Group Theory for Procedural Content Generation: Towards Generating Objects from Mathematical Description

Jonas P. Knochelmann,[1] **Rogelio E. Cardona-Rivera,** [1, 2]

[1] Kahlert School of Computing
[2] Entertainment Arts and Engineering Program
University of Utah, Salt Lake City, UT, USA
jonas.p.knochelmann@gmail.com, r.cardona.rivera@utah.edu

Abstract

Despite the highly technical nature of Procedural Content Generation (PCG), the holistic study of the discipline is minimal and qualitative. We argue that this gap exists because there is no formal framework to talk about PCG artifacts and algorithms and propose the mathematical field of group theory to serve as such a framework. Group theory is a well-established discipline that has been embraced in chemistry, physics, and art, with tools for analyzing, combining, and generating objects based on their structure. We outline a specific method for applying group theory to PCG and explore a number of case studies in the hopes of developing a more unified formal framework for future study.

Introduction

Procedural Content Generation (PCG) is a prevalent aspect of interactive digital entertainment generally categorized as a subdiscipline of Artificial Intelligence (AI). Unlike other areas of AI research, however, PCG does not have a solid formal foundation. That is, there does not exist any standard mathematical language to talk about PCG algorithms or artifacts. This makes discourse between PCG practitioners difficult when they venture outside of standard techniques, and it limits the impact of general-purpose analysis methods.

In particular, there exists no standard method to describe a PCG artifact in a way that lends itself to being generated. This creates a link between an artifact and the details of the algorithm that generated it. This is a problem because two PCG practitioners might use completely different algorithms to create the same artifact, making communication between the two unnecessarily difficult. Further, the algorithms will likely differ in their parameters, expandability, and possibility space. The end result is that each practitioner is ultimately limited by the arbitrary specifics of their implementation and that future work has a limited ability to take advantage of past work.

Our contribution is to propose and defend group theory as a mathematical language for describing and generating PCG artifacts. In order to do this, we first expound on the unique qualities of PCG, and why group theory is well suited to handle them. We then outline a specific framework for representing artifacts, and describe three common PCG artifacts: Recursive geometry (Sierpiński Triangle), Perlin noise, and mazes, and how we can express them using this framework. As part of this, we discuss what insights group theory gives into each artifact's structure. We conclude by showing how such a framework can be implemented. Although we do not have sufficient evidence to claim that group theory is a *practical* formalism for most purposes, we can demonstrate that group theory is robust enough to express *any* artifact, even if such expression is obtuse.

Related Work

Rabbii and Cook's recent work is perhaps most related to ours (Rabii and Cook 2023). It proves a connection between PCG and the notion of Kolmogorov Complexity from information theory. This work relates in that it applies a previously unrelated area of mathematics to the holistic study of PCG. However, their work deals heavily with the possibility space and the complexity of artifacts, whereas ours deals with their structure. Furthermore, their work focuses heavily on analyses, whereas ours is more concerned with generation.

There does exist a reasonable body of work for the analyses of PCG generators. Notably, Expressive Range Analyses which visually plots the possibility space of a generator (Smith and Whitehead 2010; Summerville 2018). There have also been works studying the quality of generator output (Summerville et al. 2017; Mariño, Reis, and Lelis 2021). Though these works provide general-purpose methods for analyzing PCG generators and artifacts, they do not heavily use mathematical formalisms nor do they provide new methods for generation or description.

Background and Overview

In this section, we enumerate the qualities of PCG and see what group theory does to address them. We then describe our formal method for describing PCG artifacts with groups.

Unique properties of PCG

Procedural content generation is not a well-defined problem. However, we can identify several unusual features of algorithms used for PCG. We will use these qualities to defend group theory as a useful tool for PCG.

Output focused PCG algorithms are concerned with output far more than algorithms for other problems. This is because most problems have a brute-force solution, but PCG problems normally do not. This makes PCG artifacts algorithm-agnostic in principle, in that many algorithms can produce the same artifact.

Constrained PCG artifacts often have unusual combinations of constraints. There are visual constraints that only need to be adhered to heuristically, technical constraints like requiring non-intersecting geometry, and structural constraints like requiring that a generated level is solvable.

Multi-leveled Relatedly, PCG artifacts have complexity at multiple levels. This can be seen in recursive generation techniques, and when multiple levels of abstraction are used, like generating a graph before geometry.

Simulative PCG is usually trying to perform a limited simulation. This can either be the simulation of human creativity, or a natural phenomenon.

Group Theory for PCG

Group theory is the field of abstract algebra that studies groups. A group, simply speaking, describes all the symmetries of a particular mathematical object. In this case, a symmetry is any action that can be taken on an object that leaves the object looking the same. For example, the group for an equilateral triangle would include the action of rotating the triangle $1/3$ of a revolution, and of flipping the triangle left-to-right.

Group Theory for Generation One can see how such a description of symmetries might be used to generate a symmetric object. Imagine drawing something on transparent paper and then photocopying it. Then imagine laying one copy on top of the other, rotating $1/3$ of a revolution, and photocopying the two. We can repeat this process again, and do the same for flipping the paper over. No matter what was initially drawn on the paper, the final image to come out of the photocopier will have the same symmetries as an equilateral triangle. See Figure 1.

Symmetry and Regularity The strength and the difficulty of group theory comes from realizing that an "action" can be any well-defined function on an artifact. Imagine, for instance, a complex, pseudo-random function that transforms the branches of a tree into one another. With a broad enough conception of "action", any kind of regularity at all can be a symmetry. Any compressible object will have some kind of regularity (Kolmogorov 1998), and thus a group could be created to describe the symmetries of *any* object representable on a computer.

Fitting the Features of PCG Remembering the qualities of PCG we saw earlier, we can use informal arguments to see how group theory handles them:

- Group theory requires precisely defining artifacts and not algorithms, and so it is *output focused*
- Groups are expressed in terms of what must be true about them, and so they are *constrained*

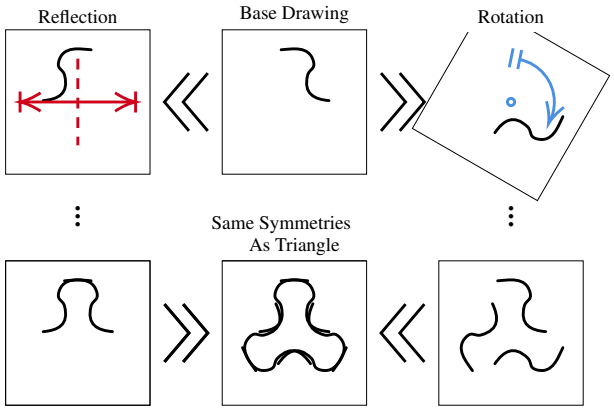

Figure 1: The base drawing (top center), can be repeatedly rotated (right) and/or reflected (left), then combined to get a shape with the same symmetries as a triangle (bottom center).

- Group theory comes with operations on groups, such as multiplication. This lets one combine the symmetries of two groups, making the regularities *multi-leveled*
- Though less well suited, group theory is used to understand low-level natural phenomena in chemistry and physics (Cotton 1963; Noether 1918). Moreover, living things have a tendency towards symmetry (Polak and Trivers 1994). Thus, group theory has the potential to be *simulative*

Group Theory in other Fields As stated, group theory has applications in physics and chemistry, but it is also relevant in music and art (Zhang 2009; Hart 2021). For physics, groups formally describe processes. For chemistry, it lets complex structures be understood. Music deals with cyclic patterns and art is concerned with the aesthetic qualities of symmetry. PCG deals frequently with understanding processes, complex structure, and aesthetics, and group theory lets us handle all of them in a unified framework.

Formalisms for PCG

To understand how to take advantage of group theory formally, we'll start with the standard mathematical definition of a group. In the following section, we use mathematical language that may be foreign to most readers. As such, the subsequent sections should be understandable with only a high-level understanding of our formalisms.

Definition 1 (Group) A non-empty set S and an operation $a \times b$, that combines any two elements from S to produce a third, in such a way that the operation is associative, every element x has an inverse x^{-1}, and there exists an identity element e.

Every group has a subset called its set of generators; a notion we'll be using later.

Definition 2 (Generators) A group G's set of generators, denoted $\text{gen}(G)$, is the smallest subset of G such that every element of G can be produced with some combination of the elements of $\text{gen}(G)$ using the group operation.

These definitions use the language of abstract algebra, which may not be familiar to most readers. We have therefore created an equivalent definition that uses functional language. This will hopefully be more understandable, but it also lends itself to a straightforward implementation in code.

Definition 3 (Group - Functional) A non-empty set of functions S, such that for $f, g \in S$, $f(g) \in S$, there exists an identity $e \in S$ such that $e(f) = f$, and there exists an inverse $f^{-1} \in S$ such that $f^{-1}(f) = e$.

For the definition above, each function corresponds to one element of the group, and $a(b)$ is equivalent to $a \times b$ in standard notation.

For mathematicians, groups are fundamentally symbolic, sometimes with corresponding actions. However, it is more useful for us to start with actions and imply a group from them. For this, we need a domain D where every element of the group is an element of D. We then need a combination function Δ that combines two domain elements such that the object described by our group is equal to the combination of all elements of the group.

Definition 4 (Artifact group) A group G, a domain D where $G \subset D$, a commutative combination function Δ where $\Delta(x, y) = \Delta(y, x) = z$ for $x, y, z \in D$, and a target artifact A where $A \in D$ and:

$$\Delta(a_1, \Delta(a_2, \Delta(...\Delta(a_{n-1}, a_n)...))) = A \text{ for all } a_i \in G$$

We can add a bit more structure for a more restrictive notion of "action" that is easier to understand for complex symmetries. Imagine trying to find the actions associated with the symmetries of a sliding puzzle. It is easiest to think of the object as being made out of smaller, *atomic objects*. We then pick one such atomic object as our *base object* and find all the ways to transform it to line up with other atomic objects. Each such transformation is an element of the group and combining two elements by the group operation combines the transformations. This way, the combination function Δ needs only add objects to the domain.

Definition 5 (Atomic Object Group) An artifact group G and a base object O where $O \in A$ and $O = e \in G$, such that every $x, y \in G$ are transformations of O, and the element x operated with y is O transformed by x and then y.

Using these definitions, one can exactly describe an artifact by implying a group structure in a way that lends itself to generation and analyses. To do this, we need a domain, a base object that will be part of the final artifact, and a list of actions that transform the object. The list of actions act as the generators when applied to the object and imply a group if they follow the group axioms. We can use a *Group Theoretic Artifact Table* (GTAT) to concisely show all this information. See Table 1.

One last notion we can take advantage of is that of group multiplication. This is an operation in group theory that takes groups G_1 and G_2 and produces a third group of order $|G_1| \times |G_2|$ called a "direct product". In terms of actions, it can be understood as any actions from G_1 plus any actions from G_2.

Artifact Name		
$\boldsymbol{a_1}$ Detailed description of the action associated with element a_1		
$\vdots$		
$\boldsymbol{a_n}$ Detailed description of the action associated with element a_n		
Group	Object	Domain

Table 1: An example GT artifact table.

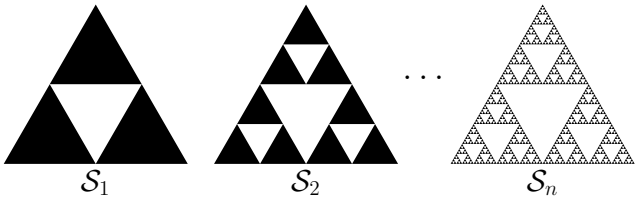

Figure 2: The varying degrees of Sierpiński triangle.

Definition 6 (Direct Product) The direct product of groups G_1 and G_2 is denoted $G_1 \times G_2$ and equals:

$$\bigcup_{\forall a \in G_1} \bigcup_{\forall b \in G_2} (a, b)$$

Case Studies

In an effort to demonstrate the effectiveness of group theory for PCG, we have gone through the process of representing a number of common PCG artifacts using the formalisms described previously.

For each artifact, we describe exactly what object we are trying to generate, then describe the regularities of the object and how those can be translated into actions that act as generators. We then show the GTAT that results, and discuss any insights resulting from the implied group.

Sierpiński Triangle

A Sierpiński Triangle is a recursive fractal object composed of triangles made of smaller triangles (Erickson 2011). Sierpiński Triangle are an example of recursive geometry, a useful class of PCG artifact.

Artifact Description A Sierpiński triangle is usually thought of as an infinite shape, but we can generalize it to a family of finite shapes denoted $\mathcal{S}_1, \mathcal{S}_2, \ldots$, where $\mathcal{S}_1$ is a triangle is made up of 3 smaller triangles, $\mathcal{S}_2$ is a triangle made up of 3 smaller copies of $\mathcal{S}_1$, and so on. See Figure 2.

Symmetry Description To decide the symmetries we want to be captured in our group, we can look at our target artifact and how it can be transformed while looking the same. Observe that $\mathcal{S}_n$ can be rotated by $1/3$ of a revolution, and so can each of the three $\mathcal{S}_{n-1}$, and so on.

Thinking in terms of atomic objects, we'll use the indivisible triangle at the top of $\mathcal{S}_n$ as our base object. Rotation about the center of $\mathcal{S}_n$ lines it up with another triangle, so

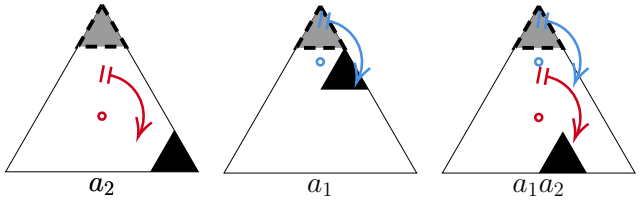

Figure 3: The symmetries of $\mathcal{S}_2$ we've decided to preserve; rotating an indivisible triangle about the center of $\mathcal{S}_2$ and the center of the top $\mathcal{S}_1$. Observe the action of a_1 operated with a_2.

does rotation about the center of the topmost $\mathcal{S}_{n-1}$, and so on. We'll call these actions $a_n, a_{n-1}, \ldots, a_1$.

If we apply a_n and then a_{n-1}, it is important that we apply the smallest rotation first. This means that any element of our final group can be described as a tuple of n elements, being the degrees of rotation at each scale. Therefore, we can find the groups generated by a_n, by a_{n-1}, and so on, taking the direct product of all of them. An element would therefore take the form $(x_n, x_{n-1}, \ldots, x_1)$, and could be made by applying each of those rotations to the base object in order. The artifact could be generated by combining every such element with the combination function. See Figure 3.

These actions obey the axioms of group theory. Associativity comes for free with our functional group definition, and the base object is our identity by definition. We can be confident of invertibility by seeing that every time a generator is applied, we can unambiguously perform the inverse of that generator, in this case, a counterclockwise rotation.

GT Artifact Looking at the GTAT for S_n (Table 2), notice the names of each action. Here, elements are meant to be combined component-wise, so (a_1, e, e) operated with (e, a_2, e) results in (a_1, a_2, e), recalling that e is the identity element.

The group implied by these actions is made of many *cyclic groups* of order 3. This family of groups has the property of being *abelian*, meaning it is commutative. This is interesting because abelian groups are generally the simplest to analyze and construct (Carter 2009). Namely, the group for $\mathcal{S}_n$ is the *elementary abelian group* of order 3^n, which is equivalent to the group for an n-dimensional vector space over the field of 3 elements (Tisza 2023). This equivalence may be meaningless, but if it has any significance, it would have been invisible without the application of group theory.

Perlin Noise

Perlin noise is a particular type of random continuous gradient that can be generated in any number of dimensions (Perlin 1985). We are using the term "Perlin noise" only because it is colloquially recognizable, as we are not concerned with any particular noise algorithm.

Artifact Description The use case we want to generate is 2D noise, layered at multiple scales. This technique produces organic-looking images. These layers are typically referred to as "octaves", and so we can refer to an n octave

$\mathcal{S}_n$ Sierpiński Triangle		
$(\boldsymbol{a_1}, \boldsymbol{e}, \ldots, \boldsymbol{e})$ Rotate about center of the topmost S_1 by $1/3$ of a revolution $\vdots$ $(\boldsymbol{e}, \boldsymbol{e}, \ldots, \boldsymbol{a_n})$ Rotate about the center of S_n by $1/3$ of a revolution		
$C_3 \times \cdots \times C_3$ for n C_3s	Topmost indivisible triangle	Triangles on a plane

Table 2: The GTAT for an $\mathcal{S}_n$ Sierpiński triangle.

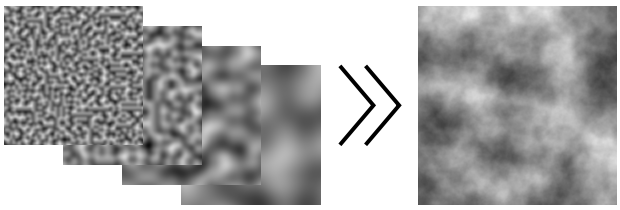

Figure 4: The construction of a 4-octave noise. Each layer is sampled at a different scale, as specified by the symmetries of $\mathcal{N}_4$.

noise as $\mathcal{N}_n$. See Figure 4.

Symmetry Description Compared to the Sierpiński triangle, noise is highly continuous and random, making it more difficult to fit into the discrete, symmetric world of group theory. We thus decided to preserve the regularity of scale.

More exactly, if we consider this layered noise to be composed of individual noise objects, each object can be scaled up (and reseeded) multiple times and then scaled down when it reaches the maximum size. Notice that $\mathcal{N}_n$ is ambiguous, as no information is given about how each octave is scaled. This, however, has no impact on the group structure.

This action satisfies the group axioms easily for the same reasons as the previous artifact.

GT Artifact Observe that the group in Table 3 is the cyclic group of order n. This is the simplest possible group of order n and is abelian. This is directly reflective of the simple nature of the regularity we decided to preserve and shows our reliance on the complexity of our base object.

Notably, this choice of representation does not give any insight into the noise algorithm itself. However, we argue that our inability to represent the noise algorithm group theoretically is a direct indication that the algorithm is better understood as an independent procedure. This is similar to how it is easier to draw a triangle than to represent the triangle with symmetries. Further, the magic of group multiplication ensures that if a useful way to represent noise was found, the resulting artifact group could be multiplied with our scaling group to achieve the same effect.

Using group multiplication We can achieve an interesting result by multiplying an $\mathcal{S}_n$ group with an $\mathcal{N}_m$ group. The images in Figure 5 may not be particularly useful, but

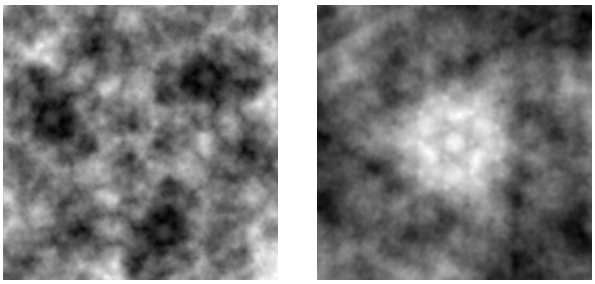

Figure 5: The artifact generated by the direct product of an S_3 group by an N_3 group, with the base object the of former.

$\mathcal{N}_n$ Noise		
$\boldsymbol{a_1}$ Scale by x, if scale is x^n, scale to 1		
C_n	Noise at scale 1, seeded by scale	Infinite height map

Table 3: The GTAT for an $\mathcal{N}_n$ noise.

the ease with which they were created shows the potential of group multiplication as a tool for PCG.

Mazes

Mazes vaguely refer to a wide range of visual-spatial navigation puzzles. As such, there are a plethora of algorithms for generating mazes subject to reasonably complex spatial and structure constraints (Buck 2015).

Artifact Description We've landed on the following definition by looking at what is often expected of maze generation algorithms: a grid of squares, each of which is connected to one of its neighbors such that the resulting graph is a tree. We can treat each cell spatially, as a square with a path extending on one side in such a way that it touches its neighbor. This way, a maze is described by the location and orientation of each of its cells.

Symmetry Description Observe that a maze may be subdivided into a number of paths with no branches ending in dead ends. Each cell can be translated and rotated to line up with the next cell of the path it falls in, cycling to the beginning if it is at the end. See Figure 6.

For a cell c in the maze, let $\text{par}(c)$ give that cell's parent (the cell it is rotated towards). Let $\text{kid}(c)$ give the set of cells with c as their parent; the children of c. For the symmetries of the maze to be well defined, we must define $\text{next}(c)$ to give the next cell in c's path:

$$\text{next}(c) = \begin{cases} c \text{ for any } c \in \text{kid}(c) & |\text{kid}(c)| > 0 \\ \text{start}(c) & |\text{kid}(c)| = 0 \end{cases}$$

$$\text{start}(c) = \text{par}^p(c) \text{ where } p \text{ is largest integer}$$
$$\text{where next}(\text{par}^p(c)) = \text{par}^{p-1}(c)$$

The base object for our group will actually be a set of cells, so let the base object for path m be denoted O_m. O_m can be any cell that does not belong to another path, and $\text{par}(O_m)$ must be part of a path with $|\text{kid}(\text{par}(O_m))| > 1$ (except for the entrance to the maze).

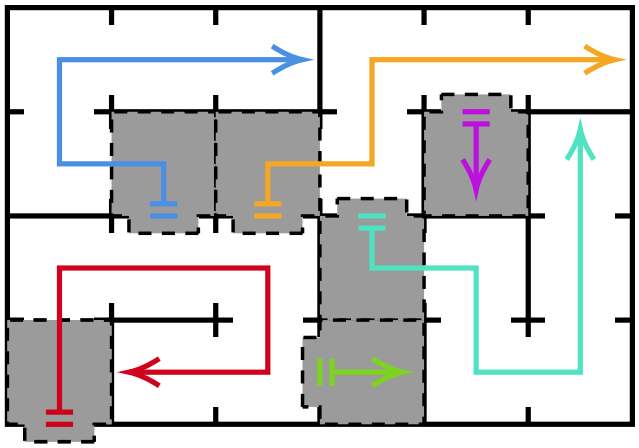

Figure 6: The symmetries for an $\mathcal{M}_{6,6\times4}$ instance. Each cell, represented as a square with a path on one side, belongs to one path. The base objects in grey can be translated and rotated along their paths.

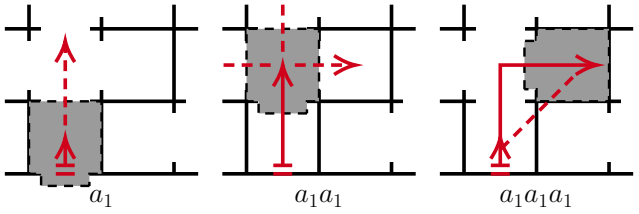

Figure 7: Repeated application of a_1 moving O_1 along a path. The dashed lines indicate the possible next cells at each step.

Note that we are dealing with functions here, so "any" can be pseudo-random, but it must always give the same answer.

We can now define exactly what an element in our group does. Let the element of the group that moves path m be denoted a_m. Let $\text{move}(c, d)$ denote a transformation that lines up cell c with cell d. The transformation for a_m therefore is $\text{move}(O_m, \text{next}(O_m))$. See Figure 7.

Using this action requires that we know the number of paths in advance. An n path maze on an $x \times y$ grid, therefore, is denoted $\mathcal{M}_{n,x\times y}$. The group for $\mathcal{M}_{n,x\times y}$ has n generators, each of which is moving the cells of a path in the way described.

Since each of these actions is cyclic and the actions do not impact one another, they follow the group axioms. Notably, if we didn't specify the number of paths and used an action to switch between paths, the axiom of invertibility would be violated.

These actions are well defined on a maze that already exists. However, they also work on a maze that is being created. $\text{kid}(c)$ for instance, might take a secret parameter μ that denotes the cells already known to be in the maze. Similarly, when the action a_m is applied for the first time, it would

$\mathcal{M}_{n,x\times y}$ Maze		
$\boldsymbol{a_1}$ move$(O_1, \text{next}(O_1))$ $\vdots$ $\boldsymbol{a_n}$ move$(O_m, \text{next}(O_m))$		
$C_{p_1}\times\cdots\times C_{p_n}$ such that $p_1+\cdots+p_n = x\times y$	A set of n cells at the beginning of each path.	$x \times y$ grid of cells

Table 4: The GTAT for an $\mathcal{M}_{n,x\times y}$ Maze.

actually create the base object in accordance with the rules.

GT Artifact This artifact has a complex set of actions associated with it. However, the group these actions imply, while more complex than previous artifacts, is rather simple as far as groups go; it is still abelian. See Table 4. This may be an indication that the structure of a maze is fundamentally simple, but that it is made complex by spatial constraints that affect the appearance but not the structure.

Another consequence of this description is the need to specify the number of paths. This may appear undesirable, however, the number of paths has a direct influence on the qualitative structure of the resulting maze.

Also, observe the details of the implied group. It is a multiplication of cyclic groups again, but it is not directly determined by the n, x, and y parameters of the artifact. This means that the details of the algorithm that generates the artifact *will* impact the group. This, however, can be seen as a result of the random nature of mazes. Notably, the maze is guaranteed to preserve the symmetries we've specified, so if an implementation produces unsatisfactory artifacts, it may indicate that this specification of symmetries is inadequate.

Implementation

One of our goals in this effort has been to separate an artifact from its algorithm. Still, there must be some method of instantiating an artifact described by a group.

Interfaces

To minimize implementation dependence, we have first designed interfaces, and then an algorithm that fits them.

Domain A domain implementation needs to be able to combine two domain elements. See Listing 1. Domain elements have no requirements, except that they be consistent.

Groups Two objects are needed to implement a group: group elements and the group itself. Group elements must implement an operation that takes a group element and produces another, obeying the axioms of group theory. It must also provide its domain element. The group needs to provide its identity element and a list of generators. See Listing 2.

You may observe that we do not require inverse elements. This is because inverses can be tedious to implement, and are not needed in the current generation algorithm. However, it is important that an inverse *could* exist. That is, given elements `e1` and `e2`, one could unambiguously find the element `e3` where `e3.operate(e1)` equals `e2`. This restriction enforces the group structure that we take advantage of.

Listing 1: Domain Interface

```
interface Domain
    combine(
    e1: DomainElement, e2: DomainElement
    ) -> DomainElement
```

Listing 2: Group Interfaces

```
interface GroupElement
    operate(
    e: GroupElement) -> GroupElement
    domainElement() -> DomainElement

interface Group
    identity() -> GroupElement
    generators() -> GroupElement[]
```

Generation Algorithm

Given interfaces as defined above, the actual generation algorithm is possibly the simplest aspect of this research. Recall that our groups are defined such that combining each element of the group under the domain combination function yields the desired artifact. We, therefore, need to find every element in the group.

Though we have not explicitly mentioned it, groups have a very natural graph structure in a *Cayley Diagram*, where the nodes are group elements and the paths are generators applied from each element. See Figure 8. With this implicit graph in mind, the artifact can be generated with a depth-first search. See Listing 3. Notably, however, this procedure is only guaranteed to visit every element and terminate if the group axioms are followed.

Some extra complications can occur: the implicit nature of the groups mean that some generators may not be defined until some of the artifact is already generated, as is the case with our maze. Further, always applying generators in the same order can result in strange-looking artifacts. These problems can be fixed with multiple passes and randomization, but such implementation is excluded for brevity.

Future Work

Our feeling has been that the artifacts we've explored are only scratching the surface of what could be done with these principles. In future work, we see ourselves exploring more complex and specific artifacts in areas such as narrative and graphics.

In addition, our tooling has limited our capabilities. With a more thorough understanding of what we require from a group theory-PCG library, a more detailed and robust system could accelerate our work substantially. Such a library may also be made publicly available.

The greatest obstacle we've encountered in this research, and the greatest obstacle we foresee for future work, is the

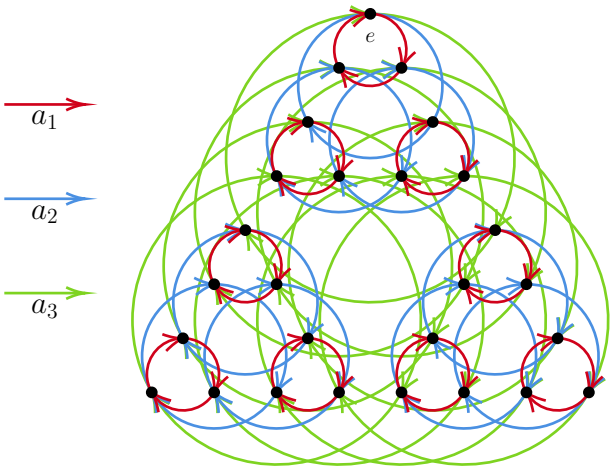

Figure 8: The Cayley diagram for $\mathcal{S}_3$. Each node represents an element of the group, and each edge color shows the result of operating a generator with that element.

Listing 3: Generator Algorithm

```
generate(g: Group)
    frontier = [g.identity()]
    elements = []

    while frontier not empty
        c = pop(frontier)
        add c to elements
        foreach e in g.generators()
            n = c.operate(e)
            if n not in elements
                add n to frontier

    return elements
```

extreme difficulty of coming up with group representations of PCG artifacts. This problem would likely increase with more substantial artifacts with more constraints. Of utmost value, therefore, would be a standard method allowing humans to take a desired artifact and derive the actions that would constitute its generators. Such a method should be understandable by those without knowledge of group theory, or the inner workings of whatever algorithm powers the generation itself. We believe the existence or complexity of such a method would ultimately determine if our research can have any widespread practical use.

Conclusion

This paper has demonstrated a novel method for describing, analyzing, and generating PCG artifacts through the mathematical study of group theory. We began by justifying and contextualizing the method and then applied the method to a number of case studies.

The research that led to this paper began with a vague notion: group theory has the power to make the lives of PCG practitioners and researchers easier. This can be stated in a slightly less vague conjecture:

Conjecture 1 (The GTPCG Hypothesis) *There exists a* deep *and* useful *connection between group theory and procedural content generation.*

Although it is impossible to resolve this conjecture with mathematical certainty, we can imagine what would be necessary to have a practical confirmation:

- A specific method for applying group theory to solve PCG problems
- A number of successful research efforts made possible by applying group theory to PCG
- A number of commercially successful products taking advantage of group theory for PCG

Conversely, the conjecture could be practically disproved after repeated failed attempts at these points. In this regard, we believe we have made the very first steps toward a positive resolution.

Regardless of the truth of the conjecture, using our method has spurred novel ways of thinking about PCG. Finding regularities in artifacts that obey the group axioms has required understanding artifacts in a holistic way we have not seen in other techniques, and the breadth of existing tools in group theory, from multiplication to public libraries of groups, have provided unique and fascinating insights. We are optimistic about the future of this research and hope this paper will be the first of many.

References

Buck, J. 2015. *Mazes for Programmers*. The Pragmatic Programmers.

Carter, N. 2009. *Visual Group Theory*, 69–71. Mathematical Association of America.

Cotton, F. A. 1963. *Chemical Applications of Group Theory*. Wiley.

Erickson, M. 2011. *Beautiful mathematics*, 20. Mathematical Association of America.

Hart, S. 2021. The Art of Group Theory & The Group Theory of Art.

Kolmogorov, A. 1998. On tables of random numbers. *Theoretical Computer Science*, 207(2): 387–395.

Mariño, J.; Reis, W.; and Lelis, L. L. 2021. An Empirical Evaluation of Evaluation Metrics of Procedurally Generated Mario Levels. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Noether, E. 1918. Invariante Variationsprobleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1918: 235–257.

Perlin, K. 1985. An image synthesizer. In *ACM Siggraph Computer Graphics*.

Polak, M.; and Trivers, R. 1994. The science of symmetry in biology. *Trends in Ecology & Evolution*, 122–124.

Rabii, Y.; and Cook, M. 2023. Why Oatmeal is Cheap: Kolmogorov Complexity and Procedural Generation. *Foundations of Digital Games 2023 (FDG 2023)*.

Smith, G.; and Whitehead, J. 2010. Analyzing the Expressive Range of a Level Generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*.

Summerville, A. 2018. Expanding Expressive Range: Evaluation Methodologies for Procedural Content Generation. In *Proceedings of the Fourteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 116–122.

Summerville, A.; Mariño, J. R. H.; Snodgrass, S.; Ontañón, S.; and Lelis, L. H. S. 2017. Understanding Mario: An Evaluation of Design Metrics for Platformers. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*.

Tisza, L. 2023. *The n-dimensional vector space $V(n)$*, Section 1.3. LibreTexts.

Zhang, A. 2009. The Framework of Music Theory as Represented with Groups.