. *Software Quality Journal*, 4(2):131–154, 1995.

[107] National Research Council et al. *Beyond productivity: Information technology, innovation, and creativity*. National Academies Press, 2003.

[108] Matt Cox. Occupy White Walls is an enthralling virtual gallery MMO. (`https://www.rockpapershotgun.com/2018/04/17/occupy-white-walls-is-an-enthralling-virtual-gallery-mmo/`) April 17, 2018.

[109] Chris Crawford. The art of computer game design. 1984.

[110] Mihaly Csikszentmihalyi. Flow and the psychology of discovery and invention. *HarperPerennial, New York*, 39, 1997.

[111] Mihaly Csikszentmihalyi. *Flow and the psychology of discovery and invention*. HarperPerennial, New York, 1997.

[112] Joao M Cunha, Pedro Martins, and Penousal Machado. How Shell and Horn make a Unicorn: Experimenting with Visual Blending in Emoji. *Proceedings of International Conference on Computational Creativity 2018*, pages 145–152, 2015.

[113] Marcel Danesi. *The semiotics of emoji: The rise of visual language in the age of the internet*. Bloomsbury Publishing, 2016.

[114] Christopher M Danforth. Chaos in an Atmosphere Hanging on a Wall. *Mathematics of Planet Earth*, 17, 2013.

[115] Jennifer Daniel. Neural Network-Generated Illustrations in Allo. *Google AI Blog*. (`https://ai.googleblog.com/2017/05/neural-network-generated-illustrations.html`) 2017.

[116] Rudolph P Darken and Barry Peterson. Spatial orientation, wayfinding, and representation., 2002.

[117] Rudolph P Darken and John L Sibert. Wayfinding strategies and behaviors in large virtual worlds. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 142–149. ACM, 1996.

[118] Whitney Davis. The origins of image making. *Current Anthropology*, 27(3):193–215, 1986.

[119] Jos de Jong. A web-based tool to view, edit, format, and validate JSON. (`https://github.com/josdejong/jsoneditor`).

[120] Josh Dean. Is This the World's Most Creative Manufacturer? (`https://www.inc.com/magazine/201310/josh-dean/is-quirky-the-worlds-most-creative-manufacturer.html`) 2013.

[121] Serin Delaunay. Cheap Markovs, Traced Quick!, 2018. (https://github.com/serin-delaunay/cheapmarkovstracedquick).

[122] Alex Donaldson. From Thanos to Skeletor, here's some of the best SoulCalibur 6 character creations. (https://www.vg247.com/2018/10/23/soulcalibur-6-character-creation-best/) Oct 23, 2018.

[123] Alan Dorin, Jonathan McCabe, Jon McCormack, Gordon Monro, and Mitchell Whitelaw. A framework for understanding generative art. *Digital Creativity*, 23(3-4):239–259, 2012.

[124] Joris Dormans. Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the 2010 workshop on procedural content generation in games*, page 1. ACM, 2010.

[125] Joris Dormans. Simulating Mechanics to Study Emergence in Games. *Artificial Intelligence in the Game Design Process*, 2(6.2):5–2, 2011.

[126] Luke Dormehl. Not your average table: Genetic algorithms and 3D printing set this furniture apart. (https://www.digitaltrends.com/cool-tech/genetic-algorithm-rio-table-chairs/) Sept 30, 2016.

[127] Dale Dougherty. The maker movement. *Innovations: Technology, Governance, Globalization*, 7(3):11–14, 2012.

[128] Paul Dourish. *Where the action is: the foundations of embodied interaction*. MIT press, 2004.

[129] Scott Draves. Electric Sheep (instructions.rtf with software), 1999.

[130] Theresa (MaxisCactus) Duringer. Holiday Mystery Giveaway #2 - Real Monster Fruitcake by MaxisKate (pics) . (http://forum.spore.com/jforum/posts/list/47291.page).

[131] Ciro Durán. Tracery: a story-grammar generation library, ported to Haxe. (https://github.com/chiguire/traceryhx).

[132] Alistair DN Edwards. Visual programming languages: the next generation. *ACM SIGPLAN Notices*, 23(4):43–50, 1988.

[133] W Keith Edwards, Takeo Igarashi, Anthony LaMarca, and Elizabeth D Mynatt. A temporal model for multi-level undo and redo. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 31–40. ACM, 2000.

[134] Ryan Feddersen. Coyote Now, 2018. Installation artwork.

[135] FinchBeak. Petalz. (https://web.archive.org/web/20160322111055/http://finchbeak.com/index.php/petalz-game/).

[136] Bradley J Fisher and Diana K Specht. Successful aging and creativity in later life. *Journal of aging studies*, 13(4):457–472, 1999.

[137] Tesca Fitzgerald, A Thomaz, and A Goel. Human-Robot Co-Creativity: Task Transfer on a Spectrum of Similarity. In *Proceedings of Seventh International Conference on Computational Creativity, Atlanta*, 2017.

[138] Steven Fortune. Voronoi diagrams and Delaunay triangulations. In *Computing in Euclidean geometry*, pages 225–265. World Scientific, 1995.

[139] Bay 12 forum user (Cagbee). What is the most epic fortress fail you had??? (http://www.bay12forums.com/smf/index.php?topic=152039.0).

[140] Mark Frauenfelder. "Amazing illustrations from a doodle pad printed with a partially completed drawing of a naked woman". (https://boingboing.net/2017/09/26/amazing-illustrations-from-a.html) Sept 26, 2012.

[141] Lindsay Friedman. Coloring Books for Adults Are Good for Creativity, and Selling in the Millions. (https://www.entrepreneur.com/article/254682) Jan 10, 2016.

[142] Nathan Friend. Inspirograph. (https://nathanfriend.io/inspirograph/).

[143] GameGirlAdvance. Click on this most delicious burger! (https://www.gamegirladvance.com/2002/09/click-on-this-most-delicious-burger.html) Sept 16, 2002.

[144] Evolutionary Games. (https://steamcommunity.com/app/249610).

[145] Yoyo Games. Gamemaker. (https://www.yoyogames.com/gamemaker).

[146] Jacob Garbe. Simulation of History and Recursive Narrative Scaffolding. (http://project.jacobgarbe.com/simulation-of-history-and-recursive-narrative-scaffolding/).

[147] Martin Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223(4):120–123, 1970.

[148] John Gaudiosi. 'The Sims' Creator Will Wright Adds Interactivity to 'Bar Karma' TV Series. (https://www.hollywoodreporter.com/news/sims-creator-wright-adds-interactivity-76475).

[149] Paul Gauguin. Paul Gauguin to Vincent van Gogh. Paris, on or about Thursday, 20 March 1890. (http://www.vangoghletters.org/vg/letters/let859/letter.html).

[150] David Gauntlett. *Making is connecting*. John Wiley & Sons, 2013.

[151] William W Gaver, Andrew Boucher, Sarah Pennington, and Brendan Walker. Cultural probes and the value of uncertainty. *interactions*, 11(5):53–56, 2004.

[152] Geraldine Gay and Helene Hembrooke. *Activity-centered design: An ecological approach to designing smart tools and usable systems*. Mit Press, 2004.

[153] Boardgame Geek. KeyForge: Call of the Archons (2018). (https://boardgamegeek.com/boardgame/257501/keyforge-call-archons).

[154] Steven M Gelber. *Hobbies: Leisure and the culture of work in America*. Columbia University Press, 1999.

[155] Johan Gielis. A generic geometric transformation that unifies a wide range of natural and abstract shapes. *American journal of botany*, 90(3):333–338, 2003.

[156] Harry Josephine Giles. #NABOMAMO: THE FIRST 15 BOTS. (https://harrygiles.org/2016/11/15/nabomamo-the-first-15-bots/).

[157] Chaim Gingold. Miniature Gardens & Magic Crayons: Games, Spaces & Worlds. Master's thesis, School of Literature, Communication, and Culture, Georgia Institute of Technology, 2003.

[158] Chaim Gingold, 2007. ("https://www.gdcvault.com/play/527/SPORE-s-Magic").

[159] GitHub. The State of the Octoverse. (https://octoverse.github.com/).

[160] Google. Oskar Fischinger's 117th Birthday. (https://www.google.com/logos/2017/fischinger/fischinger17.html) June 22, 2017.

[161] Google, Kate Compton, and Calvin Ashmore. Bottery: A conversational agent prototyping platform by Kate Compton. (https://github.com/google/bottery).

[162] Robert F Gordon, George B Leeman Jr, and Clayton H Lewis. Concepts and implications of undo for interactive recovery. In *Proceedings of the 1985 ACM annual conference on The range of computing: mid-80's perspective: mid-80's perspective*, pages 150–157. ACM, 1985.

[163] Emily Gosling. Hey Froshmin, Welcome to Art Sqool! (https://eyeondesign.aiga.org/hey-froshmin-welcome-to-art-sqool/) Oct 25, 2018.

[164] Harrison G Gough. Studying creativity by means of word association tests. *Journal of Applied Psychology*, 61(3):348, 1976.

[165] Jan Goyvaerts. RegexMagic: Regular Expression Generator. (https://www.regular-expressions.info/regexmagic.html).

[166] Fritz Graf. Rolling the Dice for an Answer. *Mantike: Studies in Ancient Divination*, 51:98, 2005.

[167] Joanna Graham and Kefin Mahon. The Most Bizarre Fan-Made Pro Wrestlers From WWE 2K16. (https://www.kotaku.com.au/2016/10/the-most-bizarre-fan-made-pro-wrestlers-from-wwe-2k16/).

[168] Roy Graham. The World's Most Complex Problems, Now In Emoji. (https://killscreen.com/articles/the-worlds-most-complex-problems-now-in-emoji/) Jan 08, 2016.

[169] Nathan Grayson. Rust's Controversial Random Gender Update Has Led To More Players. (https://kotaku.com/rusts-controversial-random-gender-update-has-led-to-mor-1772390030) April 22, 2016.

[170] Joseph L Griffith. Drawing Machine 1, 2009. Installation artwork.

[171] Joseph L Griffith. ADA, 2010. Installation artwork.

[172] Mark Gritter. Mark's Meta Artbot. (http://artbot.combinatorium.com/#!/tracery) 2017.

[173] April Grow, Melanie Dickinson, Johnathan Pagnutti, Noah Wardrip-Fruin, and Michael Mateas. Crafting in Games. *DIGITAL HUMANITIES QUARTERLY*, 11(4), 2017.

[174] Milan Gruner. Lua port of Tracery. (https://github.com/lemilonkh/moontrace).

[175] Raiford Guins and Henry Lowood. *Debugging game history: a critical lexicon*. MIT Press, 2016.

[176] Beau Gunderson. Tracery Writer. (https://beaugunderson.com/tracery-writer/).

[177] Thu-Huong Ha and Nikhil Sonnad. How do you draw a circle? We analyzed 100,000 drawings to show how culture shapes our instincts. (https://qz.com/994486/the-way-you-draw-circles-says-a-lot-about-you/) June 15, 2017.

[178] Patrick Hackett and Double Fine Productions. Augmented Imagination: Exploiting Kinect for Double Fine Happy Action Theater. (https://www.gdcvault.com/play/1017901/Augmented-Imagination-Exploiting-Kinect-for).

[179] Christopher Hadnagy. *Social engineering: The art of human hacking*. John Wiley & Sons, 2010.

[180] Mark Hamilton, Chris Hoder, Matthew Ritchie, and Sarah Schwettmann. Gen Studio. (https://gen.studio/).

[181] Sarah Harmon. FIGURE8: A Novel System for Generating and Evaluating Figurative Language. In *ICCC*, pages 71–77, 2015.

[182] Sonia Harris. Swearing as a meditation. (https://medium.com/\spacefactor\@m{}soniaharris/swearing-as-a-meditation-ea9524aac7ba) Sept 26, 2017.

[183] Ben (aka Rabid Squirrel) Harvey. The Line Rider Archival Project. (http://www.benjaminharveydesign.com/line-rider-archival-project/).

[184] Marc Hassenzahl. User experience (UX): towards an experiential perspective on product quality. In *Proceedings of the 20th Conference on l'Interaction Homme-Machine*, pages 11–15. ACM, 2008.

[185] Erin Jonathan Hastings, Ratan K Guha, and Kenneth O Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.

[186] Nicole He. The Best Art. (https://the-best-art.computer/) 2017.

[187] Douglas Heaven. When infinity gets boring: What went wrong with No Man's Sky. (https://www.newscientist.com/article/2104873-when-infinity-gets-boring-what-went-wrong-with-no-mans-sky/) Sept 6, 2016.

[188] Chris Hecker, Bernd Raabe, Ryan W Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. In *ACM Transactions on Graphics (TOG)*, volume 27, page 27. ACM, 2008.

[189] Paul Heisig (themroc). EDDB - Elite: Dangerous Database, 2019. (https://eddb.io/).

[190] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1, 2013.

[191] Barbara Herman. 'Secret Garden' Coloring Book Outsells Harper Lee As Adults Seek 'Digital Detox'. (https://www.ibtimes.com/secret-garden-coloring-book-outsells-harper-lee-adults-seek-digital-detox-1863678) Mar 3 2015.

[192] Christopher Hesse. Image-to-Image Demo: Interactive Image Translation with pix2pix-tensorflow. (https://affinelayer.com/pixsrv/) Feb 19, 2017.

[193] Craig Hickman. Kid pix: The early years, 2013. (http://red-green-blue.com/kid-pix-the-early-years/).

[194] Robin K Hill and Mark Guzdial. Manipulating word representations, and preparing students for coding jobs? *Communications of the ACM*, 60(10):12–13, 2017.

[195] David Hirmes and Beckman Joshua. Erasures. (https://erasures.wavepoetry.com/).

[196] Gregory Hornby, Al Globus, Derek Linden, and Jason Lohn. Automated antenna design with evolutionary algorithms. In *Space 2006*, page 7242. 2006.

[197] Gregory Hornby, Al Globus, Derek Linden, and Jason Lohn. Automated antenna design with evolutionary algorithms. In *Space 2006*, page 7242. 2006.

[198] Jurie Horneman. Choba-tracery: A Tracery plugin for Choba. (https://github.com/jhorneman/choba-tracery).

[199] Ian Horswill and Ethan Robison. What's the Worst Thing You've Ever Done at a Conference? Operationalizing Dread's Questionnaire Mechanic. *Experimental AI in Games Workshop*, 2018.

[200] Matthew Hughes. "Let's face it: The best part of RollerCoaster Tycoon was killing tourists". (https://thenextweb.com/gaming/2017/09/29/lets-face-it-the-best-part-of-rollercoaster-tycoon-was-killing-tourists/) Sept 29, 2017.

[201] Johan Huizinga. The nature of play. *Philosophic Inquiry in Sport. Champaign: Human Kinetics*, 1995.

[202] Robin Hunicke, Marc LeBlanc, and Robert Zubek. MDA: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, volume 4, page 1722, 2004.

[203] Edwin Hutchins. Cognitive ecology. *Topics in cognitive science*, 2(4):705–715, 2010.

[204] Edwin L Hutchins, James D Hollan, and Donald A Norman. Direct manipulation interfaces. *Human-computer interaction*, 1(4):311–338, 1985.

[205] IMVU. Earn Money by Creating! Creators can earn real cash! (`https://help.imvu.com/s/article/Earn-Money-by-Creating`) Nov 16, 2018.

[206] Katherine Isbister. Game Heroine Invades Bucolic Needlepoint Landscapes; Wreaks Aesthetic Havoc. (`https://www.gamegirladvance.com/2003/08/game-heroine-invades-bucolic-needlepoint-landscapes-wreaks-aesthetic-havoc.html`) Aug 12, 2003.

[207] Atil Iscen, Adrian Agogino, Vytas SunSpiral, and Kagan Tumer. Controlling tensegrity robots through evolution. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1293–1300. ACM, 2013.

[208] Gita Jackson. The Story Of Dwarf Fortress' Loneliest Necromancer. (`https://kotaku.com/the-story-of-dwarf-fortress-loneliest-necromancer-1821481046`) Dec 20, 2017.

[209] Lucy James. History Of The Sims: How A Major Franchise Evolved From City-Builder To Life-Simulator. (`https://www.gamespot.com/gallery/history-of-the-sims-how-a-major-franchise-evolved-/2900-1623/5/`) Nov 19, 2017.

[210] Keith Johnstone. *Impro for storytellers*. Faber and Faber, 1999.

[211] Daniel Jones, Andrew R Brown, and Mark d'Inverno. The extended composer. In *Computers and Creativity*, pages 175–203. Springer, 2012.

[212] Sussane Jul and George W Furnas. Navigation in electronic worlds: a CHI 97 workshop. *SIGCHI bulletin*, 29:44–49, 1997.

[213] Matt Kamen. From Blocks To Bucks: How An 18-year-old Gamer Built a Business Inside Minecraft. (`https://www.fandom.com/articles/building-a-business-on-minecraft`) Dec 17, 2018.

[214] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design guidelines for domain specific languages. *arXiv preprint arXiv:1409.2378*, 2014.

[215] Isaac Karth and Adam M Smith. WaveFunctionCollapse is constraint solving in the wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, page 68. ACM, 2017.

[216] James C Kaufman and Ronald A Beghetto. Beyond big and little: The four c model of creativity. *Review of general psychology*, 13(1):1, 2009.

[217] Paul Kearney. Cognitive Callisthenics: Do FPS computer games enhance the player's cognitive abilities? *Proceedings of Digital Games Research Conference (DIGRA) 2005*, 2005.

[218] Michael Keller. FreeCell – Frequently Asked Questions (FAQ). (`http://www.solitairelaboratory.com/fcfaq.html`).

[219] Ahmed Khalifa, Gabriella AB Barros, and Julian Togelius. DeepTingle. *arXiv preprint arXiv:1705.03557*, 2017.

[220] David Kirsh. Embodied cognition and the magical future of interaction design. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 20(1):3, 2013.

[221] Michele Knobel, Colin Lankshear, et al. Cut, paste, publish: The production and consumption of zines. *Adolescents and literacies in a digital world*, pages 164–185, 2002.

[222] Donald E Knuth. Semantics of context-free languages. *Mathematical systems theory*, 2(2):127–145, 1968.

[223] Andy J. Ko. Designing and evaluating programming languages: Dagstuhl trip report. (`https://medium.com/bits-and-behavior/designing-learnable-teachable-and-productive-programming-languages-dagstuhl-trip-report-81e41bde84bd`) Feb 9, 2018.

[224] Kenneth Kolson. The Politics of City Planning Simulations. *Annual Meeting of the American Political Science Association*, 1994.

[225] Bernd Kreimeier. The case for game design patterns, 2002. (`https://www.gamasutra.com/view/feature/132649/the_case_for_game_design_patterns.php`).

[226] Max Kreminski. blackout poetry generator. (`https://mkremins.github.io/blackout/`).

[227] Max Kreminski. TracerySharp: C#/Unity port of Tracery (heavily WIP). (https://github.com/mkremins/TracerySharp).

[228] Tore Kristensen. The physical context of creativity. *Creativity and innovation management*, 13(2):89–96, 2004.

[229] Cameron Kunzelman. 'Art Sqool' Is a Game About Following Your Passion in an Exploitative World. (https://waypoint.vice.com/en_us/article/evev9z/scam-arts-education-art-sqool) Feb 8, 2019.

[230] Yayoi Kusama. The Obliteration Room, 2011. Installation artwork.

[231] Evil Mad Scientist Labs. StippleGen. (https://wiki.evilmadscientist.com/StippleGen).

[232] Karim R Lakhani and Robert G Wolf. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. *Perspectives on Free and Open Source Software*, 2003.

[233] M. Lazer-Walker. Native desktop app for experimenting with Tracery grammars. (https://github.com/lazerwalker/tracery-dot-app).

[234] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[235] Jonathan Lessard. Game Genres and High-Level Design Pattern Formations. In *Proceedings of the 2014 Foundations of Digital Games Conference. Florida*, 2014.

[236] Angela K-y Leung, Suntae Kim, Evan Polman, Lay See Ong, Lin Qiu, Jack A Goncalo, and Jeffrey Sanchez-Burks. Embodied metaphors and creative "acts". *Psychological science*, 23(5):502–509, 2012.

[237] Golan Levin. Computer vision for artists and designers: pedagogic tools and techniques for novice programmers. *AI & SOCIETY*, 20(4):462–482, 2006.

[238] Hannah Lewis. Community moments: Integration and transnationalism at 'refugee'parties and events. *Journal of Refugee Studies*, 23(4):571–588, 2010.

[239] Faye Li. High Fidelity Tutorial: How to Share Tilt Brush Artwork in Social VR. (https://medium.com/\spacefactor\@m{}fayeli/high-fidelity-tutorial-how-to-share-tilt-brush-artwork-in-social-vr-70b293b33093) Jan 11, 2017.

[240] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. Sentient Sketchbook: Computer-aided game level authoring. In *FDG*, pages 213–220, 2013.

[241] Louis Lin and Ian Santillano. Musical Casual Creator for Easy Self-Expression, 2018. (Senior Thesis, Santa Clara University).

[242] Aristid Lindenmayer. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of theoretical biology*, 18(3):280–299, 1968.

[243] Jim Linderman. Spirograph? No, Magic Pattern! (`https://www.artslant.com/chi/articles/show/16968`).

[244] Duri Long, Mikhail Jacob, Nicholas Davis, and Brian Magerko. Designing for socially interactive systems. In *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition*, pages 39–50. ACM, 2017.

[245] Phil Lord and Christopher Miller. The Lego Movie, 2014.

[246] Steve Losh. Chancery: A text and data generation library for Common Lisp, inspired by Tracery. (`https://github.com/sjl/chancery`).

[247] Ada Lovelace. 'Notes on L. Menabrea's 'Sketch of the Analytical Engine Invented by Charles Babbage, Esq.'. *Taylor's Scientific Memoirs*, 3:1843, 1843.

[248] Andrés Lucero. PLEX Cards - Playful Experiences Cards. (`http://www.funkydesignspaces.com/plex/`).

[249] Michelle Luo. Where in the world is your Art Selfie? (`https://www.blog.google/outreach-initiatives/arts-culture/where-world-your-art-selfie/`).

[250] Tien-Tsin Wong Yi Ji LvMin Zhang, Chengze Li and ChunPing Liu. Two-stage Sketch Colorization. *ACM Transactions on Graphics*, 37(6), nov 2018.

[251] Kevin Lynch. *The image of the city*, volume 11. MIT press, 1960.

[252] Colleen Macklin, John Sharp, and Eric Zimmerman. Metagame:A PARTY GAME THAT GETS YOU TALKING. (`http://metaga.me/`).

[253] Stéphane Magnenat, Dat Tien Ngo, Robert W Sumner, Fabio Zund, Mattia Ryffel, Gioacchino Noris, Gerhard Rothlin, Alessia Marra, Maurizio Nitti, Pascal Fua, et al. Live texturing of augmented reality characters from colored drawings. *IEEE Transactions on Visualization & Computer Graphics*, (11):1201–1210, 2015.

[254] Emanuel Maiberg. 'No Man's Sky' Is Like 18 Quintillion Bowls of Oatmeal. (https://motherboard.vice.com/en_us/article/nz7d8q/no-mans-sky-review) Aug 21, 2016.

[255] MAKE/Intel Maker Market Study and Media Report. An In-depth Profile of Makers at the Forefront of Hardware Innovation, 2012. (https://cdn.makezine.com/make/sales/Maker-Market-Study.pdf).

[256] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.

[257] Tomasz Marciszewski. Starbound Map. (https://starbound-map.herokuapp.com/planets).

[258] Evan Marcus. Disney Animation Legend Glen Keane Draws Ariel In Tilt Brush. (https://www.vr.org/2017/12/07/disney-animation-legend-glen-keane-draws-ariel-in-tilt-brush/) Dec 7, 2017.

[259] Chris Martens. Ceptre: A language for modeling generative interactive systems. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[260] Chris Martens and Rogelio E. Cardona-Rivera. The 1st Workshop on Tutorials in Intelligent Narrative Technologies (TINT 16), 2016.

[261] Janetta Mitchell McCoy and Gary W Evans. The potential role of the physical environment in fostering creativity. *Creativity Research Journal*, 14(3-4):409–426, 2002.

[262] Joshua McCoy, Mike Treanor, Ben Samuel, Noah Wardrip-Fruin, and Michael Mateas. Comme il Faut: A System for Authoring Playable Social Models. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.

[263] Gretchen McCullouch. Autocomplete Presents the Best Version of You. (https://www.wired.com/story/autocomplete-presents-the-best-version-of-you/) Feb 2, 2019.

[264] Innes McKendrick and Hello Games. Continuous World Generation in No Man's Sky, 2017. (https://www.youtube.com/watch?v=sCRzxEEcO2Y).

[265] Norman McLAren. Pas de deux (1968). (http://www.nfb.ca/film/pas_de_deux_en/).

[266] Sean McManus. *Scratch programming in easy steps: covers Versions 2.0 and 1.4*. In Easy Steps, 2013.

[267] Ivan Mehta. Google's AI to detect toxic comments can be easily fooled with 'love'. (https://thenextweb.com/artificial-intelligence/2018/09/11/googles-hate-speech-ai-easily-fooled/).

[268] Annalisa Merelli. https://qz.com/1256191/ai-learned-to-mimic-brothers-grimm-fairytales-including-their-sexist-stereotypes/. (https://qz.com/1256191/ai-learned-to-mimic-brothers-grimm-fairytales-including-their-sexist-stereotypes/) April 23, 2018.

[269] Peter Merrotsy. A note on big-C Creativity and little-c creativity. *Creativity Research Journal*, 25(4):474–476, 2013.

[270] Those Metamakers and Michael Cook. Wevva - Gameplay Video. (https://www.youtube.com/watch?v=WnE55QG86_Q).

[271] Michael Michalko. *Thinkertoys: A handbook of creative-thinking techniques*. Ten Speed Press, 2010.

[272] Microsoft. A new Xbox bundle brings together some of the best Minecraft worlds ever created. (https://news.microsoft.com/en-gb/2018/10/09/a-new-xbox-bundle-brings-together-some-of-the-best-minecraft-worlds-ever-created/) Oct 9, 2018.

[273] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[274] DL Miller. traceRy a tracery implementation in R. (https://github.com/dill/traceRy).

[275] Ria Misra and IO9 contributors. Read This Collection Of Dark Six-Word Science Fiction Stories Right Now. (https://io9.gizmodo.com/read-this-collection-of-dark-six-word-science-fiction-s-1703719532) May 11, 2015.

[276] Christopher Moore. Hats of affect: A study of affect, achievements and hats in team fortress 2. *Game Studies*, 11(1):1–14, 2011.

[277] Neema Moraveji, Jason Li, Jiarong Ding, Patrick O'Kelley, and Suze Woolf. Comicboarding: using comics as proxies for participatory design with children. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1371–1374. ACM, 2007.

[278] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks, 2015. (https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html).

[279] William Morris. *Useful work versus useless toil*. Judd Publishing Company, 1919.

[280] Brad A Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123, 1990.

[281] Jonathan Nafarrete. Tilt Brush Launches New Audio Reactive Brushes. (`https://vrscout.com/news/tilt-brush-launches-new-audio-reactive-brushes/`) Aug 2, 2016.

[282] Steve Nebel, Sascha Schneider, and Günter Daniel Rey. Mining learning and crafting scientific experiments: a literature review on the use of minecraft in education and research. *Journal of Educational Technology & Society*, 19(2):355, 2016.

[283] Mark Nelson, Simon Colton, Edward Powley, Swen Gaudl, Peter Ivey, Rob Saunders, Blanca Perez Ferrer, and Michael Cook. Mixed-initiative approaches to on-device mobile game design. *Proceedings of the CHI Workshop: Mixed-Initiative Creative Interfaces*, 2016.

[284] Mark J Nelson, Swen E Gaudl, Simon Colton, and Sebastian Deterding. Curious users of casual creators. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, page 61. ACM, 2018.

[285] András Neltz. The "Largest Minecraft Project" Will Cover An Area Of Over 5,200km^2. (`https://kotaku.com/the-largest-minecraft-project-will-cover-an-area-of-o-1609467291`) July 23, 2014.

[286] Nintendo. Mario Party, 1998.

[287] Erin O'Connor. The Snowclones Database. (`https://snowclones.org/`).

[288] Elaine O'Hanrahan. The Contribution of Desmond Paul Henry (1921–2004) to Twentieth-Century Computer Art. *Leonardo*, 51(2):156–162, 2018.

[289] Martin O'Leary. Ketchum: Use word vectors to interactively generate lists of similar words. (`https://github.com/mewo2/ketchum`).

[290] Roman Ondak. Measuring the Universe, 2007. (`https://www.moma.org/learn/moma_learning/roman-ondak-measuring-the-universe-2007/`).

[291] Charles Onyett. Dead Rising Photo Op. (`https://www.ign.com/articles/2006/07/21/dead-rising-photo-op`) July 21, 2006.

[292] Orteil. Nested. (`http://orteils-nested.wikia.com/wiki/Orteils_nested_Wiki`).

[293] Orteil. RandomGen. (http://orteil.dashnet.org/randomgen/).

[294] Windell Oskay. StippleGen: Weighted Voronoi stippling and TSP paths in Processing, 2012. (https://www.evilmadscientist.com/2012/stipplegen-weighted-voronoi-stippling-and-tsp-paths-in-processing/).

[295] Johnathan Pagnutti. How Can I Cook With This: User Experience Challenges for AI in the Home Kitchen. In *2018 AAAI Spring Symposium Series*, 2018.

[296] Johnathan Pagnutti, Kate Compton, and Jim Whitehead. Do You Like This Art I Made You: Introducing Techne, A Creative Artbot Commune. *Proceedings of the Workshop on Procedural Content Generation in Games (PCG) at FDG 2016*, 2016.

[297] DJ Pangburn. This Dapper Robot Is an Art Critic. (https://www.vice.com/en_us/article/aenq45/robot-art-critic-berenson) Feb 25, 2016.

[298] Seymour Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[299] Allison Parrish. Experimental Creative Writing with the Vectorized Word. (https://www.youtube.com/watch?v=L3D0JEA1Jdc).

[300] Allison Parrish. Python port of Kate Compton's tracery text generation library. (https://pypi.org/project/tracery/).

[301] Allison Parrish. Sea Duck: A bare-bones simulation-driven narrative framework. (https://github.com/aparrish/seaduck).

[302] Allison Parrish. Tracery for Python. (https://github.com/aparrish/pytracery).

[303] Allison Parrish. Working with Tracery in Python. (https://gist.github.com/aparrish/73c19a36b9cdcf604d04e95020418cd4).

[304] Allison Parrish. Poetic Sound Similarity Vectors Using Phonetic Features. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[305] Brad Pasanek. Shakespeare, "Sonnet 16". (https://bpasanek.github.io/puzzlepoesis/puzzlepoems/sonnet16.html).

[306] Alison Pease and Anna Jordanous. Report on the Eighth International Conference on Computational Creativity. *AI Magazine*, 39(1), 2018.

[307] Rafael Pérez y Pérez and Mike Sharples. Three computer-based models of storytelling: BRUTUS, MINSTREL and MEXICA. *Knowledge-based systems*, 17(1):15–29, 2004.

[308] Martin Pichlmair. Powerful extensible content generation library inspired by Tracery.io. (`https://github.com/martinpi/ephemeral`).

[309] Roelof Pieters. DeepDreamVideo: Implementing #deepdream on video. (`https://github.com/graphific/DeepDreamVideo`).

[310] Michael Polanyi. *The tacit dimension*. University of Chicago press, 2009.

[311] Spore Forum poster (miller0771). "What is the weirdest star/planet names you've found? I just found star system didzo...". (`http://forum.spore.com/jforum/posts/list/1515/15907.page`).

[312] Raymond Queneau. A hundred thousand billion poems. *The New Medi aReader*, 1961.

[313] Steven Ray, Vahl Scott Gordon, and Laurent Vaucher. Evolving QWOP gaits. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 823–830. ACM, 2014.

[314] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. Design principles for tools to support creative thinking. In *NSF Workshop Report on Creativity Support Tools*, pages 25–36. Citeseer, 2005.

[315] Andrew C. Revkin. The Man Behind Spore Explores Gaming as Learning. (`https://dotearth.blogs.nytimes.com/2011/02/05/the-man-behind-spore-explores-gaming-as-learning`) Feb 5, 2011.

[316] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH computer graphics*, volume 21, pages 25–34. ACM, 1987.

[317] Frances Reynolds. Managing depression through needlecraft creative activities: A qualitative study. *The Arts in Psychotherapy*, 27(2):107–114, 2000.

[318] Epistolary Richard. The evolution of Jason Morningstar's dice mechanics. (`http://www.londonindierpg.com/2016/10/the-evolution-of-jason-morningstars-dice-mechanics/`).

[319] Ruth Richards. Everyday creativity. *The Cambridge handbook of creativity*, pages 189–215, 2010.

[320] Cathy Risberg. More than just a video game: Tips for using Minecraft to personalize the curriculum and promote creativity, collaboration, and problem solving. *Illinois Association for Gifted Children Journal*, pages 44–48, 2015.

[321] Sebastian Risi, Joel Lehman, David B D'Ambrosio, Ryan Hall, and Kenneth O Stanley. Combining Search-Based Procedural Content Generation and Social Gaming in the Petalz Video Game. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[322] Sebastian Risi, Joel Lehman, David B D'Ambrosio, Ryan Hall, and Kenneth O Stanley. Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):244–255, 2016.

[323] Sebastian Risi, Joel Lehman, David B D'ambrosio, and Kenneth O Stanley. Automatically categorizing procedurally generated content for collecting games. In *Proceedings of the Workshop on Procedural Content Generation in Games (PCG) at the 9th International Conference on the Foundations of Digital Games (FDG-2014)*. ACM New York, 2014.

[324] Sebastian Risi, Sandy D Vanderbleek, Charles E Hughes, and Kenneth O Stanley. How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 153–160. ACM, 2009.

[325] Carl R Rogers. Toward a theory of creativity. *ETC: A review of general semantics*, pages 249–260, 1954.

[326] Harold Rosenberg. The American action painters. *Art news*, 51(8):22, 1952.

[327] Mark Rosewater. Nuts & Bolts, The Three Stages of Design. (`https://magic.wizards.com/en/articles/archive/making-magic/nuts-bolts-three-stages-design-2015-03-30`) March 30, 2015.

[328] James C Rosser, Paul J Lynch, Laurie Cuddihy, Douglas A Gentile, Jonathan Klonsky, and Ronald Merrell. The impact of video games on training surgeons in the 21st century. *Archives of surgery*, 142(2):181–186, 2007.

[329] Franz Rothlauf. On the locality of representations. *Proceedings of GECCO 2003*, 2003.

[330] Adam Rothstein. White Paper: Open Card Deck Standard, 2012. (`http://www.poszu.com/open-card-deck-standard.html`).

[331] Adam Row. If The Adult Coloring Book Craze Is Dead, It Needs A Postmortem. (https://www.forbes.com/sites/adamrowe1/2018/05/31/if-the-adult-coloring-book-craze-is-dead-it-needs-a-postmortem/#1f4168957dae) May 31, 2018.

[332] Aaron Rowe. Forward 40: What Became of the LOGO Programming Language? *Wired*, 2007.

[333] David Rudin. Desert Golfing's First Unbeatable Level Has Finally Been Discovered. (https://killscreen.com/articles/desert-golfings-first-unbeatable-level-has-finally-been-discovered/) March 08, 2016.

[334] John Ruskin. *The stones of Venice*, volume 2. J. Wiley and sons, 1890.

[335] Craig ("Russ") Russell. The Best of the Spore Creature Creator. (https://www.somethingawful.com/comedy-goldmine/goons-spore-creature/1/) Jun 24, 2008.

[336] James Ryan, Ethan Seither, Michael Mateas, and Noah Wardrip-Fruin. Expressionist: An authoring tool for in-game text generation. In *International Conference on Interactive Digital Storytelling*, pages 221–233. Springer, 2016.

[337] Kimiko Ryokai, Stefan Marti, and Hiroshi Ishii. I/O brush: drawing with everyday objects as ink. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 303–310. ACM, 2004.

[338] Kimiko Ryokai, Stefan Marti, and Hiroshi Ishii. I/O brush: drawing with everyday objects as ink. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 303–310. ACM, 2004.

[339] Dyani Sabin. It's Best to Scientifically Name Your 'No Man's Sky' Critters. (https://www.inverse.com/article/19490-better-animal-names-no-man-s-sky-come-scientific-naming-system-linnaeus) Aug 12, 2016.

[340] Sadiq M Sait and Habib Youssef. *Iterative Computer Algorithms: and their applications in engineering*. Wiley-IEEE Computer Society, 1999.

[341] Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

[342] Weekend Edition Saturday. 'Revelations' About a Precursor to 'Mad Libs'. (https://www.npr.org/templates/story/story.php?storyId=7584979) Feb 24, 2007.

[343] Rob Saunders and John S Gero. The digital clockwork muse: A computational model of aesthetic evolution. In *Proceedings of the AISB*, volume 1, pages 12–21, 2001.

[344] Rob Saunders and John S Gero. How to study artificial creativity. In *Proceedings of the 4th conference on Creativity & cognition*, pages 80–87. ACM, 2002.

[345] J Schell and Joe Shochet. Designing interactive theme park rides. *IEEE Computer Graphics and Applications*, 21(4):11–13, 2001.

[346] Jesse Schell. *The Art of Game Design: A book of lenses*. AK Peters/CRC Press, 2014.

[347] Thomas C Schelling. Dynamic models of segregation. *Journal of mathematical sociology*, 1(2):143–186, 1971.

[348] Anne-Marie Schleiner. Does Lara Croft wear fake polygons? Gender and gender-role subversion in computer adventure games. *Leonardo Music Journal*, 34(3):221–226, 2001.

[349] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[350] Christian Schmieder. World of Maskcraft vs. World of Queercraft? Communication, sex and gender in the online role-playing game World of Warcraft. *Journal of Gaming & Virtual Worlds*, 1(1):5–21, 2009.

[351] Donald A Schön. *The reflective practitioner: How professionals think in action*. Basic Books, New York, 1998.

[352] Christopher Scoates. *Brian Eno: visual music*. Chronicle Books, 2013.

[353] Scratch. Scratch Statistics. (`https://scratch.mit.edu/statistics/`).

[354] Thomas Albert Sebeok. *Signs: An introduction to semiotics*. University of Toronto Press, 2001.

[355] Jimmy Secretan, Nicholas Beato, David B D Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O Stanley. Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1759–1768. ACM, 2008.

[356] Janelle Shane. About me. (`http://aiweirdness.com/aboutme`).

[357] Janelle Shane. Do neural nets dream of electric sheep? (`http://aiweirdness.com/post/171451900302/do-neural-nets-dream-of-electric-sheep`).

[358] Janelle Shane. Do neural nets dream of electric sheep? (http://aiweirdness.com/post/171451900302/do-neural-nets-dream-of-electric-sheep) March 2, 2018.

[359] Janelle Shane. New paint colors invented by neural network. (http://aiweirdness.com/post/160776374467/new-paint-colors-invented-by-neural-network).

[360] Janelle Shane. personal correspondence with Janelle Shane. 2018.

[361] Janelle Shane. SkyKnit: When knitters teamed up with a neural network. (http://aiweirdness.com/post/173096796277/skyknit-when-knitters-teamed-up-with-a-neural) 2018.

[362] Shapeways. Steel (Material Information) (Design Guideline). (https://www.shapeways.com/materials/steel).

[363] Jianhua Shen and Daniel Thalmann. Interactive shape design using metaballs and splines. In *Proc. Implicit Surfaces95*, number VRLAB-CONF-2007-105, 1995.

[364] Ben Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour & Information Technology*, 1(3):237–256, 1982.

[365] Ben Shneiderman. Direct manipulation: A step beyond programming languages. *Computer*, (8):57–69, 1983.

[366] Ben Shneiderman. User interfaces for creativity support tools. In *Proceedings of the 3rd conference on Creativity & cognition*, pages 15–22. ACM, 1999.

[367] Ben Shneiderman. *Leonardo's laptop: human needs and the new computing technologies*. Mit Press, 2003.

[368] Ben Shneiderman. Creativity support tools: Accelerating discovery and innovation. *Communications of the ACM*, 50(12):20–32, 2007.

[369] Ben Shneiderman, Gerhard Fischer, Mary Czerwinski, Mitch Resnick, Brad Myers, Linda Candy, Ernest Edmonds, Mike Eisenberg, Elisa Giaccardi, Tom Hewett, et al. Creativity support tools: Report from a US National Science Foundation sponsored workshop. *International Journal of Human-Computer Interaction*, 20(2):61–77, 2006.

[370] Emily Short. Casual Procgen Text Tools. (https://emshort.blog/2016/10/27/casual-procgen-text-tools/).

[371] Emily Short. Casual Procgen Text Tools. (https://en.wikipedia.org/wiki/Lincoln_Logs).

[372] Matt Silverman. 25 'Minecraft' Creations That Will Blow Your Flippin' Mind. (https://mashable.com/2013/02/13/amazing-minecraft-creations/#8HrtHB4OpOq1) Feb 12, 2013.

[373] Joel Simon. Ganbreeder: Create beautiful, wild and weird images. (https://ganbreeder.app/).

[374] Divya Singh, Margareta Ackerman, and Rafael Pérez y Pérez. A ballad of the mexicas: Automated lyrical narrative writing. In *Eighth International Conference on Computational Creativity, ICCC, Atlanta*, 2017.

[375] James Skorupski, Lakshmi Jayapalan, Sheena Marquez, and Michael Mateas. Wide ruled: A friendly interface to author-goal based story generation. In *International Conference on Virtual Storytelling*, pages 26–37. Springer, 2007.

[376] Dan Slater. Chaotic sound synthesis. *Computer Music Journal*, 22(2):12–19, 1998.

[377] Adam Smith. Art Failure: Action Painting Pro. (https://www.rockpapershotgun.com/2014/03/21/art-failure-action-painting-pro/) Mar 21, 2014.

[378] Adam M Smith. *Mechanizing exploratory game design (Dissertation)*. PhD thesis, UC Santa Cruz, 2012.

[379] Adam M Smith, Erik Andersen, Michael Mateas, and Zoran Popović. A case study of expressively constrainable level design automation tools for a puzzle game. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 156–163. ACM, 2012.

[380] Alvy Ray Smith. Digital Paint Systems Historical Overview, 1995. Tech. report, Microsoft Research.

[381] Alvy Ray Smith. Varieties of digital painting, 1997. Tech. report, Microsoft Research.

[382] Gillian Smith. Understanding procedural content generation: a design-centric analysis of the role of PCG in games. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 917–926. ACM, 2014.

[383] Gillian Smith and Jim Whitehead. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 4. ACM, 2010.

[384] Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 209–216. ACM, 2010.

[385] Kristen Sollee. What Does Each Emoji Heart Mean? (`https://www.bustle.com/articles/81971-what-does-each-emoji-heart-mean-instagram-breaks-it-down-for-us-based-on-their-most`).

[386] Spore and John Cimino. Spore Creature Creator Tutorial 1. (`https://www.youtube.com/watch?v=ZRr3lgckIAM`).

[387] 71 Squared. Deep Dreamer. (`https://www.71squared.com/deepdreamer`).

[388] Dietrich Squinkifer. Interruption Junction, 2015. (`http://squinky.me/2015/01/18/interruption-junction/`).

[389] Dietrich Squinkifer. Fitzwilliam Darcy's Dance Challenge , 2016. (`https://squinky.itch.io/darcy`).

[390] Dietrich Squinkifer. Most Sincere Greetings, Esteemed One, 2016. (`http://squinky.me/2016/01/31/most-sincere-greetings-esteemed-one/`).

[391] IGN Staff. Get out and Spore the Vote. (`https://www.ign.com/articles/2008/11/03/get-out-and-spore-the-vote`) 3 Nov, 2008.

[392] Andrew Stern. Grand Text Auto: introductory post. (`https://grandtextauto.soe.ucsc.edu/2003/05/`) May 10, 2003.

[393] Robert Sternberg. Creativity as a habit. In *Creativity: A handbook for teachers*, pages 3–25. Citeseer, 2007.

[394] Robert J Sternberg and Todd I Lubart. The concept of creativity: Prospects and paradigms. In *The Cambridge handbook of creativity*, pages 3–15. Cambridge University Press, 1999.

[395] Jenna Stoeber. Pokémon Snap was ahead of its time. (`https://www.polygon.com/videos/2018/9/24/17876562/pokemon-snap-camera-photo-mode`) Sept 26, 2018.

[396] style2paints. style2paints.github.io: sketch + style = paints! (`https://style2paints.github.io/`).

[397] Anne Sullivan, Joshua McCoy, Sarah Hendricks, and Brittany Williams. Loominary: Crafting Tangible Artifacts from Player Narrative. *Proceedings of the Twelfth International Conference on Tangible, Embedded, and Embodied Interaction*, pages 443–450, 2018.

[398] Ian Sutherland and Sophia Krzys Acord. Thinking with art: from situated knowledge to experiential knowing. *Journal of Visual Art Practice*, 6(2):125–140, 2007.

[399] Colin Symes. Writing by Numbers: OuLiPo and the Creativity of Constraints. *Mosaic (Winnipeg)*, 32(3):87, 1999.

[400] Dave Tach. No Man's Sky is so big, the developers built space probes to explore it for them. (`https://www.polygon.com/2015/3/3/8140343/no-mans-sky-space-probes-gdc-quintillion-worlds`) Mar 3, 2015.

[401] Jerry Talton, Daniel Gibson, Pat Hanrahan, and Vladlen Koltun. Collaborative mapping of a parametric design space. Technical report, Citeseer, 2008.

[402] Joshua Tanenbaum, Karen Tanenbaum, and Michael Cowling. Designing Hybrid Games for Playful Fabrication: Augmentation, Accumulation & Idleness. In *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play*, pages 413–419. ACM, 2017.

[403] Spore Team. Ten of the Top Spore Creators Visited Maxis! (`http://www.spore.com/comm/updates/update051209`) May 12, 2009.

[404] Spore Team. Holiday Team Contest 2009, 2009. (`http://www.spore.com/comm/contests/archivedcontests`).

[405] teamLab. Play Installations. (`https://futurepark.teamlab.art/en/attraction/`).

[406] Maria Thomas and Rick Roberts. Zentangle. (`https://www.zentangle.com/`).

[407] Maria Thomas and Rick Roberts. Zentangle: What Is It? (`https://zentangle.com/pages/what-is-the-zentangle-method`).

[408] D'Arcy Wentworth Thompson. *On growth and form*. Cambridge Univ. Press, 1917.

[409] Michael (incobalt) Thomét. A wrapper for Tracery.js (by galaxykate) for Twine 2.x Sugar Cube format. (`https://github.com/incobalt/Trice`).

[410] Hamish Todd. Virus, the Beauty of the Beast. (http://viruspatterns.com/).

[411] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[412] Bruce Tognazzini. Principles, techniques, and ethics of stage magic and their application to human interface design. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*, pages 355–362. ACM, 1993.

[413] Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. Game-o-matic: Generating videogames that represent ideas. In *Proceedings of the The third workshop on Procedural Content Generation in Games*, page 11. ACM, 2012.

[414] Sherry Turkle. *Life on the Screen*. Simon and Schuster, 2005.

[415] Plus Ultra. Monstr (A Dating Website for Monsters), 2015. (http://ludumdare.com/compo/ludum-dare-33/?action=preview&uid=55241).

[416] IMGUR user (blagh). Monument Valley 2 mandalas. (https://imgur.com/gallery/LNLYF).

[417] Youtube user (cat333pokemon). Bohemian Rhapsody Mario Paint Composer. (https://www.youtube.com/watch?v=e0_70TBTTiY).

[418] Youtube user (DoodleChaos). Line Rider - Mountain King. (https://www.youtube.com/watch?v=RIz3klPET3o).

[419] IMGUR user (GrieverXIII). Random Name Generator fail. (https://imgur.com/gallery/nsRpCOj) May 30, 2015.

[420] Reddit user (Obyekt). Cycle Count Leaderboards. (https://www.reddit.com/r/opus_magnum/comments/780d71/cycle_count_leaderboards/).

[421] Youtube user (Squakenet). Music Construction Set gameplay (PC Game, 1984). (https://www.youtube.com/watch?v=SMWJf7tm3v0).

[422] Steam user (Stranger). "Silliest ways you died". (https://steamcommunity.com/app/333640/discussions/0/1743342647560092923/) Aug 6, 2018.

[423] Valve. Team Fortress 2, 2007.

[424] Tony Veale and Mike Cook. *Twitterbots: Making Machines that Make Meaning*. MIT Press, 2018.

[425] James Vincent. Twitter taught Microsoft's AI chatbot to be a racist asshole in less than a day. (https://www.theverge.com/2016/3/24/11297050/tay-microsoft-chatbot-racist) Mar 24, 2016.

[426] Paul Virilio. *Art and fear*. A&C Black, 2006.

[427] Alex Walker. Some Keyforge Decks Are Being Recalled Because Of Their 'Unfortunate' Names. (https://www.kotaku.com.au/2018/11/some-keyforge-decks-are-being-recalled-because-of-their-names/) Nov 21, 2018.

[428] Noah Wardrip-Fruin and Nick Montfort. *The New Media Reader, A User's Manual*. MIT press, 2003.

[429] Philippa Warr. Firewatch: Why The Official Photos Don't Quite Work. (https://www.rockpapershotgun.com/2016/09/05/firewatch-official-photo-print-service/) September 5, 2016.

[430] Philippa Warr. Turning painting into a game mechanic in the gorgeous Eastshade. (https://www.gamasutra.com/view/news/336400/Turning_painting_into_a_game_mechanic_in_the_gorgeous_Eastshade.php) Feb 12, 2016.

[431] Howard Crosby Warren. *Human psychology*. Houghton Mifflin, 1919.

[432] Christian Weichel, Manfred Lau, David Kim, Nicolas Villar, and Hans W Gellersen. MixFab: a mixed-reality environment for personal fabrication. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3855–3864. ACM, 2014.

[433] Richard Whitelock. Quiet as a Stone, 2018. (http://www.quietasastone.com/).

[434] Dwarf Fortress Wiki. Bloodline:Boatmurdered. (http://dwarffortresswiki.org/index.php/Bloodline:Boatmurdered) April 18, 2008.

[435] Second Life wiki. Making money. (http://wiki.secondlife.com/wiki/Making_money).

[436] Andrew Willmott, Dan Moskowitz, and Ocean Quigley. Inside GlassBox. (http://www.andrewwillmott.com/talks/inside-glassbox).

[437] Emma Winston. Blackout Poetry Maker. (https://blackoutpoetry.glitch.me/#).

[438] Wired. Monterey Bay Aquarium Opens Interactive Jelly Exhibit. (https://www.youtube.com/watch?v=eySeFvFAybQ).

[439] Lior Wolf, Yaniv Taigman, and Adam Polyak. Unsupervised creation of parameterized avatars. *CoRR, abs/1704.05693*, 2017.

[440] Paul Woolcock. tracery - Rust. (https://docs.rs/tracery/0.1.0/tracery/).

[441] Wilhelm Max Wundt. *Principles of physiological psychology*. Macmillan Co., NY, 1904.

[442] Rafael Pérez y Pérez, Santiago Negrete, Eduardo Peñalosa, Rafael Ávila, Vicente Castellanos-Cerda, and Christian Lemaitre. MEXICA-Impro: A Computational Model for Narrative Improvisation. In *ICCC*, pages 90–99, 2010.

[443] YAYOMG. Design Your Own Disney Fun with These Seedling Craft Kits. (https://www.yayomg.com/design-your-own-disney-kits-seedling/) Nov 1, 2016.

[444] Derek Yu. *Spelunky: Boss Fight Books# 11*, volume 11. Boss Fight Books, 2016.

[445] José P Zagal, Staffan Björk, and Chris Lewis. Dark patterns in the design of games. In *Foundations of Digital Games 2013*, 2013.

[446] Lawrence M Zbikowski. *Conceptualizing music: Cognitive structure, theory, and analysis*. Oxford University Press on Demand, 2002.

[447] Eva Zeisel. *On design: the magic language of things*. Overlook Press, 2004.

[448] Eva Zeisel. *On design: the magic language of things*. Overlook Press, 2004.

[449] Natalie Zmuda. 'Sims 2' isn't just a game: it's a lifestyle. (https://adage.com/article/madisonvine-news/sims-2-a-game-a-lifestyle/127117/) May 15, 2008.

[450] Zack Zwiezen. Grand Theft Photo: Why These GTA Shots Look So Damn Good. (https://kotaku.com/grand-theft-photo-why-these-gta-shots-look-so-damn-goo-1823743440) March 13, 2018.

[451] Boštjan Čadež. Line Rider. (https://web.archive.org/web/20080618082205/http://linerider.com/en/news).