Generominator: Operationalizing the Computational Creativity Ideation Cards "Generominos"
Paper type: System or Resource description paper

Gabriel Bacon* and **Rosalind Dixon*** and **Max Kluiwstra-Descalle*** and **Audrey Ostrom*** and **Alexander Sherbrooke*** and **Thomas Wessel*** and **Jiangyue Wu*** and **Markus Eger**
{gbacon,romdixon,mkluiwst,aostrom,asherbro,twessel,jwu454,meger}@ucsc.edu
UC Santa Cruz
Department of Computational Media

Abstract

To facilitate the exploration of computational creativity in all its aspects, the concept of "Casual Creators" arose to describe software that allows the playful expression of creativity in some creative space. Separately, Compton et al. presented a set of (physical) ideation cards for computational creativity they called "Generominos". In this paper, we present a digital implementation of these ideation cards that allow users to compose a wide range of different algorithms in a visual, graph-based web interface to generate digital computational creativity artifacts. We discuss the process of giving formal execution semantics to the previously loosely-defined ideation cards, additions and limitations we had to impose on the system, and demonstrate the wide range of possibilities enabled by our system, which include generative processes for sound, images, text, 3D graphics, etc., as well as transformations between them. Additionally, we evaluate our system against a set of desirable design patterns from the literature.

Introduction

Computational Creativity can involve a wide range of different media (Graves et al. 2021), operations (Compton, Kybartas, and Mateas 2015; Eger 2022), output artifacts (Mazeika and Whitehead 2017), and means of expression (Tuffs 2017; Mathewson and Mirowski 2017). Exploring such a vast space may seem intimidating to more casual users, as even an overview of what *may* be done is challenging to come by. Various authors have thus explored ways of making computational creativity more approachable, with Kate Compton arguably being the most vocal proponent of systems called "Casual Creators" (Compton and Mateas 2015). These systems present an approachable way to generative methods, following the tradition of thousands of years of human play with assembling building blocks into larger structures, and have also found adaption in computational domains, such as with Tanagra (Smith, Whitehead, and Mateas 2010). One way of enabling such exploration is through ideation cards, like Compton et al.'s (2017) Generominos, which present computational concepts as a possibility space which users can explore through imagination, with the option to later implement a system ideated this way. While this allows a more free-form exploration, it also does not provide any direct feedback of what a system may actually produce as output. In this paper, we present an operationalization of aforementioned Generominos in the form of our system, aptly named the Generominator, which allows users to explore a wide range of computational creativity algorithms and get real time feedback in the process. Our core contribution is this system, consisting of formal semantics for roughly 100 different cards which can process sound, images, 3D meshes, curves, and other data types and combine them in a wide variety of ways, and a web-based editor which allows users to construct graphs that combine these operations in a guided way, and supports them in finding cards compatible with their current processing pipeline. We will discuss our implementation in detail, provide several sample output artifacts showcasing the richness of the possibility space enabled by our system, and evaluate it against criteria outlined by Kate Compton in her work on Casual Creators.

*These authors contributed equally.

Related Work

Our work exists in the space Kate Compton and Michael Mateas (2015) termed "Casual Creators", i.e. software that allows the playful exploration of computational creativity, as an expression of everyday creativity (Kaufman and Beghetto 2009), or, as Compton and Mateas so aptly put it, because *it is fun*. While there may often be an emphasis on (computational) creativity in the pursuit of some more "productive" goal, such as engineering (Masood et al. 2024) or industrial design (Heisserman 1994), creation for creation's sake has also been explored by prior work. Kreminski et al. (2020) present the Germinate system, based on the Gemini game generator by Summerville et al (2018). Germinate allows users to define parameters for a game using a web-based interface, which are then passed to Gemini to generate a playable game artifact. While our system targets a different type of target artifact, and has a very different underlying formalism, it follows a very similar design philosophy.

More broadly, our work exists within a long tradition of visual, graph-based programming tools that span many genres of systems (Morrison 2011). Sood et al. (2025), for example, presented a graph-based system that supports creative writing. These graph-based editors also commonly appear within procedural content generation and game devel-

opment work, and can be seen in tools such as Blender[1], Unreal Engine[2] or Unity ShaderGraphs[3]. Our system takes inspiration from these designs, enabling users to add, remove, and connect various types of nodes in a graph to form a completed and executable program.

The core inspiration for our work, though, are the Generominos ideation cards by Compton et al. (2017). Each card represents an input, transformation, or output operation that can process data of different data types, including sound, images, text, 3D meshes, etc. Our system represents an operationalization of these cards, together with a web-based interface that supports casual exploration and reflection (Kreminski and Mateas 2021). Since the Generominos were designed to encourage flexible interpretation across many possible implementations, some operational details such as data representations, evaluation order, and cross-type conversions were intentionally underspecified in the original design. Our implementation of these cards gives one of many possible definitions as a way to produce immediately working prototypes. We would like to note that Compton maintains an existing Generominos editor[4] that supports browsing and connecting cards in a web-based user interface, similarly to how the physical cards are used, but lacks the execution semantics of our system. Additionally, other community versions of Generominos exist, such as a port to the game TableTop Simulator[5]. In the next section, we discuss the concrete operational choices we made, and how our interface encourages graph exploration for artifact generation.

Operationalizing Generominos

As presented by Compton et al., each of the Generominos ideation cards can have various inputs and outputs that can have a variety of different data types. The entire generative process is driven by cards consuming their inputs, transforming them in some way and producing the corresponding outputs. Cards may be sources, which feed data into the system, transformations, which apply some algorithm (or just a simple operation) to data, or outputs, which produce an artifact consumable by an audience. While the original paper briefly describes the data types, its goal was not any particular implementation, and thus it does not provide concrete detail of how each is represented. For our system, we therefore first had to provide a concrete definition for each data type, and how cards can interface to consume and produce this data. We also implemented an execution engine that can take a graph consisting of connected cards and evaluate them in a defined order. To facilitate the design of these graphs, we also developed a web-based user interface in which users can intuitively connect the various cards to explore the creative capabilities of the system. In this section, we will describe each of these parts of our system, starting with the data types.

[1]https://www.blender.org/

[2]https://www.unrealengine.com/

[3]https://unity.com/

[4]https://www.galaxykate.com/generominos/editor-dev/

[5]https://steamcommunity.com/sharedfiles/filedetails/?id=1396406737

Data Types

Generominos are best understood as defining a data flow graph, where data can have one of several data types. Our system supports:

- Atomic types: value (number), text (string), and state (boolean).
- Array types: vector (array of numbers), vectorfield (array of vectors), shape (array of curves).
- Curve: A parametric curve that maps a value between 0 and 1 to points along the curve. Curves may also have a color associated with them. We define an abstract Curve interface for this purpose, and our system currently contains Bezier Curves and Polygons as concrete curve implementations.
- Graph: Node positions represented by vectors with edges that are defined in an adjacency matrix.
- Color: Stored as red, green, blue, and alpha values.
- Image: ImageData object, which represents the image as a flat array of bytes with an associated width and height.
- Waveform: Audio sample data stored as a flat array of numbers with an associated sampling rate.
- Depthmap: An array of numbers, going from top left to bottom right, storing a depth value for each point.
- Geolocation: Longitude and latitude stored as two values.
- Trimesh: A representation of 3D geometry as an instance of the THREE.Mesh class from the threejs library[6].
- Generic: A wrapper that is used as a placeholder by cards that may operate on any type (e.g. conditionals).

We will now describe how cards can process this data.

Cards

To process data in our system, we define two interfaces: Card and Port. The Card interface outlines what a card itself may do. It includes a title, a list of input ports, a list of output ports, and an optional description, as well as 3 functions: `evaluate`, `init`, and `cleanup`. The title is the name of the card displayed in the UI, as well as the card's identifier for operations such as saving to JSON. The description, like the title, is printed on the card in the UI and provides the user with more information on what the card does and how to use it. The input and output ports define what data types the card operates on and make use of the Port interface.

The lifecycle of a card consists of three stages:

1. Before execution begins, the `init` method is called, which can prepare any data needed to run the card, connect to API services, download assets, etc.
2. `evaluate` is called by the execution engine to process data and defines the cards behavior. It both takes in and outputs a list of DataTypes, which need to match the input and output port list of the card, respectively.
3. `cleanup` is run when execution ends and should be used to dispose of any resources the card used.

[6]https://threejs.org/

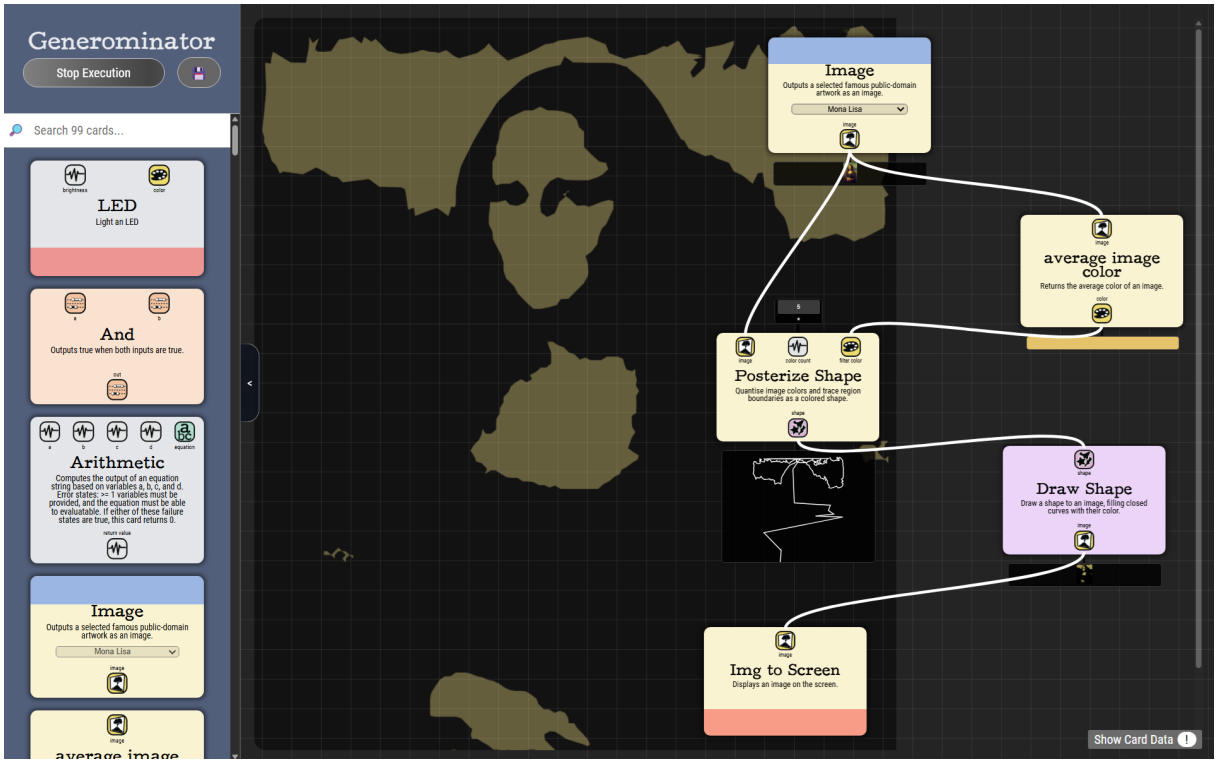


Figure 1: The user interface of Generominator during a system run. Using the card bank on the left, users can create systems by connecting cards together. Users can then run their system to see the outputs of each card, and the system itself.

Most simple cards will only implement `evaluate`, but more complex operations involving API calls or hardware access may require initialization or cleanup.

The Port interface outlines a single data port on a card. It includes four variables: a label, a flag indicating if the (input) port is optional, a data type and an optional default value. The label is defined per instance of each port and is printed above or below each data port in the UI. The optional flag marks whether or not the data port will require input for the execution graph to run. The data type defines which data can be passed through this data port, and the default value will be used in place of data from another card if no connections are made to the port during execution.

Using input- and output-ports as connecting points, we can define a graph consisting of multiple cards that connect some or all of the outputs of a card to another card, which in turn produces outputs that are passed to another card, and so on, until we reach a card that does not have any output ports. In our system, such cards will typically have side effects that display data on the screen, or play something on the speakers. Next, we will describe how our system takes a graph defined in such a way to actually produce an output.

Execution Model

At the heart of the Generominator is the execution engine, responsible for both managing the card life cycle and graph execution. When cards are dragged onto the editor canvas, an entry for them is created within the engine's internal Directed Acyclic Graph. As connections are made between cards in the interface, nodes in the graph are also linked. This gives an internal representation of where data should flow, and the order in which card execution must occur.

At the start of each engine run, the graph is first sorted topologically via Kahn's Algorithm (Kahn 1962). This results in a sorted list that guarantees any card dependent on another's output will be placed after it in the list. This ensures that execution flow is handled in the correct order and data can be propagated through the graph correctly. Our implementation supports running multiple disconnected graphs, but does not currently support cycles. If one is found, it may be dropped from the dependency tree and result in undefined behavior. To prevent this, the editor halts execution if a cycle is found.

Normal runs of the engine traverse the entire list in order, starting from the eldest parent nodes and passing their results through each child until every node in the list has been visited. In some cases, however, it may be desirable to execute only a subset of the full graph. This is most prominently on display with another engine feature: card events. Events enable cards to inform the engine that their data has changed and cause it to re-execute the graph in order to remove any stale data left in the result cache from the previous state. In other words: Using events, cards can cause the graph to be reevaluated when their data changes, which may include when certain external events have occurred that were queried over an API, the user performs an input action, or through

cards that provide a continuous stream of input, like a webcam, or time. When such cards are near the beginning of the graph, this is not much of an issue as all downstream cards will need to evaluate anyway to update their information, but can become a problem if they are near the end. With every run request, card data is needlessly re-evaluated.

To solve this, our engine supports starting execution from any arbitrary node with a feature that we call "subgraph execution." With this feature, when a card's data changes and the graph moves to re-evaluate it, any parent cards to that one are skipped and execution begins directly with the node that changed. Any data that the parent cards would have generated is instead pulled from the engine's internal cache of results from the previous execution. Executing from that node onwards completes like a standard run, and the resulting data is cached for later use.

Events and subgraph execution enable the creation of card types that would otherwise be impossible to implement, ranging from the potentiometer which updates its state with a GUI element, to the Bluesky card, which continually polls the API and outputs any strings that match its input. In the case that a card updates very frequently, however, such as the mouse coordinate card, the engine can still be overwhelmed with re-execution requests. Moving the mouse across the screen might push hundreds of requests into the queue which could take seconds to execute, leading to the user feeling as if the program is not responsive. Our solution to this problem is what we call "run merging": Each new execution request is compared with the one already in the run queue (if it exists). Whenever possible, new requests have their data merged into that request, overriding any data that might otherwise have been pulled from the cache. In the case that the same card is repeatedly updated, old states are overwritten by the new state which enables us to skip running steps that would otherwise appear to the user extremely briefly, and maintain realtime reactivity.

Finally, there are card interactions that might require complex routing of data and operations on that data. To address this, we have added the generic datatype and the ability for cards to trigger output on ports independently of each other. Generic ports have the ability to transform into any other type when connected to another card and can act independently or transform all other generic types on the card. Both of these features are employed by the branch card, where any data is taken in and is only pushed to the branch determined by a value port, enabling complex data flow compositions that can route data in a data-dependent way.

We will now describe how we allow users to manipulate the graphs that are run by our execution engine using our user interface.

User Interface

The Generominator is available as a web application accessible on both desktop and mobile devices[7], shown in figure 1 with our source code available on Github[8]. Users can drag cards from the card bank on the left onto the canvas on the right. On each card is a number of input and output ports that determine what data the card takes in or provides; by clicking and dragging from a port, users can create connections to other cards. Alternatively, users can set ports to accept constant inputs by interacting with the floating editors on the input ports.

When a user is done creating their system, they can click "Run Syste" to run their card graph through the execution engine. The outputs of each card appear under the output ports of each card, showing their system at every stage of the graph. The user is able to modify their graph and see their results update in real time. Additionally, some cards, such as the Button, Webcam, and Potentiometer, provide values that can change over time or through user inputs, allowing for dynamic systems to be created.

Unusual among graph-based programming systems, the Generominator allows users to save their systems as a PNG file, depicting a card, that can be shared online. Inspired by existing implementations, such as Spore's creature sharing system[9] or the PICO-8 game cartridge system[10], this file stores the system data within the pixels of the image using steganography. The image is then able to be uploaded to various social media platforms, such as Discord, Twitter and Tumblr[11]. Once shared, other users can save the image and load it into their editor to load the saved system.

Results

In this section, we present an evaluation of our system. A key goal during development was the creation of a system that is both flexible for developers and explorable for users. We will discuss how the very development process itself promotes this flexibility and extensibility, and then evaluate our system against design patterns set forth by Compton and Mateas (2015). Additionally, we will present a selection of actual graphs designed using our system, and the capabilities they demonstrate, while also discussing the limitations our system has.

Collaborative Coding and Extensibility

Generominator's strength as a tool lies in its modularity. Its appeal as creative support manifests in the possibility space each connection offers, and its reliability in that supportive pursuit is dependent on the fidelity of each card. This dynamic results in a massive capacity for meticulous, concentrated development. A natural extension of the tool is through the addition of cards. This is logically inherited from Kate Compton's original physical version. In modifying an analog set of cards, insertion is most straightforward. Without a technical framework, the path *between* cards is handled in the theater of the mind and modifying essential qualities of the card-base would necessitate reprinting

[7] Available online: https://generominator.dev

[8] https://github.com/Generominator/Generominator

[9] As described in Kate Compton's 2025 AIIDE keynote: https://www.youtube.com/watch?v=jlW8aaGcKzM

[10] https://www.lexaloffle.com/dl/docs/pico-8_manual.html

[11] Care must be taken, though, as some social media platforms, like Bluesky, may re-encode PNG files in a way that loses the embedded data

or posthumous editing. This leaves card-creation as the instinctive mode of alteration and is encouraged, even, by the original, where tools to create new cards were provided[12].

The Generominator technical framework endeavors to support card-creation with ease equal to its analog counterpart. For a collaborator in the card-creation process, only an understanding of input, output, and data type is required. Individual cards do not impact each other unless paired, and only function together if their input and output are compatible, significantly reducing the chance for dissonance between cards. Because incoming data is screened before being transformed, in a rare incompatibility scenario, problem data is surfaced to the user before contaminating outgoing data or its surrounding graph downstream.

This benefits the development process in a very practical sense. For a hopeful collaborator to contribute a card, they most likely do not need to alter major, load-bearing files like those in the card base or those associated with the execution engine. Instead, their card can be entered into the card folder, built in accordance with the card template, all without the risk of impacting any card but their own.

With our repository, we provide a detailed README file that contains a step-by-step guide for card contributions. In general, to add a new card, a developer only has to create a new subclass of the `Card` class, providing the title, as well as input and output ports, and implementing the `evaluate` function, and the new card will be found automatically by the system and be available for use in the system. In Generominator's development process, it was this exact disconnect between each card that allowed code contributions from a diverse set of developers (all eight authors of this paper contributed multiple cards to the system) to be swift, reliable, and safe. After the card base was established, each team member could develop their own selection of cards independently of each other. Without the risk of multiple parties significantly changing the same files, the likelihood of code conflicts was and remains low.

Additional opportunity for major changes lie in the addition of data types. This category of change was explored in this digital iteration of Generominator via the `generic` data type, which accepts all others, and is not present in Kate Compton's analog version of the tool, but future work can easily add new port types to process kinds of data we currently do not support.

Because of their detachment from each other, adding and altering data types bear the same benefits as adding and altering cards. Type additions do not require altering any other data type. This makes their contribution straightforward, while still being hugely valuable, compounding Generominator's dynamism and exponentially expanding each graph's possibility space, especially if conversion cards are provided to interface the existing cards with a newly added data type.

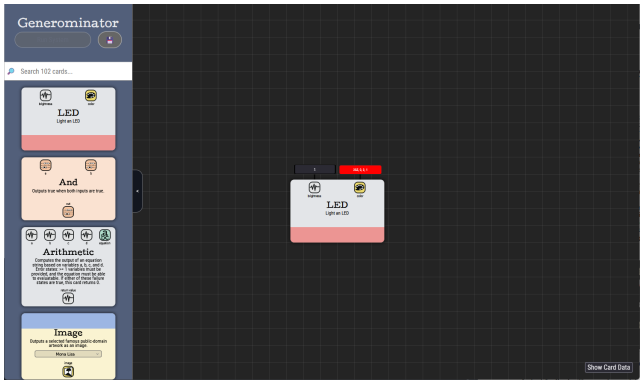
Generominator's inherent modularity, assumed from the card-based metaphor at its core, results in a code base that is easily improved, added to, and revised. This is exemplified by the authors' contribution patterns, which rarely overlapped, and by future contribution guidelines, which can be followed without risking the fidelity of the tool as a whole.

[12]This repository provides tools for users to create their own cards and suggests ways to get them printed: https://github.com/galaxykate/generominos

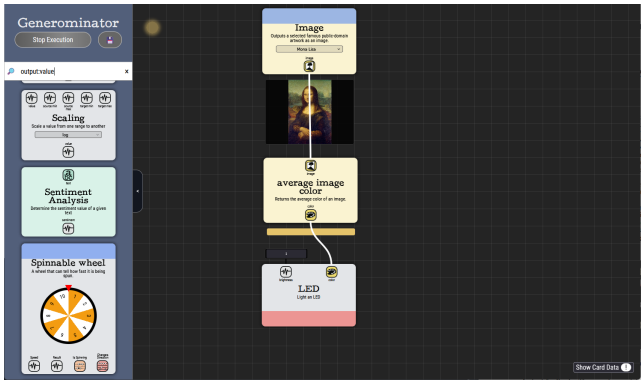
Design Patterns

Drawing upon the Casual Creators framework (Compton and Mateas 2015), our system is designed to support autotelic creativity and facilitate a state of flow. By integrating modularity into a card-based interface, we encapsulate complex functions within card units. This approach allows users to engage in reflection-in-action (Schön 2009) through direct interaction, where possible connections between the cards guide exploration of creative possibilities. By emulating physical-world object interactions, the system fosters a sense of Direct Engagement (Hutchins, Hollan, and Norman 1985). Instead of navigating through abstract menus, users experience a direct, unmediated connection to the generative components. This design challenges traditional command-based interaction, allowing users to "inhabit" a modular world where creative outcomes are the immediate result of physical-like manipulation. The Casual Creators framework specifically enumerates a list of desirable design patterns, of which our system directly employs several:

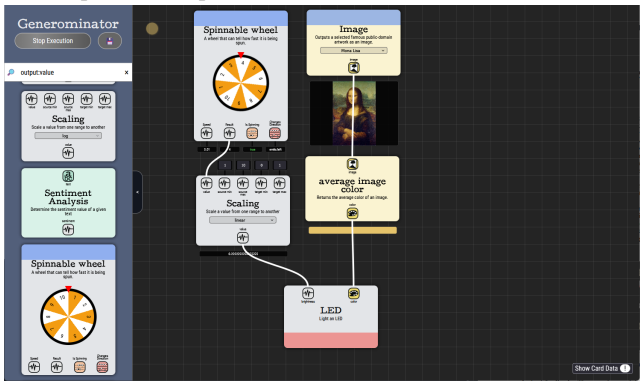
- **Instant feedback:** The core design philosophy centers on direct, card-based interaction. The Input/Output interfaces are engineered to provide "reactive electricity": instantaneous computational propagation that occurs the moment two cards are connected. Each card offers real-time feedback in the form of a direct preview of its outputs. On the other hand, by emulating the behavior of physical cards, the system significantly lowers the learning curve and effectively bridges the "Gulf of Evaluation" (Hutchins, Hollan, and Norman 1985), which is the cognitive gap between a user's intentions and the system's perceived state. In contrast, traditional creative suites like Photoshop or Maya bury their logic within deeply nested menus, forcing users into taxing mental simulation (Norman 2013). In the Generominator, the main interaction mode users are engaged with is adding cards to the graph by dragging new cards onto the canvas or establishing connections between cards.
- **Limiting actions to encourage exploration:** Instead of providing absolute freedom, the system deliberately restricts the user's action space. Each card is designed with a limited number of slots and specific data-type constraints, which a user can explore in a targeted way. Dragging a connection from an open port and releasing the mouse also shows a popup window listing only the cards that can connect to that port, and establishing such a connection immediately if one is selected, as shown in Figure 4. These constraints shift the user's cognitive focus from "what can I do" to "let me see which card fits here," effectively scaffolding the exploration process. By narrowing the possibility space, the system prevents the decision paralysis and anxiety often associated with excessive choice (Csikszentmihalyi 1990). Furthermore, as illustrated in Figure 3, the interface provides immediate negative feedback when an invalid connection is attempted.



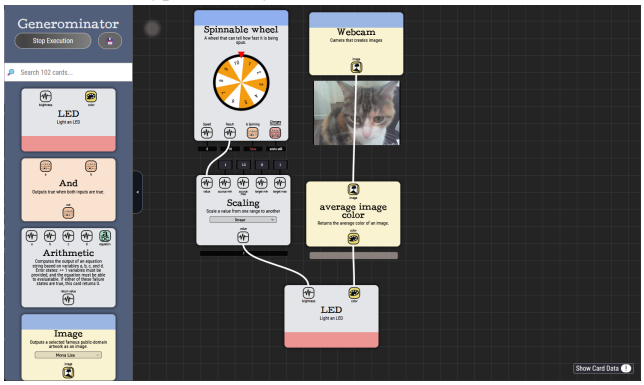
(a) The user can start their exploration by selecting any card, such as a simple LED output.



(b) The user can use the search function to browse cards that produce data of a type that they need.



(c) As the wheel is spun by the user, the halo effect (representing brightness) on the LED changes in real time.



(d) Cards can be replaced easily to explore other possible data sources or transformations.

Figure 2: A possible exploration sequence.

This mechanism ensures that the user remains within the "flow zone," where challenges are perfectly balanced with the system's structural guidance.

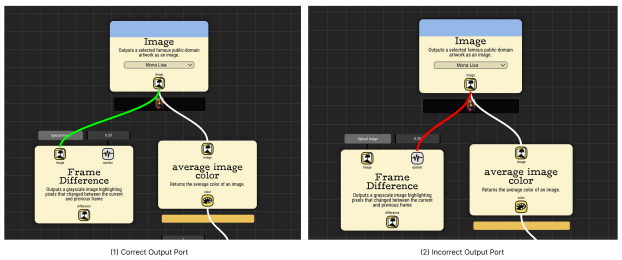


Figure 3: Attempting to connect invalid data types is disallowed and results in immediate feedback (red connection).

- **Mutant shopping:** Mutant Shopping serves as a mechanism to help users discover unexpected solutions within the vast possibility space. In our system, the card bank situated on the left side of the interface allows users to explore cards with diverse functionalities. Aforementioned suggestion box when the user drags from an unconnected port serves a similar purpose. Rather than drawing from a blank canvas, users "shop" from a series of generated functional cards with subtle variations. This shifts the creative act from active production to a process of browsing and recombination of existing processes. By leveraging the psychological pleasure and motivation inherent in non-directed discovery, this design alleviates the pressure to "produce" and encourages creators to find new opportunities in a state of freedom. This approach mirrors the collaborative discovery mechanism of Picbreeder (Secretan et al. 2011): users do not construct logic from scratch but instead "navigate" the possibility space, thereby significantly reducing cognitive load.

- **Saving and sharing:** As users explore our interface, they may design more and more complex graphs that incorporate a variety of different nodes in novel ways. We allow users to save and load graphs in a conventional JSON format, which can be edited in a text editor or from other scripts. However, we also support saving graphs to PNG files by using steganography, which embeds the graph data in the pixels of the PNG itself, while looking like an actual card with a label and (optional) description, as shown in figure 5. This allows graphs to be shared easily by posting the image on social media. To reimport the graph, the user just drags the image file onto the canvas,

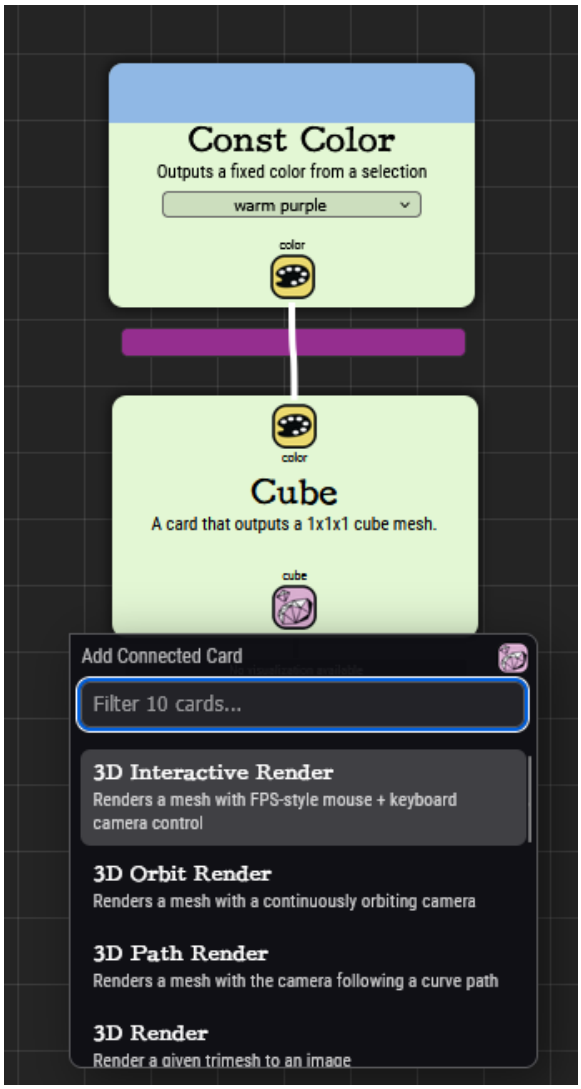


Figure 4: When the user drags a connection from an open port and releases it over the empty canvas, they are presented with the list of cards that are valid connection options. In this example, dragging from the unconnected mesh port will show the user all cards that can process mesh input. Filtering nodes on possible data types is a common feature in flow-based programming systems, and was directly inspired by how Compton's original editor filters nodes, as well as how Unity ShaderGraph presents the same filter operation.

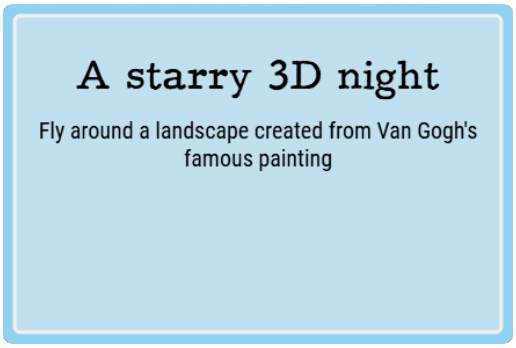


Figure 5: A graph exported to a PNG file looks like a card with a custom label, but the graph data remains embedded in the pixels of the image.

which will restore the embedded graph data.

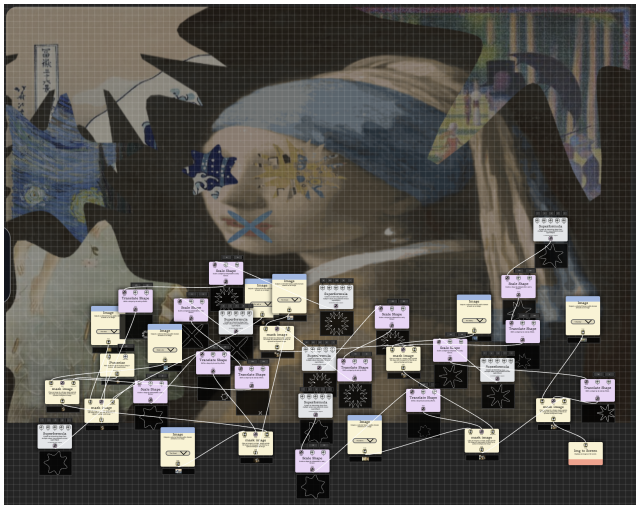
- **Modifying the meaningful:** While our system may not directly enforce this, the creation of complex graphs typically involves having some high level control parameters, which have ripple effects on other cards' behavior downstream. This way, the user can interact with the meaningful parameters of a generative system, and see their changes have an effect in real time.
- **Modding, hacking, teaching:** During the exploration process, user may discover that Generominator does not yet support every conceivable function. One core design goal was to make the system easily extensible, and advanced users can easily script their own card logic and share it with others. In fact, during the development process, we would often come up with new card ideas that expanded the creative space significantly but were trivial to implement, such as creating a duality between waveforms and vectors (handling sampling rate appropriately), which opens up the space of vector-processing cards for use with waveforms and vice versa. The card-based interaction tool is thus transformed from a "black box" into an extensible, dynamic ecosystem.

Examples

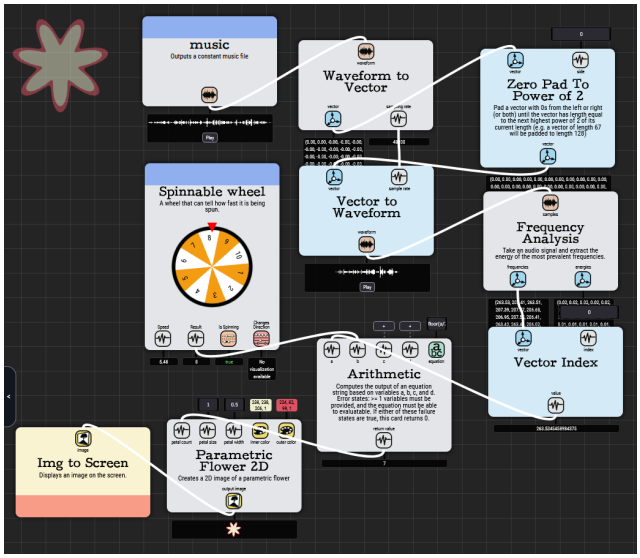
The raison d'être of our system is the design of graphs for creative expression. In this section, we want to showcase a variety of different graphs to demonstrate the range of artifacts that can be designed with our system. Figure 6 shows four example graphs and a preview of their output, where applicable. These graphs cover a wide range of data being processed, from images being masked, converted to heightmaps, or posterized, to audio-files driving parametric flowers. We also want to note that due to the arithmetic card and other capabilities of the system, it is possible to encode quite complex structures in our graphs, like the simple transformer model seen in figure 7 (with some modest custom additions to handle token-encoding).

Limitations

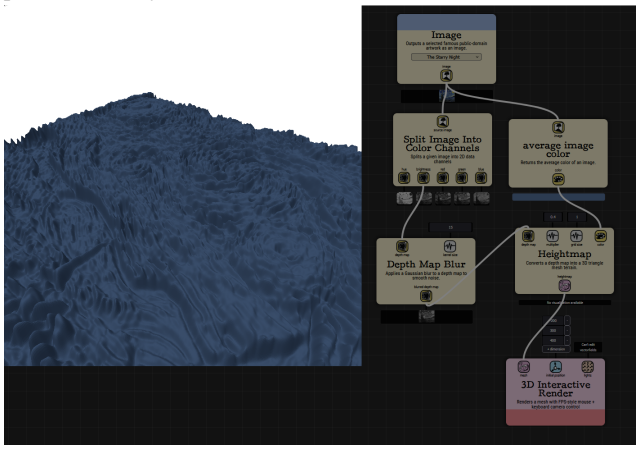
While our system enables a wide range of creative expression, several limitations exist within our work. As the most



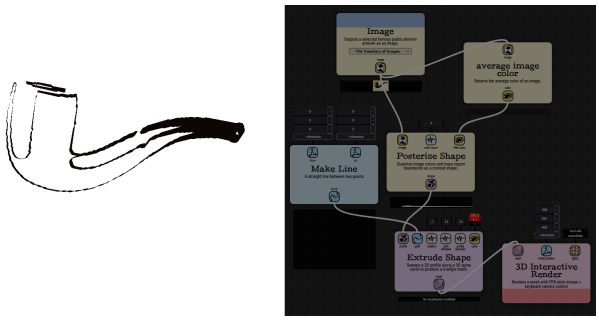
(a) Complex image masking and overlay operations can be composed arbitrarily.



(b) A parametric flower with a number of petals that depends on the predominant frequency in a given music file and the value the user spins on the wheel.



(c) Image color channels can be used to create heightmaps which can be rendered in 3D. In this case, the user can interact with the 3D scene using standard camera controls.



(d) Image posterization produces shapes which can be extruded to form 3D structures (and definitely not a pipe).

Figure 6: Four example graphs in the Generominator.

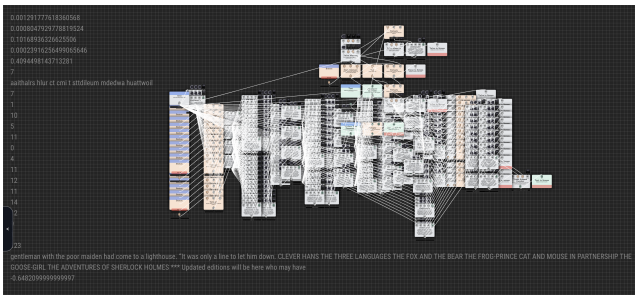


Figure 7: A transformer model implemented using the Generominator. Note that this example requires some custom nodes for token stream decoding.

notable deviation from the original Generominos, we currently do not support the *"voxel"* and *"particle"* data types from the original set. We purposefully omitted them due to their complex spatial requirements needed to represent these types of data visually, as well as the relative lack of cards that support them. Additionally, due to the nature of our system as a web-deployed application, there were several cards or modifiers with a physicality aspect described in the original work that we could not support. For example, Compton et al. (2017) describe modifiers that change which person performs a particular action, e.g. "the artist themself" or "a stranger", which does not have a meaningful interpretation in our execution environment. As another consequence of our web-based UI with the goal of being usable on a wide range of devices, we excluded cards that re-

quired specialized input or output devices (e.g. Kinect controllers). Overall, we implemented 64 of Compton et al.'s original cards, with the omissions falling into the aforementioned categories. However, we also added several cards that were necessary to construct meaningful graphs. The data types present in the Generominos are color-coded to indicate which types can be trivially converted between each other (Compton 2026). These trivial conversions themselves often need additional context, though, such as to define the sampling rate when converting an array of numbers to a waveform. To provide these transformations, and to make graph construction easier in general, we therefore also added several of our own cards to the system, bringing the total number of implemented cards up to over 100.

Additionally, due to resource constraints we were unable to conduct a formal user study to empirically evaluate artifacts. While earlier evaluations within an undergraduate student population exist for similar systems (Compton, Melcer, and Mateas 2017), we have not measured if a similar population would have differences in opinion regarding limitations in the previous study we have since resolved (e.g., missing construction cards, issues matching data types and conversion, etc.). Instead, we opted to evaluate our system against the Casual Creator design patterns, and present a wide range of potential output artifacts to showcase the flexibility of our system, and leave any claims regarding potential user interactions and perceptions of our system for future work.

Conclusion and Future Work

In this paper we have presented the Generominator, which is a web-based application that allows users to compose data flow graphs representing creative systems, based on the Generominos ideation cards by Compton et al. Each card in our system has several ports that may provide inputs and/or produce outputs, where a card can represent a data source like the keyboard, a random number, or predefined constants, a transformation, ranging from simple operations like extracting an element from a vector to more complex algorithms like a Fourier transform (Fourier 1822), sentiment analysis, or calculating Voronoi diagrams, or an output card that can show images, text, 3D meshes, etc. on the screen or play sound through the speakers. We provide a user interface that allows users to connect these cards in a guided fashion, and an execution engine to execute the graph and see the output on the screen in a live preview. Graphs can be saved, shared, and loaded as JSON files or even PNGs, allowing the easy exchange of graphs on social media.

While our implementation provides significant coverage of the originally presented Generominos, we also added several new cards that are necessary to actually be able to connect cards using different media by converting between them or extracting specific information. The way our system is designed makes it easily extensible, allowing rapid development of new cards to fill such gaps. Nevertheless, as users explore the system, the "wish list" for new cards keeps ever growing. While adding each card is straightforward for a developer, exploring how the creation of new cards could be made more approachable is an exciting avenue for future work. Additionally, our current approach is reliant on the UI for users to be able to preview their cards, but we would like to explore more direct ways for how users can share the artifacts they create as well.

Acknowledgements

The authors would like to extend special thanks to Kate Compton for creating the Generominos in the first place, and graciously providing additional context and feedback during the preparation of this paper for publication.

References

Compton, K., and Mateas, M. 2015. Casual creators. In *ICCC*, 228–235.

Compton, K.; Kybartas, B.; and Mateas, M. 2015. Tracery: an author-focused generative text tool. In *International Conference on Interactive Digital Storytelling*, 154–161. Springer.

Compton, K.; Melcer, E.; and Mateas, M. 2017. Generominos: Ideation cards for interactive generativity. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 13, 49–53.

Compton, K. 2026. Personal communication.

Csikszentmihalyi, M. 1990. *Flow: The Psychology of Optimal Experience*. New York: Harper & Row.

Eger, M. 2022. Instant architecture in minecraft using box-split grammars. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 1–9.

Fourier, J. B. J. 1822. *Théorie analytique de la chaleur*. Firmin Didot.

Graves, J.; Royer, K.; Smith, G.; and Sullivan, A. 2021. Procedural patchwork: Community-focused generative design for quilting. In *Proceedings of the 13th Conference on Creativity and Cognition*, 1–3.

Heisserman, J. 1994. Generative geometric design. *IEEE Computer Graphics and Applications* 14(2):37–45.

Hutchins, E. L.; Hollan, J. D.; and Norman, D. A. 1985. Direct manipulation interfaces. *Human-Computer Interaction* 1(4):311–338.

Kahn, A. B. 1962. Topological sorting of large networks. *Commun. ACM* 5(11):558–562.

Kaufman, J. C., and Beghetto, R. A. 2009. Beyond big and little: The four c model of creativity. *Review of general psychology* 13(1):1–12.

Kreminski, M., and Mateas, M. 2021. Reflective creators. In *ICCC*, 309–318.

Kreminski, M.; Dickinson, M.; Osborn, J.; Summerville, A.; Mateas, M.; and Wardrip-Fruin, N. 2020. Germinate: A mixed-initiative casual creator for rhetorical games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, 102–108.

Masood, Z.; Usama, M.; Khan, S.; Kostas, K.; and Kaklis, P. D. 2024. Generative vs. non-generative models in engineering shape optimization. *Journal of Marine Science and Engineering* 12(4):566.

Mathewson, K., and Mirowski, P. 2017. Improvised theatre alongside artificial intelligences. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 13, 66–72.

Mazeika, J., and Whitehead, J. 2017. Solving for bespoke game assets: Applying style to 3d generative artifacts. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 13, 73–79.

Morrison, J. P. 2011. *Flow-based programming: a new approach to application development*. Unionville, Ont.: J. P. Morrison Enterprises, 2. ed edition.

Norman, D. 2013. *The Design of Everyday Things*. New York: Basic Books. Revised and expanded edition.

Schön, D. A. 2009. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books.

Secretan, J.; Beato, N.; D'Ambrosio, D. B.; Rodriguez, A.; Campbell, A.; Folsom-Kovarik, J. T.; and Stanley, K. O. 2011. Picbreeder: A case study in collaborative evolutionary exploration of design space. 19(3):373–403.

Smith, G.; Whitehead, J.; and Mateas, M. 2010. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, 209–216.

Sood, A.; Llano, M. T.; and McCormack, J. 2025. Do conversational interfaces limit creativity? exploring visual graph systems for creative writing. In *ICCC*.

Summerville, A.; Martens, C.; Samuel, B.; Osborn, J.; Wardrip-Fruin, N.; and Mateas, M. 2018. Gemini: Bidirectional generation and analysis of games via ASP. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 14, 123–129.

Tuffs, H. 2017. Poetry generation. In *Procedural Generation in Game Design*. AK Peters/CRC Press. 209–214.