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 systems 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.

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 (it's 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" Casual creators have a limited possibility space that is defined by their generative pipeline. 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.
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 it’s 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 let’s 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 surprising 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 (it’s 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 Saunders'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."

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 laying 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:

- 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 possibility 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).
It's 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.

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 it's 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:

- 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 them 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)

### 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?
How do we get from “what we want to do” to “how do we do it” to “did it work?” in a digital system. 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. (pictures from his 1986 paper “Cognitive Engineering”)

The gulf of execution and the gulf of evaluation

Your choice of how the user controls the model input (sliders, text fields, gesture-control, a menu) will be defined by your model-space. 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 Execution. The size of the gulf of execution determines how quickly your users can make things, explore the tool, and discover hidden possibilities.

- increase speed of generation
- at-a-glance evaluation of the artifact
- visualizations, evaluations, or other annotations of the artifact
  - some annotations are incomplete or even fictional!

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.

### 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!

### 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.
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

tiny: 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.

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 possibility 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 possibility space and you may discover a new pattern.
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 mastery 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

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.
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 aficionados 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!