Lindenmayer’s Defense: 
Generating projectile patterns in a video game environment 
using L-Systems

Lindenmayer’s Defense: Generera mönster för projektiler i spel genom L-System

Christofer M. A. Malmberg
Henrik Phan
**Lindenmayer’s Defense**

Generating projectile patterns in a video game environment using L-Systems

Christofer M. A. Malmberg and Henrik Phan
Malmö University
Malmö, Sweden

**Abstract**—The potential of L-systems is explored by procedurally generating patterns for use as video game content. By procedurally generating content for video games, the development costs of game development can be significantly reduced. An artifact in the form of a tower defense game is developed and tested to evaluate the generation algorithm. The algorithm was successful in generating a wide range of pattern and user feedback indicates a high level of perceived variation. The algorithm is highly customizable and could have applications in various game content such as particle systems or weapons.

**Keywords**—procedural content generation; Lindenmayer system; mixed-initiative; video game, pattern

I. INTRODUCTION

Creating game content is a time consuming and costly endeavor, which means the amount of content available in a game is limited [1]. One way to solve this issue is to automatically generate content for the game using procedural content generation (PCG) [1]. By generating content automatically, it is possible to cut costs and development time. There are many existing examples of content generation in modern games. In the open-world sandbox game **Minecraft** [2] a whole 3D world is generated to become a playground for the player. In the action-RPG game **Diablo 3** [3] items are generated using an algorithm to give items varying attributes. There are also various established algorithms that can be used in PCG [1]. One such algorithm, is called Lindenmayer system, or L-system [4]. In games, L-systems are commonly used to generate flowers, trees and other plants. Although L-systems have been used to generate many types of content, such as city maps, 3D-models and music [5, 6, 7], there are possibly a lot more viable applications for L-systems. One possible application was presented during a meeting with Tarsier Studios, a game development studio that is located in Malmö, Sweden. The idea was to have a tower defense game where the core game mechanic was a crafting system that resulted in the towers that the player could place in the game world. The specification for a successive solution to this idea was a system that could produce unique towers that where controllable through the different crafting components. This study aims to present a possible solution to the specifications given by Tarsier studios by generating projectile patterns that can be used in a tower defense game. The solution that is presented uses a PCG method called L-Systems.

Our research questions are as follows:

Can L-systems be used to create a projectile pattern generator that:

- Allows controllability using player input while creating emergent patterns
- Produces a wide variety of patterns
- Creates visually appealing patterns.

This study uses a modified version of design research. Traditional design research identifies a problem, defines a potential solution, develops an artifact, and evaluates the artifact [8]. To save some development time, we defined use cases which we applied to our potential solutions. If the solution would fail, iterations of the potential solution were made before reapplying it to use case. If a solution passed all the required parameters the solution was then implemented in the program.

The study will focus on testing a certain algorithm in a gameplay setting. This is done by creating a tower defense game referred to as Lindenmayer’s Defense. In this game, the player uses a crafting system to build towers that are used to protect an objective against enemies. The study will not cover other workable solutions or algorithms that could be applied to this problem. The study will rely on data collected through gameplay with complementary forms.

The artifact created by this study demonstrates a unique way of creating novel content in the form of weapons by using L-systems. The solution that is presented can also be used in the creation of pattern-based particle systems, giving developers more options when creating content for their games.

II. RELATED RESEARCH

In this section, various concepts and research related to the study are introduced. Procedural content generation and mixed-initiative PCG are first introduced followed by two different PCG methods that were considered for the study. Lastly, expressive range is explained as the primary tool for evaluating the algorithm.
A. Procedural Content Generation

Procedural Content Generation (PCG) is a relatively new field of study. PCG is defined by Togelius et al. [9] as “...the algorithmic creation of game content with limited or indirect player input”. Further, Shaker et al. [1] defines content as most of what is contained in a game, excluding the game engine itself. Without PCG, creating an entire simulated 3D world with unique trees, cities and people, would require the artists to handcraft every different plant, building and person. These assets would then have to manually be placed into the world, one by one. This would take a tremendous amount of man hours. Using PCG, the same thing could be accomplished by only developing a few different algorithms for generating the 3D-models themselves and another one for placing them in the world [1]. This arguably comes at the loss of quality, depending on the type of content that is generated, since an algorithm doesn’t have the artistic sense of a person. The resources used, however, can be minimal in comparison. By continuously generating additional content, the replayability of a game can increase as well.

B. Mixed-initiative PCG

This study aims to create an algorithm that uses a combination of human and computer creativity for its content generation, this is called mixed-initiative PCG [10]. The amount of involvement between the two can vary greatly depending on the end goal of the program. It can be viewed as a spectrum, where on one end are tools that let the computer do most of the work and on the other end the human does most of the work. On the user-focused side of the spectrum lies tools like Tanagra [11], a 2D platformer level generation tool, where the computer makes sure that levels are playable and supports the user in other way. An example on the other end of the spectrum would be a level generator that only requires a small amount of input from the human. The user might specify some desired properties of a level which the generator will consider, or they might just specify a seed as an input, leaving all the work to the generator. Sentient Sketchbook [12] is another example of computer aided design tool that helps the designer to create levels that can be used in for example strategy games. It tests the maps playability automatically by evaluating the map on different gameplay properties and converting map sketches into playable levels.

C. L-systems

An L-system is a mathematical form created by the biologist Aristid Lindenmayer [4]. It consists of an initial string, called the axiom, and a set of rewrite-rules, called the grammar. The axiom is rewritten based on the grammar, usually resulting in a longer string. This is then repeated many times, each time yielding a new generation. For example, given the axiom A and the rewrite-rule A→ABA, the axiom would first be rewritten to ABA. Doing another rewrite would result in ABABABA and so on. By interpreting the string as commands for turtle graphics, it is possible to generate complex fractal images from even simple L-systems. Further expanding with bracketed L-systems, it is possible to make branching structures using brackets in the grammar for interpreting when a branch begins and ends [13].

D. Evolutionary search-based PCG

One of the solutions that was considered in this study was evolutionary search-based PCG, which is an algorithm that is similar to Darwinian evolution by using natural selection [14]. The algorithm starts with a population of individuals that are measured by a fitness function. The ones that score the highest from the fitness function get to reproduce while the lowest scoring individuals in the population get culled. Reproduction is done by combining attributes from individuals to create new ones. The process then repeats until a desirable outcome is reached. The trick is to find a fitness function that is specific enough to properly breed the correct individuals while not being too strict. If it’s too strict it can plateau too early in generation and not reach an optimal population of high scoring individuals.

Galactic Arms Race (GAR) [15], is a game created to explore evolutionary search-based PCG and the use of neural networks in content creation, specifically generating projectile behavior. It collects data about the player’s preferences as the game is being played and generates weapons accordingly. This results in generated content that is specifically tailored to each player. The players themselves, however, do not have any direct control over the type of content that is generated. This means players will not have any idea of what content they are going to get to play with. GAR showcases the effects of generating game content by using neural networks. The algorithm, called content-generating NeuroEvolution of Augmenting Topologies (cgNEAT), generates game content based on the perceived user preferences. The result is different weapons that fire projectiles which have been generated through implicitly learning how the player interacts with the game.

E. Expressive Range

When creating procedural content generators, it is important to be able to evaluate various aspects of the generator. The expressive range of a generator refers to the level of variation in the content that the generator can create [16]. A generator with a high expressive range is more likely to generate content that is unique and varied. On the contrary, a generator with a low expressive range, would generate the same content with slight variation. By measuring the expressive range, it is possible to discover what tendencies a generator might have, or if it lacks the ability to generate a certain type of content [17]. In our study, we will analyze the expressive range based on the patterns that can emerge from the generated towers. These patterns can be plotted onto a heat map to discover what type of towers our generator is more likely to create.

III. Method

A. Research Methodology

The study uses Peffer’s et al. [8] design-science research methodology in a set of 7 stages described as follows.
1) Problem identification and motivation: To find a suitable game idea that would be relevant to the current market we consulted experts at Tarsier studios, an independent development studio located in Malmö, Sweden. The resulting specification was used for identifying problems and viable solutions. The identified problem was generating varied but controllable weapons for a game based on player input. By accomplishing this, the playability of the game would increase due to the player being able to generate content suitable to their own taste. This is relevant because high-quality generated content can partially replace the work that game designers must do, resulting in more effective development of games.

2) Objectives of a solution: The idea was to procedurally generate weapons using a combination of different components. This would result in a unique weapon based on the combination used. The components should be made in such a way that they would follow some order of logic. This would mean that the player would be able to somewhat predict what the result would look like, based on the components. An important attribute is the range of weapons that can be generated. The player should be able to experiment with many different combinations and each weapon should be as unique as possible. A high degree of controllability and a wide expressive range are the desired properties. Increasing stochasticity would increase the expressive range, but it would decrease the predictability of the algorithm. This makes it difficult to get both desired attributes as an improvement in one might come at the expense of the other. Finding the right balance is therefore an important aspect of the solution.

3) Design and development: The design and development process was agile and did not follow any specific guidelines. It started with internal discussion of how the objectives of a solution could be handled and resulted in use-cases for different methods. The design of the game followed a simple tower defense formula derived from reviewing other games in the genre.

4) Demonstration: The artifact uses a mixed-initiative PCG approach with an L-system. The string that is interpreted by the L-system is built by the player using different components. The artifact can produce enough variation while still applying a strict ruleset given by the player resulting in a design that is mostly procedurally generated but has a foundation made by the player. The resulting weapons are predictable by the player but are still varied. The weapons are showcased as towers that shoot projectiles in a game created during the study. The game follows a tower defense formula where the player needs to protect an objective by placing towers that defends the objective against approaching enemies.

5) Evaluation: The artifact was presented to Tarsier studios and received feedback and suggestions for a potential next iteration. The artifact was also tested by several game development students at Malmö University. The tests were either sent out to the tester as a zip file or conducted in person at Malmö University. In both cases a file with the program, instructions and a link to an online form was included.

6) Communication: The result was discussed with Tarsier studios to see if the artifact managed to fulfill the desired purpose. The artifact was also presented in a student opposition.

7) Contribution: This research will enhance the foundation of PCG and L-systems and may result in further exploration in combining L-systems with projectile pattern development in games.

B. Data Collection

To collect data that could be used to determine whether the algorithm fulfills its intended purpose, two different strategies were used: expressive range and play testing along with forms.

1) Expressive range: The expressive range of the algorithm was gathered from a heat map constructed in the game that tracks the number of projectiles that exists on a single position within certain timeframes. The heatmap was then converted to an image and used to visualize the expressive range of the algorithm.

2) Playtest: The playtest consisted of playing the game for as long as the tester wanted with the encouragement of trying many different components. Afterwards the tester was instructed to click on the link to the online form.

3) Forms: The form was structured as an attitude form [18], each with eight questions using the Likert scale [19] with scores ranging from 1-7.

4) Use case: To determine if a possible solution was applicable to a problem, use cases were created. The use cases did not follow any specific method and was created by the developers as part of the implementation process. They consisted of a simple description of a specific problem and a potential solution.

C. Data Analysis

The mean of each question from the attitude form was calculated and used for evaluation as a measurement for how the algorithm fit the intended purpose [20]. High or low scores could indicate possible changes were needed for the next iteration. The expressive range was displayed in a heatmap and used for evaluating the amount of variety the algorithm could produce.

D. Lindenmayer’s Defense

“Lindenmayer’s Defense”, is a tower defense game that was created as an environment in which to test the algorithm. The goal is to protect the home base from enemies that approach from all sides. This is accomplished by building towers and placing them strategically on the field to shoot down the enemies before they reach the base. Towers are built by selecting up to 5 desired components, which will generate a unique tower based on the selected components. The player starts with a small amount of gold, which can be spent to purchase and upgrade towers. By defeating enemies, more gold can be accumulated. As the game progresses, the player will be
able to improve their towers and build more of them, strengthening their defenses.

An example of how the game looks during gameplay can be seen in Fig. 1. In the bottom-left corner is the component inventory. These are the components that the player can select by clicking on them. The selected components show up in the bottom-middle boxes. After up to 5 components have been chosen, the player can generate a tower by clicking on the big blue box on the right. The tower can then be placed anywhere on the playing field. Enemies will start spawning upon pressing the space bar button. Enemies will move towards the center base and self-destruct on contact with the base, dealing damage to it. When the base is destroyed, the game is lost. There is no ultimate win condition in the game. The only goal is to keep the base alive for as long as possible.

IV. RESULT

After discussing the idea of generating towers with Tarsier Studios we acknowledged two main problems that needed to be solved to produce a viable solution. The algorithm needed to be random enough to keep the towers feeling fresh while keeping the player in control. After internal discussions, we came up with two suggestions for algorithms that might fit these requirements. One of those was an evolutionary search-based algorithm that would use the player's component choices as a sort of fitness function, the other one was a bracketed L-system with a similar component idea.

To decide which of these two algorithms that would fit we made a use case and applied a small prototype for each algorithm. After evaluating the results of the search-based algorithm, we immediately realized that the player would lose a lot of control when manipulating the tower, because of the search-based nature of the algorithm. It was also difficult to implement the visual qualities of the weapons in this algorithm, which resulted in the variations being nothing more but a change in values that were mostly invisible to the player, e.g. increased damage or more health. The bracketed L-system proved very fitting after being applied in a prototype for the use case. Using this system, we realized that it was possible to generate more visually appealing towers by having the projectiles fired from the towers follow the L-system, in a similar fashion to turtle graphics. Depending on the rules used, a wide range of patterns could be generated. During our tests, we found that even simple rules could give birth to complex, but predictable behaviors, which was exactly what we needed.

A. The algorithm

To create seemingly unique towers without compromising the value of player input, a PCG algorithm called L-system was used as a solution. The L-system would create strings put together by various visual components that were selected by the player. The player would unknowingly create a string through these components that would then be used as a rule for the L-system grammar, called the X-rule.

The grammar used in our L-system consists of a set of predefined rewrite-rules, based on various commands. These commands represent many ways to manipulate a projectile, such as “turn left”, “go forward” or “split in two”. In addition to these rewrite-rules, every tower has its own unique X-rule. This X-rule is generated based on the selected components and is the starting point for the L-system. Each rewrite-rule, except the X-rule, includes a reference to itself to keep expanding through multiple generations, creating a recursive effect. This means that subsequent generations will be like previous ones, keeping in line with the desired predictability. There is also a 5% chance of mutation each time the L-system rewrites a character, to create a bit of stochasticity. This works by randomly adding a character to the string at that position.

Every component has an associated string, which is the string added to a tower’s X-rule upon creating the tower. After the player has selected their components, these component strings are added to the X-rule. The string is then expanded throughout many generations, creating the final command string. The projectiles fired from the tower would parse each command accordingly, giving birth to our “L-system towers”.

To give an example, as displayed in Fig. 1, let’s start by selecting a “spinner left” component for our new tower. It would have the associated string “LL”. This would then be
added to the currently empty X-rule, giving us X→LL. Continuing, we select the “arrow” component, with the associated string “SF”, gives us X→LLSF. Finally, we also select the “spinner right” component, with the string “RR”, giving us X→LLSFRR. Let’s expand this rule by two generations, using our rewrite rules. In this case, our rewrite rules are as follows:

F→ Ff
L→ L→F
S→ sF
R→ R++F
X→ LLSFHF

Our starting point is just “X”, so after the first expansion, we would have “LLSFRR”. Expanding again would give us “L--FL--FssF FR++F”, which is the completed string for our tower. As mentioned, most characters in this string represent a command, which will decide how projectiles fired from this tower will behave. Some characters are only used for the L-system and does not have an associated command; these are simply skipped.

In the string “L--FL--FssF FR++F”, we have the following commands:

F → move forward
L → turn left
S → turn right
s → speed up
h → turn towards target
F → move forward

Projectiles fired from this tower would therefore first make a left turn, speed up and move forward and then make a right turn. We only expanded two generations in this example for the sake of simplicity. Higher generations cause the string to grow exponentially in both size and complexity, which is the reason why they are limited to 5 generations. Another important thing to note is that the order in which the components are chosen will also affect the behavior of the projectiles. For example, picking “left” and then “right” will not give the same behavior as first picking “right” and then “left”.

In Fig. 2, generation 1, the projectile barely moves before reaching the end of the string and is terminated almost immediately. In the next generation, the effects of the components are beginning to show. For the following generation, these effects are amplified, resulting in an emergent pattern in generation 5. In the game, the towers start at generation 3 and can be upgraded to generation 5, so the player can see the effects of their chosen components immediately. For the same reason, our later analysis will not include towers of generations 1 and 2.

B. Expressive Range

Fig. 3 shows three towers that were generated by the algorithm as an example of some different varieties of towers that can be generated.

Fig. 4 shows the expressive range of generation 3, 4 and 5 towers from left to right.
that area. The array was then converted to an array of colors, creating a 2D heat map. This was done using data from 1000 towers, each with 5 randomly selected components, to include as many types of towers as possible. This was done for 3, 4 and 5 generations, with new towers each generation.

C. Forms

The data collected from the forms that were filled out by the test subjects after completing a playtesting session were used to calculate a mean used to evaluate the algorithm. The resulting means of each question can be seen in Fig. 5. To validate the statistical significance of the result a one sample t-test [20] was conducted for relevant questions. The hypothetical mean for the test was 4.0, and the number of testers were N = 30.

<table>
<thead>
<tr>
<th>Question</th>
<th>T value</th>
<th>Df</th>
<th>Sd</th>
<th>Mean</th>
<th>Two-tailed P value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The towers were random</td>
<td>0.5651</td>
<td>29</td>
<td>1.5733</td>
<td>4.1300</td>
<td>0.5763</td>
</tr>
<tr>
<td>I had control over the towers’ behavior</td>
<td>1.8935</td>
<td>29</td>
<td>1.6392</td>
<td>4.5666</td>
<td>0.0683</td>
</tr>
<tr>
<td>I understood what each component did</td>
<td>4.8321</td>
<td>29</td>
<td>1.6385</td>
<td>4.6333</td>
<td>0.6333</td>
</tr>
</tbody>
</table>

V. ANALYSIS

One thing to note with the heat map is that it only shows the pattern that emerges when there are no enemies to target. The towers in this case only shoot straight ahead. This also means that properties like homing, are not expressed in this heat map. The reason for this is that if data were gathered from a regular play session, it would be hard for it to yield any useful data, since projectile positions would be affected by enemy positions as well. Unsurprisingly, more generations cover more of the field, and creates a much greater number of projectiles. The towers that are generated can shoot projectiles in numerous different patterns. There is an obvious forward tendency, but that is to be expected since a weapon should shoot forward. In the case when the tower shoots behind itself or to the sides, it can most likely be attributed to the Spinner components which spin the projectile, releasing it in a different direction. For the generation 5 towers, many projectiles have a reach that goes beyond the screen.

Reviewing the result from the forms high scores in “Towers were random” coupled with mixed scores in “I had control over the towers’ behavior” could indicate that some players felt that they were given enough control over the tower creation while receiving a satisfying amount of variation. Furthermore, the algorithm does include a small chance of mutation to keep the towers from being completely identical, this could skew the answers in question X towards high scores. Feedback given by the participants included encouragement to increase the mutation factor since it was perceived as a fun element that added excitement to the tower creation.

VI. DISCUSSION

A. Answering the research questions

Our choice of using L-systems to generate projectile pattern in a tower defense game proved successful in generating a large expressive range for the towers. The expressive range shows a clear indication that the algorithm provides a wide variety of tower patterns. This result provides an answer to one of the research question whether this method could be used to create a wide variety of patterns. By only modifying the towers’ components, it results in a completely new pattern for the projectiles. However, these patterns are limited to the structure of the components and the commands that a projectile can take. It is partially up to the designers to figure out what kind of components will yield the best results and what kind of rules they need to get the functionality that the component is supposed to represent. Another option is to completely omit
these components and create a system that doesn’t include the use of components. They can also contain more complex grammar which will affect the outcome of the tower behaviors. The number of components that can be selected is another variable that can heavily influence the result.

Using forms to collect player feedback gave a good indication if the artifact and algorithm could fulfill the rest of the research question, however the select few questions relevant for statistical significance failed the one sample T-test. However, the fact that contradictory questions such as “Was the towers random?” and “Did you feel that you were in control of the tower behavior?” both got relatively high means could be interpreted in multiple ways. One such interpretation is that players did feel like they could control the behavior but the outcome was always new and exciting. This means that the artifact did in fact manage to generate emergent patterns. Another interpretation is that, since the towers did have a small mutation factor, players got the impression that the towers were simply randomly generated. Another question that was asked to verify the research questions was if the players found the towers visually appealing which got a relatively high mean, confirming the success of the artifact.

B. Discussing the method

Using design science to explore further uses of L-systems made it easy to iterate through different ideas and didn’t force the algorithm to be fully ready before implementation. This gave the developers a lot more freedom to try different approaches since another iteration was always an option if one approach did not seem promising.

The choice to create projectile patterns to showcase the algorithm was backed up by GAR and their choice to do the same [15]. In this study, the built towers are the weapons, while in GAR they generate new weapons as drops during gameplay. The choice proved successful in displaying a wide expressive range and made it relatively easy to build a game for testing the algorithm.

The fact that the study combines L-systems with a mixed initiative approach could be the key to its success. Creating patterns with L-systems is nothing new but being able to put so much emphasis on human input to control the output of the L-systems makes this approach more interesting. If you were to expand on this research and create a tool with a similar approach, custom components could be added on the fly along with modifiable L-systems. This could then generate anything from projectile patterns to controlled particle effect patterns which could be very useful to create powerful and unique game content.

C. Discussing other approaches

Different approaches were initially discussed and one of them was to use an evolutionary search-based approach [14]. The idea was to have the player select different components and then have a “Constructor-agent” combine these, which would determine the starting stats of the tower. The agent would then have its own fitness function and would have a limited amount of generations to evolve the given components into something that was as close to the optimal as possible. The problem that was immediately discovered was that there already was a blueprint for a perfect tower with the constructor-agent, which was decided by its fitness function. This greatly limits the possible expressive range for the algorithm, and the influence of player input. Another problem was that the genetic algorithm approach would focus too heavily on different statistics of the tower and only alter set behaviors further reducing the expressive range. This would however place a lot more emphasis on designer involvement and enable much freer handmade design in, for example, the look of the tower based on its components.

Another approach that was brought up was to use neural networks like GAR [15]. The idea was to have the tower log each successful hit from its projectiles and try to expand upon its “successful” projectiles i.e. shoot more of that kind. The risk was that this approach would not fit the game idea very well as the player has absolutely no control in what the tower would evolve into. However, this approach has the potential to create unique projectiles, as is evident in Galactic Arms Race.

VII. CONCLUSION AND FURTHER RESEARCH

Three one sample t-tests were carried out on the most relevant questions related to the algorithms controllability. Unfortunately, the result did not prove to be statistically significant. When asked to rate if they felt the towers were random ($M = 4.13, SD = 1.57$) the result was: $t(30) = 0.56, p = 0.57$. When participants were asked to rate if they felt that they had control of the towers behavior ($M = 4.56, SD = 1.63$) the result was: $t(30) = 1.89, p = 0.06$. When asked to rate if they understood what each component did ($M = 4.63, SD = 1.63$) the result was: $t(30) = 4.83, p = 0.63$.

There are a lot of variables that are open for manipulation that can greatly affect the outcomes of the algorithm. It’s up to the designers to decide how detailed each component should be. In any case this method can be very useful in creating many kinds of weapons by simply tweaking the grammatical components that are combined. By also having so much freedom in organizing each component, the number of components and the detail of each individual component, the algorithm can be either more deterministic or more stochastic.

The algorithm in this paper is demonstrated by generating towers in a defense-type game, however, the algorithm is not limited to this game genre. It could easily be applied to any type of game that uses projectile weapons. It is also not limited to 2D and could easily be adjusted for a 3D game. The algorithm could also be used for generating distinct patterns for particle generators, allowing more deterministic particle effects.

The results showcase a novel use of L-systems in generating projectile patterns. A wide expressive range was achieved using a relatively simple set of components, which indicates a much greater expressive range if considering the possible component variations. These indications coupled with the high scores in
evaluation forms show a successful application of PCG in games using L-systems.

REFERENCES


