6CCS3PRJ Final Year
Implementing Procedural Content Generation Algorithms in a Tile Map RPG in the Godot Game Engine

Final Project Report

Author: Zishan Rahman

Supervisor: Senir Dinar

Student ID: 20071291

April 20, 2023

Abstract

Procedural generation refers to content in a medium that is produced algorithmically in lieu of by hand. Most notably, procedural generation algorithms are implemented in video games, for generating levels, terrain and other game contents programmatically. This project takes some of the more prominent algorithms for procedural generation- Lindenmayer Systems, Voronoï Points, Poisson Disk Generation and Simplex Noise- and implements them in a 2D tile-map-oriented RPG-like game in the open-source Godot game engine, and compares their workings and performance. My aim with this project is to (1) increase my knowledge of procedural generation in games beyond the surface level, by going in-depth into some of the algorithms that are used, and (2) use this knowledge to implement said algorithms in a 2D tiled RPG scenario in Godot, then compare how each algorithm works and performs.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Zishan Rahman

April 20, 2023

Acknowledgements

The author of this report would like to give thanks to his supervisor, Senir Dinar, for providing him the guidance he so badly needed to make this project not only the best for his mark, but also one that he enjoyed doing very much.
He would also like to thank Kevin Lano, who helped him up when he was down and was able to offer as much additional feedback to him as he could.
He expresses his thanks also to his family and friends, both online and offline, who gave him their wholehearted support throughout his studies and his time at King's College London.
He also thanks Kenney, for providing the huge tile set asset used in all of the algorithm implementations for free (CC0-1.0).
He also thanks Alexander Gillberg, who gave him the kind permission to adapt a small part of his code for one of the implementations of his chosen scenario, as well as to add the MIT license to a fork of his code.
And, finally, his sincerest thanks are sent, from the bottom of his heart, to Shaan Vieru, to whom he lovingly dedicates this project. Shaan were so polite and humble, fun to be around and he is sure he would have loved to see the final product. Shine on, you crazy diamond, and fly high among the stars.

Contents

Chapter 1

Introduction

Procedural Content Generation, or PCG, refers to the use of algorithms and programming in lieu of human handiwork to design and implement various contents in video games, such as levels, terrains, trees and cities. A PCG algorithm is ontogenetic when it tries to produce a foreseeable end result as it goes along. For this project, several well-known ontogenetic algorithms have been implemented in a basic 2D tile-map-oriented RPG-like game, using the open-source Godot game engine, and then comparing how each algorithm carries out the creation of levels in said game, both performance-wise and comparing the kinds of level layouts generated by each algorithm. The aim here is to weigh up the best algorithm for the chosen scenario based on how similar and how different each implementation of the algorithm provides with its level layouts, and show how every algorithm was well integrated into the chosen scenario and adapt the scenario and the algorithm as needed.

1.1 Report Structure

First, there will be some background provided into the work done behind this dissertation, demonstrating some understanding of PCG in video games and eventually justifying the choice to use Godot, why a 2D tile map RPG was chosen to adapt to a PCG context and why each algorithm was chosen to implement each PCG algorithm into the defined scenario.

The main report body will go through firstly how each algorithms works, and secondly go into how they were implemented into the chosen scenario at a surface-level explanation. This report goes into further detail in the Implementation section.

The Design & Specification section will firstly go through what was sought after in every implementation of the chosen scenario, in order to be able to compare each implementation with one another and then determine how each algorithm was ranked according to the relevant criteria in the Evaluation section.

The Implementation section will go into further detail than done in the report body as to how each algorithm implemented and any code issues were worked around, including GDScript code snippets where needed.

The Legal, Social, Ethical and Professional Issues section will discuss how, firstly, any issues with any external code references used for any software artefacts were eventually worked around, and how integrity was practiced while doing so. Secondly, this section will go through the plans set out to make both the dissertation and the artefacts behind it publicly available while still taking care of any potential professional issues.

The Results & Evaluation section will go through some of the quantifiable results obtained in experimenting with theses implementations and any custom values that were set during those experiments. This report has included these tables in the Appendix, so as not to break the flow of the report itself. This section will also discuss the conclusions obtained and how each algorithm was ranked in terms of how they turned out with the given scenario, as well as how similar and different the produced level layouts were.

Finally, in the Conclusion and Future Work section, there will be a final summary of the conclusions obtained and discussed earlier, in addition to what was gained by the author of this report as a games programmer and student from this project, and where this project and its aims could be taken further.

Chapter 2

Background

For the author's BSc individual project, this paper will be researching procedural content generation (PCG) algorithms and then implementing them each in a small 3D game made with the Godot Engine (and its domain-specific GDScript language).

2.1 Procedural Generation: Background

Procedural content generation (usually referred to as simply "procedural generation") refers to the creation of levels and other game objects programmatically and algorithmically, in lieu of a human being doing all the work. While procedural generation algorithms can be used to generate a myriad of things, from textures (for things like trees and clouds) to music ("generative music," as coined by legendary musician Brian Eno), by far its most common context is in automated level design, generating level layouts algorithmically in lieu of work from level designers. Game developers may opt to use procedural generation to save time and money designing levels or show off technical prowess in their games.

Procedural generation in video games has a rich history. Pioneering games such as Rogue (1980) took direct influence from tabletop role-playing games such as Dungeons and Dragons[11], and thus had a player navigate a randomly-generated world that expanded further as they went on. Such games spawned the *roguelike* and *roguelite* genres, which experienced immense popularity in the last decade. In the realm of first-person shooters, 2004's .kkrieger, as seen in Figure 2.1, used procedural generation to create intricate 3D levels and fit them all into a game that takes up just 96 kilobytes of space.

Figure 2.1: The game .kkrieger, which uses procedural generation to design maps while keeping the game at a 96 kilobyte file size.[19]

Other games that use procedural generation in its levels include Elite (originally published in 1984), Elite: Dangerous (2012), Minecraft (2009), No Man's Sky (2012) and Spelunky (2013). The latter game's use of procedural generation has notably been covered by video games journalist Mark Brown in a YouTube video.[4]

Figure 2.2: The roguelike game Spelunky, which uses procedural generation to build intricate levels for the player character to explore. Source: https://store.steampowered.com/app/239350/Spelunky/

In many cases, these games end up having a **large** number of different environments that each game could generate for its players. However, by procedurally generating them upon the *loading* of the game level, in lieu of loading a layout from disk, they can save a lot of space (albeit with a considerable need for processing power, depending on the game's and algorithms' performance), as seen in Figure 2.1.

Using one or some different procedural generation algorithms, such as the use of Perlin, Simplex or other noise, Voronoï disks and also poisson disk generation, among others, games can load a seed to randomly generate a level every time it is played, meaning no two playthroughs of a game with procedurally generated content are ever the same.

2.2 Justifying The Paper's Choice of Engine: Godot

While a myriad of resources exist for procedurally generated game contents exist for Unity and Unreal, this paper wants to implement them in Godot, for several reasons:

- It is the engine the author of this paper has the most experience with, having already developed 2 published web games with it.
- It is not got as many resources on procedural generation compared to Unity, Unreal and some other popular game engines, particularly on the side of academic research (that is, there are not as many papers on procedural generation that pertain to Godot as they do to Unity, Unreal and other engines).
  - However, it is still very powerful and feature-rich (it has its own Open Simplex noise class, for example) and this paper is sure procedural generation algorithms can be made to work well on it.
- Compared to Unity and Unreal, Godot is a very light engine with a feature-rich editor, clocking in at under 100MB, with editors for Windows, macOS, Linux and even the web browser.

By the end of the author's allotted time, this paper plans to have implemented several procedurally generated environments in small Godot games, using a myriad of methods (such as Voronoï cells and poisson disk generation) in a myriad of contexts (anything from platformers to first-person games). With these games, this paper plans for the final report to be the centrepiece of this paper, with it containing this paper's research on how each environment was implemented, as well as the findings on the algorithms themselves and how they work.

This is somewhere between a research-oriented project and an implementation-oriented project, as while the produced software artifacts provide valid proof of this paper's understanding of some commonly used procedural generation algorithms and how to implement them in Godot, it is also about how it understands their workings. Nonetheless, the implementations provide the weight behind this paper's motivations and are the main focus of this dissertation. They will prove that Godot is just as adept at procedural content generation as the other major players in the game engine space, and the author of this report will have gained a wealth of knowledge on PCG in the process.

2.2.1 Note on Differing Versions of Godot

Godot currently is at version 4, which finally received a stable release in 4$^{\text{th}}$ March after years of development, but concurrently there is also Godot 3, the previous stable version which is now a **L**ong-**T**erm **S**upport release. The latter version of Godot contains several new features and breaking changes, so any project made in Godot 3 won't readily be compatible with Godot 4 (and vice-versa) without making the necessary changes and conversions. The author of this report has access to both versions of Godot and, for all the Godot projects that were made and used in this project, he have used Godot 4. Any references to other Godot 3 projects will be clearly denoted as such.

2.3 Justifying The Choice of Scenario: A 2D tile-map RPG-style roaming game

The scenario of this paper's choosing involves a monochrome tile-map created by Kenney.nl in a 2D RPG setting, in which the player character is a hollow "Golem" that is trying to search for and obtain a ring among a large 72x40 village, filled with trees, buildings and emptiness. The player can "chow down" trees by simply going to the cells where trees are and making them disappear. However, the player *will* stop at and collide with any buildings in the tile map. When the player collects the ring, they win the game and are able to either close the window or generate a new village to try and collect *another* ring.

The size of the tile map is determined by taking the window size, 1152x640 in **all** implementations, and then dividing it with the cell size, 16x16 in **all** implementations (again), hence returning a 72x40 tile map size. Using a large tile map like this, with 2880 available cells in total, allows for easy stress-testing of the algorithms, making them generate level layouts that are sufficiently large enough to produce a quantifiable performance result and time that can be easily compared across implementations, such that we can easily measure how one performs over the other. The use of a tile map *this* large with PCG algorithms also makes sense from a game developer's perspective as designing level layouts this large by hand, with such a small cell size as well (inherited from the size of the tile map assets), would add additional time and labour costs to them.

The use of a tiled role-playing game scenario, adapted to already-existing procedural generation algorithms, is relatively unusual in the context of procedural generation. However, it *will* allow us to go a degree beyond the scope of what is usually done for procedural content generation in games, which is usually seen in 2D and 3D roguelikes and platformers, as well as some other world-building games such as Minecraft and Terrraria, while also producing code that is relatively easy to process through and understand. The ability for the player character to consume trees and remove them from the level layout by moving into them allows that player to easily move around in what would otherwise be very crowded level layouts that would have been near-impossible to traverse. The addition of said player character, as well as the end goal of obtaining a randomly-placed ring within the given level, adds weight to the algorithms' practical use in games made with Godot, and not just for show or solely as demonstrations.

2.4 Justifying The Choice of Algorithms for the Above Scenario

For this paper, the following procedural content generation algorithms will be implemented within the aforementioned scenario:

1. Lindenmayer Systems (or L-Systems)
2. Perlin and Simplex Noise
3. Poisson Disk Sampling/Distribution
4. Voronoï Cells/Diagrams

Using an L-System for generating a level layout is relatively uncommon, compared to its use in generating structures such as trees and buildings. However, this paper plans to integrate a deterministic context-free L-System (or a "D0L-System") into an implementation of the scenario so that we can compare it performance-wise to the other algorithms, and see how the repeated patterns generated from L-System grammars affect comparisons to the other implementations' level layouts.

Perlin and Simplex Noise are far more commonly used for level layouts, so the author of this report has created an implementation of his scenario with one to see how it compares with the others, speed-wise and layout-wise, and see if it really is the best for the chosen scenario.

Poisson Disk Sampling is usually used for item placement in planes, even with grids, so using a grid-like implementation, we will compare how it works with in a tile map and what differences arise between its use there and in its usual uses.

Though efforts were made to make level layouts as similar as possible across implementations, there are noticeable differences between the level layouts generated by L-Systems, Simplex noise and Poisson disk samples, and this paper touchs on those when discussing those implementations in the relevant sections.

In the paper's research and implementation of Voronoï Cells the author of this report realised that the level layouts it generated for the chosen scenario were wholly unique, when compared with the other algorithm implementations, so much so that he had to re-shape his scenario and game mechanics to make both the scenario and levels generated fit with each other. Nonetheless, he believes this will serve as a unique comparison to the other algorithms and will serve as additional knowledge of procedural generation algorithms as well as more work towards understanding how to make them work in Godot games (as proven by these implementations).

Chapter 3

Report Body

In this chapter, this paper will explain how each of the chosen algorithms work, and how its author went around implementing them as a surface-level explanation. The paper will then briefly compare what challenges were faced for each of our implementations, and how they compare, both performance-wise and with regards to the kinds of layouts they produce, again as surface-level explanations. We go into greater detail on our implementations in the Implementation section (chapter 5), how the level layouts generated in each algorithm compare with each other in the Design & Specification section (chapter 4), and how each implementation compares overall (and also performance wise) in the Evaluation section (chapter 7). For this project, this paper chose to use the following 4 algorithms.

1. Lindenmayer Systems (or L-Systems)
2. Perlin/Simplex Noise
3. Poisson Disk Sampling
4. Voronoï Cells

All of the above algorithms are "ontogenetic." This means that it attempts to recreate the final steps of a real-world process or mathematical calculation without going through much of the intermediary steps.[12] This contrasts with "telelogical" procedural generation algorithms, which **directly** simulate and/or model part of the real world as part of its content generation.[15] This difference between them is described very well in a 2008 article for video games magazine Gamastura by Mick West:

"Two competing methodologies in procedural content generation are teleological and ontogenetic. The teleological approach creates an accurate physical model of the environment and the process that creates the thing generated, and then simply runs the simulation, and the results should emerge as they do in nature.

The ontogenetic approach observes the end results of this process and then attempts to directly reproduce those results by ad hoc algorithms. Ontogenetic approaches are more commonly used in real-time applications such as games. (See ”Shattering Reality,”[sic] Game Developer, August 2006.)”[59][57][58]

3.1 Algorithms

In this section, this paper will explain how each of the implemented algorithms work, then we will go into small detail as to how they were implemented. We go into further detail in the “Implementation” section of this report.

3.1.1 Lindenmayer Systems

Hungarian academic Aristid Lindenmayer devised a mathematical model for the reproduction of fungi in 1967.[35] His model involved a string of symbols, each unique symbol denoting a specific action and/or branch. Essentially, running that initial string, called the *axiom*, through a set of rules (called a *grammar*) gives us an ever-expanding string that is then taken as instructions to draw something from. Lindenmayer Systems, or L-Systems, have since been used in several scenarios beyond its initial purpose of modelling fungi, from trees to fractals. In video games, they are frequently used to aid in the creation of foliage in several environments, as well as buildings and, here, level layouts. We go over how the author of this report got his implementation to work with complex multi-rule grammars in Chapters 3.2.2 and 5.1.

A Basic 0L-System

The most basic form of L-System is a *0L*-System, 0 in this case referring to the fact that the grammar is *context-free.*

For this example[2], consider an alphabet V, which consists of the following symbols:

$$\mathrm{F}, +, -$$

where F means “to go forward”, and + and − denote turning right or left (respectively) a set number of degrees ø.

Take an axiom ω, for example:

$$\mathrm{F+F+F+F}$$

And a set of rules P which, in this case, is of size 1:

$$\mathrm{F} \rightarrow \mathrm{F+F-F-FF+F+F-F}$$

We can represent this *parametric* L-system in the following form:[60]

$$G = (V, \omega, P)$$

The first 3 iterations of string replacement with this one-rule grammar G are shown here:

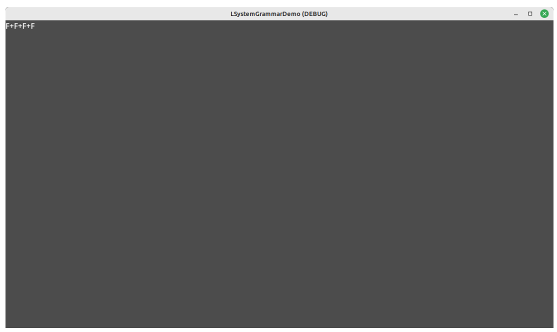

Figure 3.1: The axiom of the aforementioned simple L-System with just one rule. String size: 8. Source: Own work.

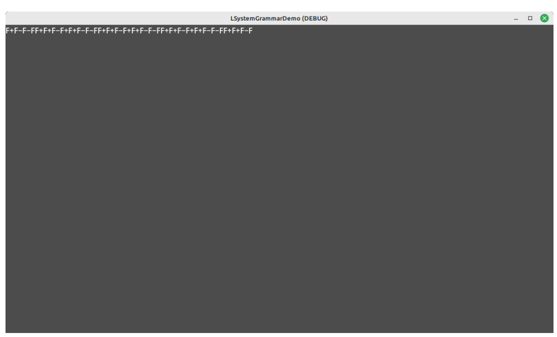

Figure 3.2: The first iteration of the aforementioned simple L-System with just one rule. String size: 59. Source: Own work.

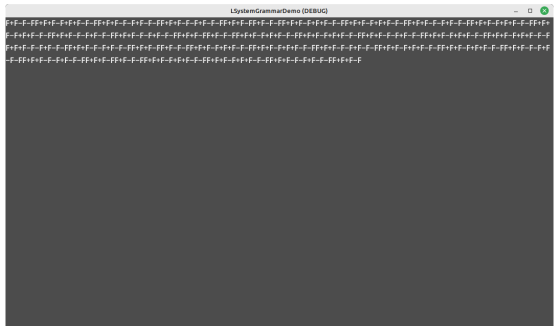

Figure 3.3: The second iteration of the aforementioned simple L-System with just one rule. String size: 475. Source: Own work.

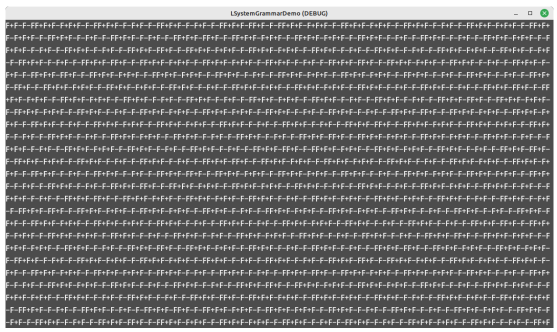

Figure 3.4: The third iteration of the aforementioned simple L-System with just one rule. String size: 3803. The string is too large to show in the window, as you can see here. Source: Own work.

The resulting string can be used to draw a lattice.[2] Examples of the above grammar in action are below.

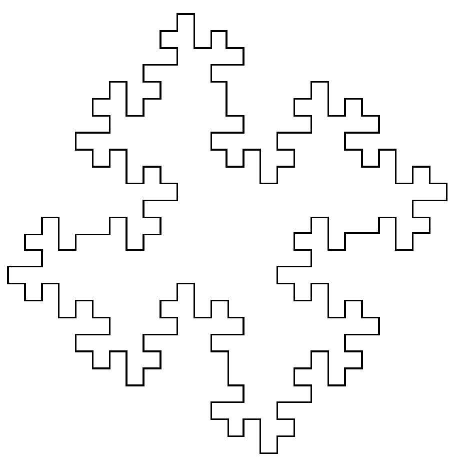

Figure 3.5: A lattice generated with the example grammar on a custom-written Classic Mac OS application specifically written for working with L-Systems.[2]

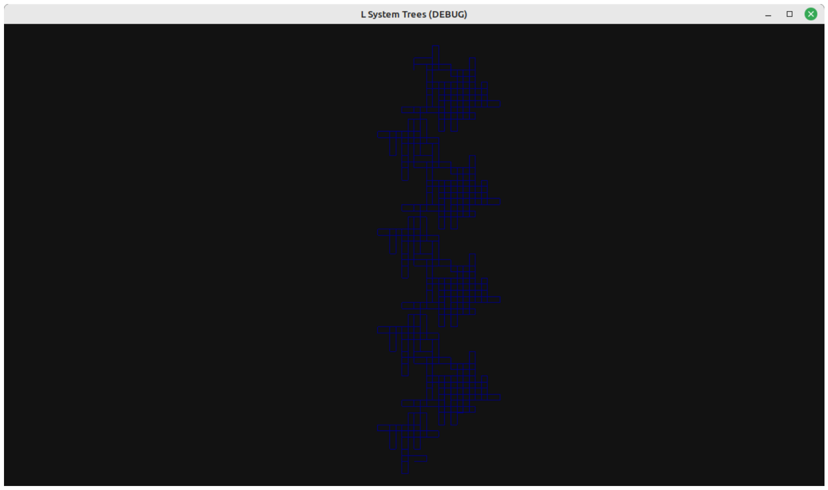

Figure 3.6: A lattice generated with the example grammar on a Godot project for drawing from L-Systems. Source: Initial project written by Alexander Gillberg for his YouTube channel Codat[23][24], and converted to Godot 4 (with the addition of the lattice grammar) by the author of this report.[25]

A More Complex D0L-System With More Than One Rule

The grammar in the following example represents a D0L-System[42], a **deterministic** L-System using a context-free grammar; the grammar in the first example was *also* deterministic.

For this example, consider a new grammar G with the alphabet V, where a and b are the only symbols. We start with the following axiom ω, which is just a. We now have a set of rules P which is, this time, of size *2*:

$$\mathrm{a} \rightarrow \mathrm{ab}$$

$$\mathrm{b} \rightarrow \mathrm{a}$$

The first few steps of the resulting derivation can be modelled like so:

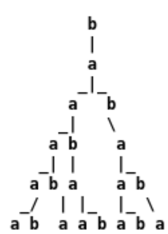

Figure 3.7: The first few steps of a derivation of our example grammar.[42]

3.1.2 Perlin/Simplex Noise

Traditionally, white noise images, and most other noise types, place noise pixels completely randomly, without each pixel considering the values of its neighbours[44], as you can see in Figure 3.8.

However, there exists several types of **value** and **gradient** noise that *do* take surrounding pixel values into consideration, and will therefore serve more use in building levels in our games.

Value noise simply takes a lattice of points with random values and then interpolates those points based on their surrounding values. This *can* be used as a procedural texture. However, due to the simple nature of the algorithm, it's possible that the difference between several values in a region is minimal, while in other regions the values may differ immensely, resulting in a noise image that is not very smooth.

Gradient noise, on the other hand, takes point lattices and instead calculates the interpolation between tangents.[10] Since both tangents between a curve must be collinear[10], the flat and bumpy curves produced by value noise's interpolation calculations are now much less likely to be returned, as seen in Figure 3.9.[10] This results in noise images of higher and more appealing visual quality as, to quote a response from Stack Exchange by Hernan J. González[26], "it cuts low frequencies and emphasizes frequencies around and above the grid spacing."

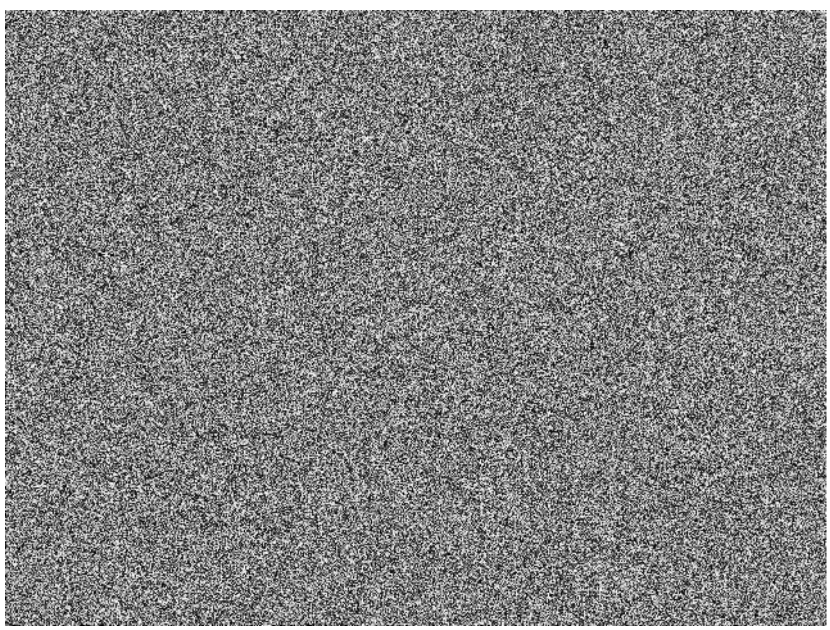

Figure 3.8: A white noise picture generated with Robson's white noise image generator.[50] Settings: 640 squares horizontally, 480 squares vertically, size of squares 1, colours greyscale, bias none.

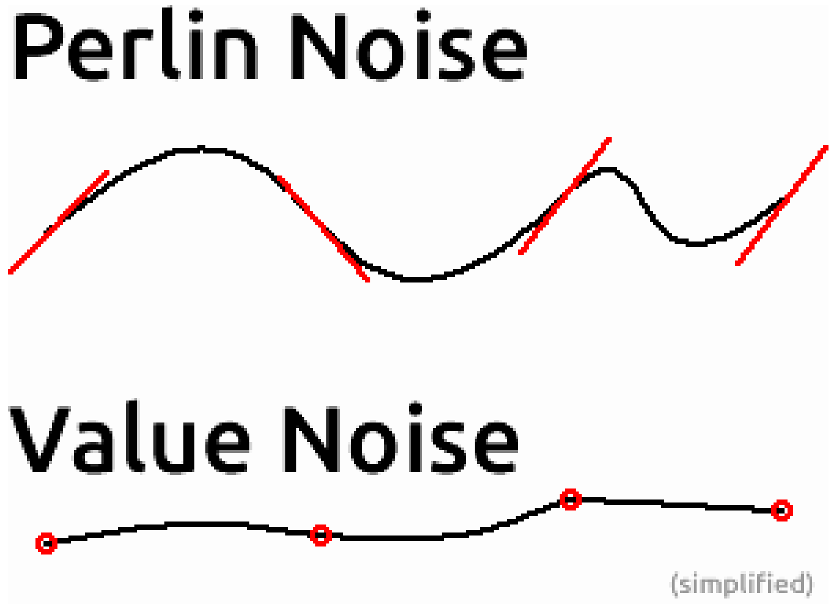

Figure 3.9: A comparison between the kinds of curves produced by Value noise interpolation and Perlin (and other Gradient) noise interpolation.[10]

Two particularly well-known Gradient noise algorithms that are commonly used for procedurally generating levels are the already mentioned Perlin Noise and Simplex Noise, both designed by American Computer Science professor Kenneth H. Perlin, with the former being an improvement on the former. Perlin Noise also takes a lattice of randomly assigned gradients, but the algorithm interpolates the dot products of those points instead of just their neighouring values.[36] Simplex noise, meanwhile, tries to reduce the grid artifacts caused by the original algorithm, and has the added benefit of scaling better to larger dimensions.[14] Perlin filed a patent on his work in 2002 that was granted in 2005[45], which prompted the creation of the OpenSimplex noise algorithm[30][46][29] for free use; the patent has since expired in 2022, allowing free use to both Perlin and the original Simplex noise.[45]

Godot 3 previously featured an OpenSimplexNoise class[28][18] for generating noise textures, which used the OpenSimplex algorithm. In addition to using a "simplectic honeycomb" for its lattices[29], this algorithm also (to quote Michael Powell) "expands the range of the gradients a bit, so they can extend a little bit into neighboring cells. This theoretically makes the noise a little bit smoother, but it also means that extra cells need to be checked."[46] Godot 4, on the other hand, allows us to use the *original* Simplex noise algorithm, as well as Perlin noise, 2 types of Value noise and a variation of Simplex noise that produces smoother, high quality noise images with an additional performance cost, and it allows us to control which algorithm we use for noise generation using the "noise_type" property and "NoiseType" enumeration in the “FastNoiseLite” class that is now used for noise.[36]

3.1.3 Poisson Disk Sampling

Poisson disk distributions are an easy way to randomly scatter objects across a field. It’s commonly used for tree placement and placement of other random objects. Points are placed over a plane, with a single point placed randomly and subsequent points calculated such that a single point has no other point lying within a given radius of said point. Different implementations of Poisson disk distributions or samples can accommodate multiple radii for points in a plane, and some implementations produce *maximal* samples- that is, a set of samples that fully cover the given plane, while still adhering to the principle that no single point has other points lying within its radius[17] (the implementation that was made for this project does **not** guarantee maximality, however).

An implementation of Poisson disk sampling was originally developed in 1991 by Don P. Mitchell[39] as a replacement for inefficient Monte Carlo “dart-throwing” algorithms.[48] Mitchell’s algorithm ran in $\mathcal{O}(n^2)$ time, whereas Robert Brinson’s 2007 improved algorithm for Poisson disk sampling[3] ran in $\mathcal{O}(n)$. Subsequent quality and speed improvements to Brinson’s algorithm were published in 2019[49], 2021[48] and 2022[51]. The implementation made for this project, as well as the Unity project this was based on, were both based on Brinson’s 2007 $\mathcal{O}(n)$ algorithm.[34][33]

The following are some examples of Poisson disk distribution in action:

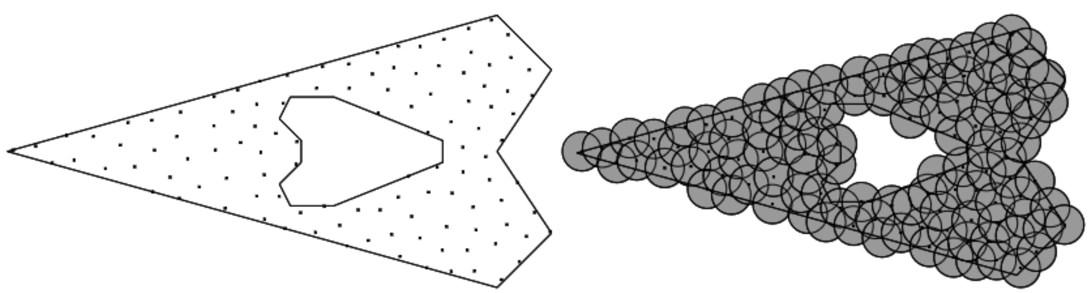

Figure 3.10: A diagram of a maximal Poisson disk distribution done on a concave plane, with the right side denoting maximality through the grey disks overlapping but not any points overlapping.[17]

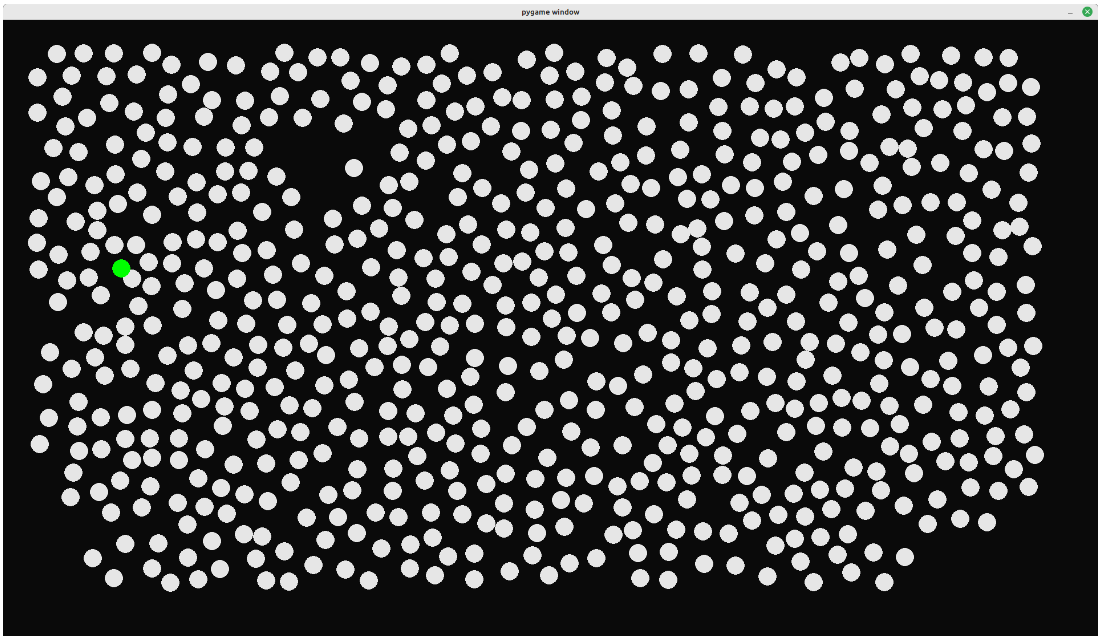

Figure 3.11: An implementation of Poisson disk sampling made in Pygame.[1] The screenshot was taken *after* all of the samples were taken.

3.1.4 Voronoï Cells

Named after the Ukranian mathematician Georgy Voronoy, Voronoï cells work by taking a map of points, and randomly selecting a group of points. Within that selected group, cells are formed by calculating, in each point of the grid, the closest of the selected points to it. That is, each cell represents the group of points that are the closest to that random point (including that point in the group as well).[16] The final arrangement of cells represents a Voronoï Diagram or Voronoï Tesselation.

Distances between points can be calculated with either the Euclidean distance:

$$d_E(p,q) = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2}$$

or the Manhattan distance:

$$d_M(p,q) = |q_x - p_x| + |q_y - p_y|$$

With the Euclidean distance producing a more "triangulated" tesselation than the Manhattan distance, with straighter diagonals and cells shaped like irregular polygons, the geometry of which is more "blocky" and resembles taxicabs (hence its alternate name "Taxicab Geometry"). Two visual comparisons of the kinds of Voronoï cells generated with either distance calculation are shown in Figures 3.12 and 3.13.

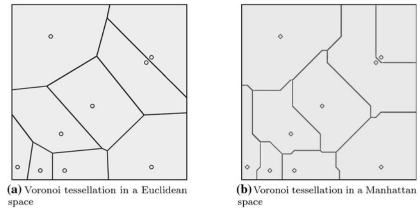

Figure 3.12: A visual comparison of the kinds of Voronoï cells generated with the Euclidean and Manhattan distance.[54]

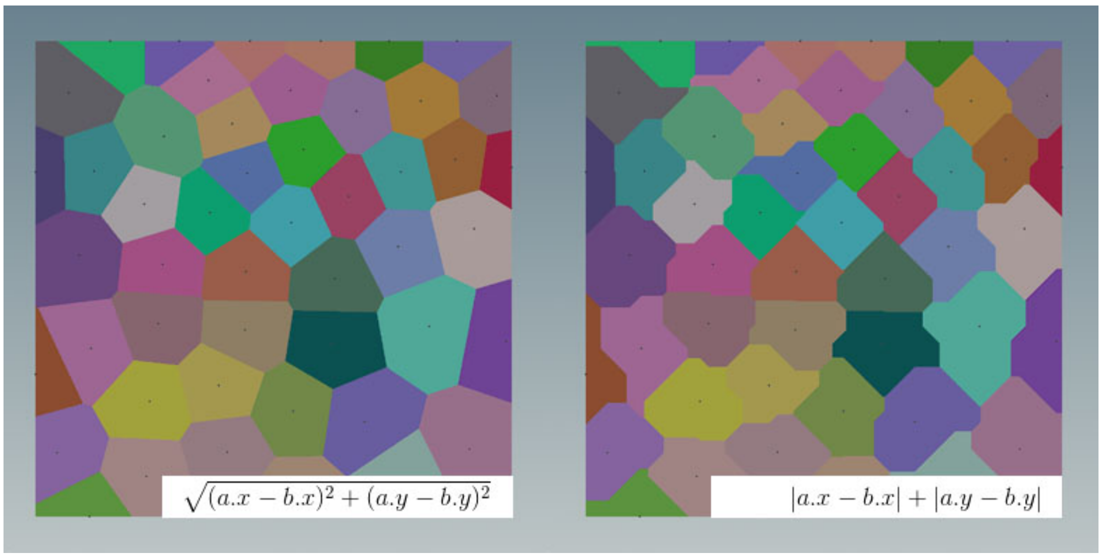

Figure 3.13: Another example of the differences between a Voronoï tesselation with distance between points calculated with either the Euclidean distance or the Manhattan distance.[55]

3.2 Implementations

Here we will describe, at surface level, the methods this paper's author went about implementing the above algorithms and what references were used for these implementations.

3.2.1 Commonalities Between Implementations

To implement the same scenario, aforementioned in the background of this report, across all 4 algorithm implementations, this paper's author had to include some of the same code and functions, as well as the same tile set shown in Figure 3.14.

From this tileset, which contains 1078 tiles, the written code uses 27 building tiles, 13 tiles for trees and other fauna, 1 tile for the player character and one of 4 tiles for the ring. The relevant coordinates of the tiles for buildings, trees and the ring are each stored in constant arrays in the script, while the player tile's coordinates are just stored in a local constant (not an array, since there is no need for one).

To handle player placement and subsequent movement, the code has several functions. Godot's built in "physics_process" function handles events that happen in real-time, and is commonly used, like in this context, for player movement. In it, we first store the current player's cell, "player_movement_cell", in "previous_cell", then we initialise a "direction" based on which input movement was pressed ("Vector2i.LEFT" when ""ui_left"" was pressed, and so on). Then we add the player's current cell with the direction to calculate the potential "new_movement_cell". If this cell is within the bounds of the enviroment, as well as either a tree or empty space (or the ring), it moves there, and the previous cell gets erased. If the player ends up moving into the cell where the ring is, the player wins the game, and all movement is paused while a winner's dialog popup shows up. The player moves **very** quickly in our games, and we have yet to figure out how to slow down this movement while also not making movement so slow that the games drags; the player will not want to have to continually press down an arrow key to move to 1 cell in a map of 2880 cells. Since the performance of the algorithms are more important in this project, however, we decided to leave the very fast player movement as is.

There are written "place_player" and "place_ring" functions that handle the random generation of the player's and ring's initial starting positions. Both use the "_get_random_placement_cell" helper function to retrieve a new cell, and both use a while loop to make sure the randomly generated cell isn't already occupied. In both functions the placement cells are assign and calculated **before** the while loop, so that their placements do not default to just (0, 0) in the beginning.

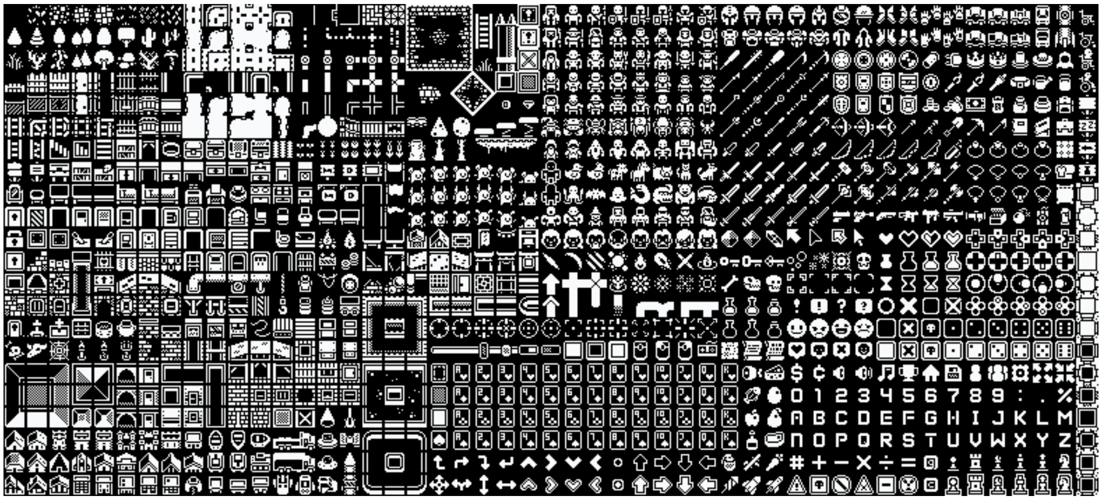

Figure 3.14: The tileset used for all 4 implementations of the chosen scenario with PCG algorithms.[31] Of all the 1078 tiles, of size 16x16, in this tileset, only 45 of them get referenced in the written code.

Across all implementations, there are two local variables, “x_tile_range” and “y_tile_range”. Both of these calculate the dimensions of our tile map by taking the display window’s respective x and y dimensions from the project’s settings (1152x640) and divides them by the respective x and y dimensions of the cell size (16x16). “x_tile_range” should resolve to 72 upon runtime, and “y_tile_range” should equal 40, giving us our 72x40 tile map that gives us a total of 2880 cells to work with in our games.

Finally, there are two dialog popups added to each scene tree, one for describing the game’s story (“AcceptDialog”, of type “AcceptDialog”) and another for when the game ends after the player has collected the ring (“WinDialog”, of type “ConfirmationDialog”). For “AcceptDialog” the “confirmed” and “canceled” signals are both connected to the function “_on_AcceptDialog_closed”, which hides the popup and unpauses the game. For “WinDialog”, on the other hand, “confirmed” is connected to “_on_WinDialog_confirmed” and “canceled” is connected to “_on_WinDialog_canceled”. “_on_WinDialog_confirmed” is meant to generate a new level layout, while “_on_WinDialog_canceled” is meant to close the game, both when the cancel button (labelled ”Get Me Out of Here”) is clicked and when the cross on the top-right corner of the popup is clicked. However, as of now, only the top-right corner of both popups does what it is supposed to; clicking any of the other buttons from both popups, for some reason, does nothing at the moment, and the author of this paper *and* the code *did* make sure, in his code, that the signals were properly connected. However, the games themselves still run as they are supposed to, and the integration of the algorithms into the levels in the games are more important here, so since *they* still work, he decided to leave the popups, their behaviour and their code as they were. *If* they are engine issues, regarding the buttons, they may hopefully get fixed in future versions of Godot.

3.2.2 Lindenmayer System

The implementation of an L-System was very simple. Inspiration was taken from a YouTube video on implementing an L-System for drawing line graphics in Godot by Alexander Gillberg.[23] In the code from the Godot 3 project Gillberg made in that video[24][23], he created a custom "Rule" class in GDScript, with which he defined new rules. This paper's author forked his project, converted it to Godot 4 and used it to create the lattice graphics in Figure 3.6.[25] He did this mainly as a reference for the implementation of L-Systems in the game itself.

With the implementation in our *game*, the "get_new_character" method in that L-System was adapted to work with the dictionary this paper originally implemented the L-System in. The new "get_new_replacement" method in our implementation allows for there to be more than one grammar rule while the L-System still performs as it should. The original L-System iterated through the original string *directly*, which produced unintended consequences in grammars with multiple rules, as seen here when trying to implement the D0L-System that was mentioned earlier[42]:

$$b \rightarrow a \rightarrow aa \rightarrow aaa \rightarrow aaaa \rightarrow aaaaa \ldots$$

By using an empty string buffer and inserting rule replacements there instead, this new implementation is now able to perform substitutions accordingly; the correct computation of the D0L-System is denoted in Figure 3.7 and repeated below:

$$b \rightarrow a \rightarrow ab \rightarrow aba \rightarrow abaab \rightarrow abaababa \ldots$$

With the L-System string parsing algorithm in place, the next step was to paint the cells of each tile. With this, we iterated through every cell of the tilemap using a nested for-loop. With the parsed string, we then accessed the character of the string at an incremented index using an iterator variable we defined before the for-loops. The string consists of three different characters repeated multiple times, "O", "W" and "B". For each string index, if the character is "W", paint a tree, if it is "B", paint a building, and if it is an "O", leave the cell blank and paint nothing. The player and ring then get placed afterwards.

Even for a large-sized tile map with 2880 cells, a constant L-System G, with the symbols

O, W and B and the following grammar

$$O \rightarrow OWO$$

$$W \rightarrow WB$$

$$B \rightarrow BWO$$

can parse the axiom OWB, paint tile map tiles with the resulting string **and** place the player and ring in just 19 milliseconds on average. This was the default grammar used by the L-System in the game. The Godot project also includes 3 more grammars, one that generated more buildings (and impossible level layouts), another that generated more trees and another that generated more empty space. These can be easily selected with the “ruleset” export variable in the Godot editor. Further variance can be added with the addition of a randomly generated axiom, capped at a maximum height or smaller (minimum 1). If said option is enabled in the Godot editor, the default value in the export variable for setting this cap is 10, and since it is an export variable, it too can be adjusted in the editor as the developer sees fit.

3.2.3 Perlin/Simplex Noise

The Simplex Noise implementation works with Godot’s built-in Noise library. Within a Sprite2D node’s Texture attribute, a new “NoiseTexture2D” field was set inside of it. In its “Noise” attribute a new “FastNoiseLite” scene was created, which generates a noise texture for us to use. The seed can be set in the sprite’s script file.

As with the other implementations, there are two separate arrays, one for trees and another for buildings. For each cell in the TileMap, the code then took the noise pixel from the generated texture at that exact point (scaling with the cell size accordingly), using the “get_noise_2d” method built-in with Godot, and then, depending on the value retrieved, decided, firstly, whether or not to place a plant/tree tile there and, secondly, whether or not to place a building tile there. As a result, not every cell in the TileMap has tiles on it. On any one of those empty cells, the Player tile will then get placed.

For the generation of the noise itself, we *could’ve* added a “Sprite2D” node to the scene tree, the root of which was the “TileMap”, and gave it a “NoiseTexture2D” texture and set its “noise” property to a newly-created “FastNoiseLite” instance, the latter of which contains the actual noise data. In the early stages of this implementation’s development, that’s what was done, and this paper’s author created a script that solely set the seed of the “FastNoiseLite” resource to a random integer (using the “randi” method). However, for a more authentic result, and to forgo the need of an additional node and noise texture that will not even be visible in the final product, it was eventually decided to create the noise for this algorithm implementation entirely programmatically. The code now stored the “FastNoiseLite” instance in its own class variable “noise”, and instantiated it with the “set_noise” method when starting the game (the “_ready” function automatically runs when the game starts).

Initially having done the noise integration with a sprite node and noise texture allowed the author of this report to experiment with some of the “FastNoiseLite” class’s properties before finally resorting to programmatic noise creation. An instance of this class, by default, uses the “Simplex Smooth” noise algorithm, a version of the Simplex algorithm that produces higher quality noise images at the expense of slower speed.[36] We can also use just “Simplex” noise for higher speed, as well as the original “Perlin” noise algorithm.[36] Godot also allows us to use two kinds of Value noise, as well as a “Cellular” type that combines algorithms like Worley Noise and Voronoï diagrams to create ”regions of the same value.”[36] There were problems with the “Cellular” noise type when experimenting with it, for reasons we will get into later, but the other noise types were made readily accessible in an “export” variable in the scene script (that is, a variable that can be easily accessed in the Godot editor when the TileMap node is clicked on) when this paper’s and the code’s author removed the sprite node and decided to programmatically make the noise. When the “set_noise” function is called, the noise type is assigned through the “_get_noise_type” function, which returns an integer value depending on the type of noise selected, and the returned result is cast to “FastNoiseLite”’s “NoiseType” enumeration[36] before it gets assigned (this prevents an “INT_AS_ENUM_WITHOUT_CAST” warning from the Godot editor’s linter for GDScript[38]).

Furthermore, there are 3 other export variables in the TileMap script for this implementation that directly correlate to some of “FastNoiseLite”’s properties. The “noise_frequency” variable in the script correlates to the “frequency” property in “FastNoiseLite”, which, as both names suggest, sets the noise frequency; the higher the frequency, the rougher and more granular the noise[36], which is probably why it is set to 0.01 by default.[36] The “fractal_type” and “cellular_distance_type” in the script **directly** correspond to the “fractal_type” and “cellular_distance_function” properties respectively, to the point where both even use the relevant enumerations from “FastNoiseLite” directly (“FractalType” and “CellularDistanceFunction” respectively).[36] The relevant values are all assigned accordingly in “set_noise”.

In terms of determining whether or not to place buildings or trees (or nothing), inspiration was taken from a YouTube tutorial by Gingerageous Games utilising Godot 3[21][22] (which breaks in Godot 4). His tutorial used multiple "TileMap" nodes in a single scene tree with a "Node2D" root, and controlled each individual tile map, representing a specific part of the environment (such as grass and roads), and used a floating point "cap" to determine whether or not to place a tile in a cell based on the noise pixel retrieved at that cell's coordinate.[21][22] Since we are using just one tile map for everything (trees and buildings), a conflict had to be mitigated where the building cap was smaller than the tree cap. If that were the case then, since the tree cells get painted first in the Godot 4 implementation, no buildings would ever get painted. To mitigate this, an additional condition was added to the if-statement for painting building cells (in the same line, to prevent creating a nested if-statement), which would allow the algorithm to overwrite an already painted tree cell with a building cell subject to a randomly generated floating point number (between 0 and 1 inclusive) being below a pre-defined floating point number in the exported variable "building_overtakes_tree". This would then allow there to be a controlled proportion of buildings compared to trees (the higher the proportion, the more buildings compared to trees), regardless of whether the building cap was lower than the tree cap or not, and the algorithm would still perform as normal should the reverse be the case.

3.2.4 Poisson Disk Sampling

The Poisson Disk Sampling implementation was based on a Unity tutorial by Sebastian Lague[33][34], in which he used his algorithm to draw points onto a grid. He based his algorithm on Bridson's $\mathcal{O}(n)$ algorithm.[3] The way he wrote *his* implementation was such that the radius of the circle would be equal to the diagonal of each square in the grid by default (when the radius was 1.0), ensuring that no point ever lies within the radius of another.

This paper's implementation of the Poisson Disk Sampling algorithm mostly took from him, with some changes. Lague did his implementation in the C# language and, while Godot 4 *does* have a separate version with C# and .NET support, this paper opted to use the standard GDScript distribution of Godot 4 with all of our implementations. This meant that the code had to be adapted to work with not just the tile map but also the way GDScript worked. For one thing, the "grid" array in the "generate_points" had to be manually initialised by inserting arrays into an empty array, the quantity determined by what would have been the outer length of the 2D array (and what basically *was* this in Lague's implementation), that being the ceiling function of the x-dimension of the sample region size divided by the cell size. From there, in each of the nested arrays, the value 0 had to be programatically inserted to all of them, the quantity of the *zeroes* also being determined by what would have been the *inner* length of the 2D array (and what basically *was* this in Lague's implementation), that being the ceiling function of the y-dimension of the sample region size divided by the cell size.

Adapting Lague's implementation from C# and Unity to GDScript and Godot involved some extensive research into Unity's API. When calculating the angle in "GeneratePoints"/"generate_points", for example, the equivalent of Unity's "Random.value" in Godot is "randf" (which *has* no static class to be called from). Furthermore, GDScript has a "TAU" constant that does the "Mathf.PI * 2" calculation done in Lague's Unity implementation. The "sqrMagnitude" method used in Lague's "isValid" function becomes "length_squared" in the "is_valid" method. When implementing "isValid" in GDScript this paper's author also had to make sure the inner and outer dimensions of the grid could be adequately accessed. We go over how that was achieved in the "Implementation" section of this report (see chapter 7).

3.2.5 Voronoï Cells

This paper's author based his implementation of this algorithm on some JavaScript code posted by an anonymous contributor to the Procedural Content Generation Wiki on the Wikidot platform in 2017, in which a brute-force implementation of the algorithm was implemented.[16] An auxiliary function in the JavaScript code, "randRange", was taken out of *this* implementation, since Godot has a built-in "randi_range" function that serves the exact same purpose.[37] As he got further and further with his implementation of Voronoï diagrams in Godot, he realised the way the algorithm inherently worked meant that the level layouts it designed would be wholly unique, espcially compared to the other three algorithms for which he made implementations of the chosen scenario.

For example, unlike the other implementations, the algorithm ensure that all cells of the tilemap were **always** covered (to start with, in our game's context), whereas the other implementations always left some cells unpainted. The nature of Voronoï tessellations also meant that groups of trees and buildings were bunched together, with no guarantee that they would ever form coherent connections that would make sense in a level of our scenario. This meant that the ring and player placements had to be altered so that, instead of being placed in non-existent empty cells, they would replace the cell of a tree.

Even *with* that, there would be no guarantee that a player would be able to complete a level successfully. For example, if a player and ring were spawned in different Voronoï cells of trees, and both of those cells were separated by cells of buildings such that they could not ever be feasibly reached, the game would be impossible to finish. Therefore, a new input event was created, “reset_position”, which can be triggered by pressing *either* the G key on a standard computer keyboard *or* the right-click mouse button. Triggering the event respawns the player character in a different position, which could be occupied by either a tree or the ring, ensuring that the ring can still be collected and, therefore, game can still be won. The code for when this event is triggered is essentially a rehash of the code for “place_player”, except that the new cell **can** be the cell occupying the ring, and also the previous cell’s contents will be deleted (as the player is no longer at that position).

While the differences are drastic and very noticeable, this paper’s nonetheless kept working on this implementation and included it in the project artefacts. He believe that the fact that he was able to work through it and implement a working version of the chosen scenario with it (albeit with some changes) adds further strength to this paper’s claims that Godot can work well with procedural generation algorithms, even ones where use in the context of a tile map RPG would be rarer, as well as proving the author’s strengths as a games programmer in making tile maps work with PCG algorithms.

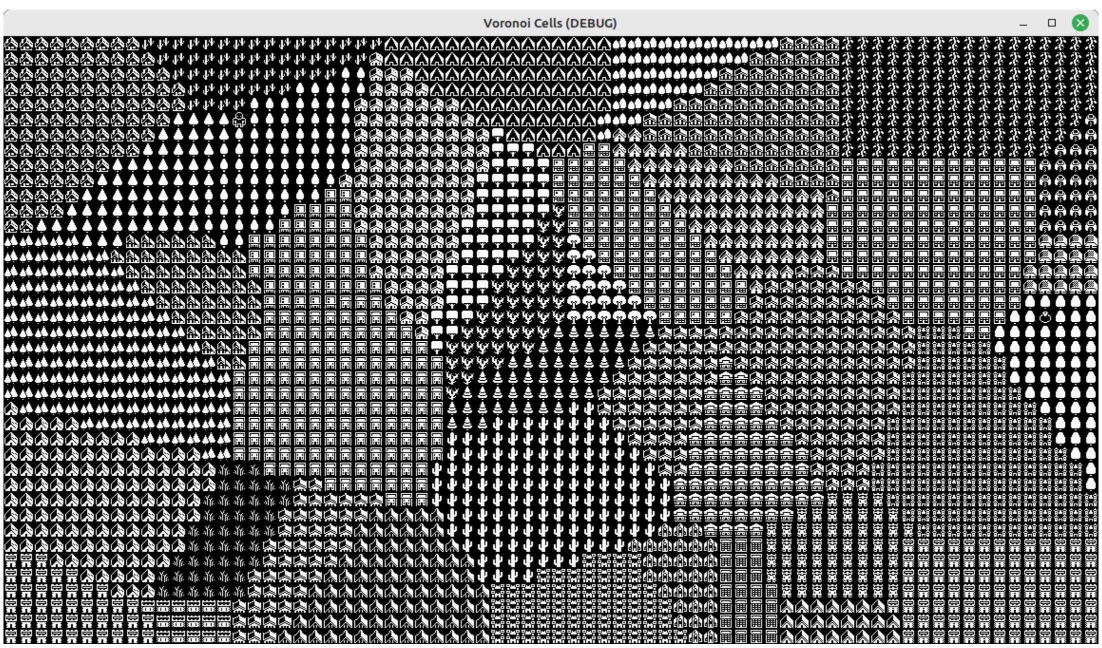

Figure 3.15: An example of the Voronoï cells implementation of the scenario in action.

Chapter 4

Design & Specification

Here, this paper will provide an abstract level of how the performance of each content generation algorithm was compared and how it was ensured that each implementation could produce as similar/like-for-like results as possible (and where they *couldn't* do so). More specifics and quantified data are delved into in Chapter 7.

4.1 What Was Sought After For In The "Best" Implementation Of The Chosen Scenario

To help determine what was wanted from these implementations and how similar they should be, it was decided, after repeatedly trying out the implementations as they were continually worked on and finessed, that they should adhere to the following *specification*, which best fit the chosen scenario:

- They should produce levels that are able to be completed every time; that is, the player character should be able to collect the ring and win the game every single time.
  - This was part of the motivation between the idea that there should be free space to move around from the outset.
- There should ideally be an ample amount of free space to move around.
  - While the golem can "eat" trees to gradually create new free space over time, it was highly desired for the levels to *start* off relatively spatial in this regard, while also still containing a very highly ample amount of trees and buildings within the boundaries of the levels.

- The golem and ring should be spawned separately from the generated environment.
  - By *always* placing the tree and golem in an empty space that *was not* always the top-right corner (or spawning at a tree for the Voronoï cells implementation where all cells are occupied), it would ensure that the player can start and win the game by roaming on empty space and consuming trees on the way to collect the ring.
  - This was *also* part of the motivation behind having free space from the outset.
- The trees and buildings should be scattered around *with purpose* (that is, they should *look* very random while also being highly calculated behind the scenes), such that it should be quite hard, but not necessarily impossible, to discern which algorithm was behind which level.
  - In the final implementations, like in Figure 7.1, the levels they generate should *seem* relatively similar to one another, albeit not necessarily identical.
- Levels should be generated fairly quickly while still adhering to the other principles in the given specification.
  - While each algorithm did not need to be the *fastest*, necessarily, any processing times above 500ms would not be advisable.

4.2 Comparing Layouts-Wise

Comparing the layouts involved trying out the implementations throughout their continuous development for this project. This involved tweaking values and calculations around. For instance, the Voronoï cells implementation involved the use of two different distance calculations that produced wildly differing cells and cell shapes. The "FastNoiseLite" class in Godot includes various configurable properties for noise images, such as the frequency of the noise, the fractal type used to alter the generated noise and even the noise

4.3 Comparing Performance-Wise

Comparing each implementation performance-wise was far easier than comparing them layout-wise, since this did not rely on assessing layouts by eye. Checking the quantified performance time for all algorithms involved both a print statement in the "_ready" function **and** the processing time showing in the opening story dialog, the text of which was *also* set in the "_ready" function.

Calculating the processing time in the "_ready" function for all implementations involves:

1. The calculations and processing of the algorithm itself.
2. The painting of cells using results returned by the algorithm.
3. The random placement of the golem.
4. The random placement of the ring.

Chapter 5

Implementation

Here, this paper will go a degree deeper as to how each algorithm was made to work with the chosen scenario. Where possible, it plans to use code snippets from the work its author have done to justify how and why things were implemented the way they were.

5.1 Lindenmayer System

To implement our basic grammar in Godot (see chapter 3.1.1), we can take each rule and replace each string in accordance to our one rule, using the replace method, as demonstrated in Figure 5.1.

For handling more than one rule, we can instead use a new string buffer variable where, for each character in our string, we can attain a new string and append it to our string buffer. The resulting string is then returned and interpreted. This can be represented in Godot as demonstrated in Figure 5.2, which uses two functions to perform string replacement. The first function "get_new_replacement" performs the character replacement according to the L-System's grammar rules, while the second function "replace_string" uses a string builder variable to allow for replacement of characters without directly affecting the original string and causing unwanted side effects (see chapter 3.2.2 and also the tutorial in the following citations[23][24][25], from which major inspiration was taken for the L-System implementation). The "get_new_replacement" function was eventually used in the final implementation, whereas the code in the "replace_string" function was adapated into this implementation's "parse" function, shown in Figure 5.5, in which some export variables were accounted for and the number of iterations was controlled through a while loop rather than a for loop, the while condition being that the string size was smaller than the total number of cells. The string that “parse” returns chops off any excess characters.

This can *then* be used to handle more complex grammars that can handle more than one rule in which characters in strings are replaced by other strings of variable length, as seen in the example in Chapter 3.1.1.

With a constant use of the same grammar rules and axiom, one issue that arose in the development of levels in our scenario, with L-Systems, is the lack of variance in the kinds of tiles placed, and it was hard to figure out how to deviate even slightly from the eventually recognisable patterns the algorithm and default grammar (see chapter 3.2.2) would create. Thus a mitigation was developed for this by allowing a choice between the provided axiom or a random one (the user/developer can also change the axiom in Godot’s editor). This is down to the exported variables in the L-System node’s script that was implemented (the L-System node’s parent is the “TileMap” root node). The relevant ones from the “l_system.gd” script file are shown in Figure 5.3. If “use_random_axiom” is set to true (and it is by default), then it takes a maximum character limit “upper_limit” (again an export variable configurable in the Godot editor itself) and then it generates a string of a length **up to** that limit- that is, the length of the returned string between 1 and the limit, both inclusive- using the alphabet that all of the provided grammars adhere to (“O” for blank space, “W” for trees and “B” for buildings). The axiom is then assigned to the “string” script variable in the “paint” method by calling the “paint” method and assigning the return type to it.

For the sake of creativity and additional experimentation, additional grammars were devised that can be configured with the “ruleset” variable. The “Default” grammar is as described in 3.2.2, and the additional grammars generate higher proportions of either buildings, trees or empty space depending on which one is chosen (with the “more buildings” grammar producing levels that are impossible to finish in our scenario). The grammars themselves are detailed in Figure 5.4, and the method used to return the specifically set grammar in “parse” is shown in Figure 5.6.

```
string = string.replace(rule["from"], rule["to"]) #Here the rules were stored in dictionaries.
```

Figure 5.1: A line of code that demonstrates directly replacing characters in a string according to our L-System grammar’s rules.

```
func get_new_replacement(character: String) -> String:
    for rule in rules:
        if rule["from"] == character:
            return rule["to"]
    return character

func replace_string(string: String) -> String:
    var new_string = ""
    for character in string:
        new_string += get_new_replacement(character)
    return new_string
```

Figure 5.2: Two GDScript functions for replacing characters in an L-System grammar with more than one rule. The first function was used in the final L-System implementation of the scenario. The second function was adapted into the "parse" function of *this* implementation. Both functions are in the l_system.gd script file.

```
@export var use_random_axiom: bool = true
## Defines how many characters a random axiom can have MAXIMUM. Only used when use_random_axiom is true.
@export var upper_limit: int = 10
```

Figure 5.3: The use_random_axiom and upper_limit variables used when allowing a random axiom for an L-System grammar and then setting a maximum character limit for it.

$$O \rightarrow BWOB$$

$$W \rightarrow WBOBO$$

$$B \rightarrow BB$$

$$O \rightarrow WWO$$

$$W \rightarrow WBWO$$

$$B \rightarrow BWWO$$

$$O \rightarrow OOBWO$$

$$W \rightarrow OB$$

$$B \rightarrow OW$$

Figure 5.4: The other 3 grammars created for the L-System implementation, aside from the default one covered in chapter 3.2.2. More buildings, more trees and more space respectively.

```
func _size() -> int:
    return tile_map.x_tile_range * tile_map.y_tile_range

func rand_axiom() -> String:
    var string_buffer: String = ""
    var limit: int = randi_range(1, upper_limit)
    for i in range(limit):
        string_buffer += ["O", "W", "B"].pick_random()
    return string_buffer

func parse() -> String:
    if use_random_axiom:
        axiom = rand_axiom()
        string = axiom
    if not use_custom_ruleset or ruleset != "Default":
        rules = _get_ruleset()
    var size: int = _size()
    while len(string) <= size:
        var new_string = ""
        for character in string:
            new_string += get_new_replacement(character)
        string = new_string
    string = string.substr(0, size)
    return string
```

Figure 5.5: The parse function in l_system.gd, which takes the rand_axiom and _get_ruleset functions (if needed; the latter is shown in Figure 5.6), then gets the string size through _size. Then, the string replacements described in 5.2 are carried out in the while loop, before the string is then returned (albeit without any excess characters that will not be used in the paint method to paint the cells in the tile map).

```
func _get_ruleset() -> Array[Dictionary]:
    match ruleset:
        "More Buildings (IMPOSSIBLE)": return MORE_BUILDINGS
        "More Trees": return MORE_TREES
        "More Space": return MORE_SPACE
        _: return DEFAULT
```

Figure 5.6: The _get_ruleset function used in the parse method in figure 5.5.

5.2 Perlin/Simplex Noise

While the “noise_type” export variable is a selection of strings that are eventually taken from to return an enumeration for “NoiseType”, a member of the “FastNoiseLite” class[36], the “fractal_type” and “cellular_distance_type” export variables more directly correspond to the relevant enumerations from “FastNoiseLite”, as described earlier in chapter 3.2.3 and shown in Figure 5.7. The *other* export variable is “noise_frequency”, which corresponds to the “frequency” property in “FastNoiseLite”. This is again shown in Figure 5.7.

Initiating the noise variable is done in the “set_noise” function, in which, after instantiating the “FastNoiseLite” class, the relevant export variables are then taken in and assigned their values accordingly, with the noise type handled through the “_get_noise_type” function, which returns an integer that is then cast to the “NoiseType” enumeration. By contrast, all the other attribute assignments- “frequency”, “fractal_type” and “cellular_distance_function”- were far more simple, since we can just use the values of “noise_frequency”, “fractal_type” (*in the script’s export variable, **not** the noise instance*) and “cellular_distance_type”, and assign them accordingly. Both functions are shown in Figure 5.8.

In both Figures 5.7 and 5.8, there is a commented out export variable, “octaves”. This paper wanted to see if changing the number of octaves in our noise image (or, rather, “layers”) would have an effect on the level layouts designed (the default is 5[36]). Unfortunately, in the report author’s experiments, he found that it did not have any noticeable effect, so it was decided to leave the default number of octaves as is.

Painting the tiles in our tile map involves a “paint_points” method in which we iterate through every single cell of our tile map. For each cell in that map, we then check first if it can paint a tree. If the noise value retrieved at the given pixel from the noise image is smaller than the maximum limit we set for trees, “tree_cap”, then we paint a tree tile at that cell. Then, we check for buildings. As described in chapter 3.2.3, additional Boolean checks had to be implement for when “tree_cap” was smaller than “building_cap”, so that buildings could still get placed. First, we check that “building_cap” is smaller than *or equal to* “tree cap” **and**, if so, whether or not we overwrite the tree tile with a building tile in the same place, subject to a random float between 0 and 1 falling below the probability value set in “building_overtakes_tree”. The *alternative*, for when “building_cap” is smaller than “tree_cap”, is to check whether the current “noise_point” is smaller than the “building_cap” we set. In both cases, of course, that cell cannot already be occupied by something else. If those conditions are true, we can then paint a building tile. The code behind this part of the algorithm is shown in Figure 5.9.

```
var noise: FastNoiseLite
@export_enum("Perlin", "Simplex", "Simplex Smooth", "Value", "Value Cubic") var noise_type: String = "Simplex Smooth"
@export var fractal_type: FastNoiseLite.FractalType
@export var cellular_distance_type: FastNoiseLite.CellularDistanceFunction
#@export_range(1, 10, 1) var octaves: int = 5
@export_range(0.0, 1.0) var noise_frequency: float = 0.894
```

Figure 5.7: The noise, noise_type, fractal_type, cellular_distance_type and noise_frequency script variables in tile_map.gd. The latter three are export variables, and the latter two of *those* are assigned to the enumerations FractalType and CellularDistanceFunction repsectively (both are members of the FastNoiseLite class).[36] noise_type, meanwhile, uses an external function called to assign properties to a newly created FastNoiseLite instance which is assigned to the noise script variable. noise_frequency here corresponds to the frequency attribute in FastNoiseLite. Notice the commented out octaves variable. Changing the number of octaves used had no effect on the level layouts produced, so the default number of octaves (5) was left as is.

```
func _get_noise_type() -> int:
	match noise_type:
		"Perlin": return 3
		"Simplex": return 0
		"Value": return 5
		"Value Cubic": return 4
		_: return 1 # Return Simplex Smooth by default

func set_noise() -> void:
	noise = FastNoiseLite.new()
	noise.frequency = noise_frequency
	noise.noise_type = _get_noise_type() as FastNoiseLite.NoiseType
	noise.fractal_type = fractal_type
	noise.cellular_distance_function = cellular_distance_type
#	noise.fractal_octaves = octaves
	noise.seed = randi()
```

Figure 5.8: The set_noise method in the tile_map.gd script, which uses the earlier defined _get_noise_type method to assign the noise type. The noise type returned is then cast from an integer to an enumeration of type NoiseType (from FastNoiseLite). The fractal_octaves line is commented out because, when experimenting with octaves, there were no real observable effects they could have on the level layouts generated, so the default number of octaves (5) was left as is.

```
func paint_tiles() -> void:
    for x in range(x_tile_range):
        for y in range(y_tile_range):
            var noise_point: float = noise.get_noise_2d(x * tile_set.tile_size.x, y * tile_set.tile_size.y)
            if noise_point < tree_cap and not get_used_cells(0).has(Vector2i(x, y)):
                set_cell(0, Vector2i(x, y), 0, trees.pick_random())
            if ((building_cap <= tree_cap and randf() < building_overtakes_tree) or (building_cap > tree_cap and noise_point < building_cap)) and not get_used_cells(0).has(Vector2i(x, y)):
                set_cell(0, Vector2i(x, y), 0, buildings.pick_random())
```

Figure 5.9: The paint_tiles method in the tile_map.gd script iterates through the tile map and gets each noise pixel from the relevant part of the noise image. It first tries to paint a tree tile there, subject to the noise_point value being below the limit set for trees. Then, it decides whether or not to paint a building there. The conditions for painting building tiles are as described in chapter 3.2.3 and further elaborated on earlier in *this* chapter, 5.2.

5.3 Poisson Disk Sampling

To be able to access the inner and outer grid sizes in our implementation of this algorithm, since GDScript does not have a concept of different “Array”s and lists, the lengths of the inner and outer grid were stored in local variables in the “generate_points” function. Those local variables, “grid_x_axis_size” and “grid_y_axis_size” as shown in Figures 5.10 and 5.11, essentially store the same grid size values as in Lague’s implementation, right down to performing the same division in a ceiling function, to the inner grid and the outer grid respectively. Since these dimensions would also be needed for “is_valid”, instead of creating 2 more script variables, the code instead took them in as 2 additional method parameters, as shown in Figures 5.12 and 5.13, and used them accordingly when calculating the maximum and minimum bounds for searching the nearest points of the cell, as shown in 5.14. Doing it this way ensured that the computation of this algorithm would stay efficient and not stall with an adequate (not too high) number of rejection samples.

```
var grid_x_axis_size: int = ceili(sample_region_size.x/cell_size)
var grid_y_axis_size: int = ceili(sample_region_size.y/cell_size)
```

Figure 5.10: The lines used to determine the inner and outer dimensions of the grid array.

```
for i in range(grid_x_axis_size):
    grid.append([])
    for j in range(grid_y_axis_size):
        grid[i].append(0)
```

Figure 5.11: The nested for-loop that initialises the grid array. First, each inner array is initialised and inserted, then a number of zeroes, determined by the grid's y-dimension, are inserted.

```
if is_valid(candidate, sample_region_size, cell_size, radius,
    points, grid, grid_x_axis_size, grid_y_axis_size):
```

Figure 5.12: The line that uses the grid's x and y dimensions as parameters. This calls the is_valid method using those additional parameters (see Figure 5.13).

```
func is_valid(candidate: Vector2, sample_region_size: Vector2,
    cell_size: float, radius: float, points: Array[Vector2], grid:
    Array[Array], grid_x_axis_size: int, grid_y_axis_size: int) ->
    bool
```

Figure 5.13: The function is_valid, which takes in 2 additional parameters denoting the x and y dimensions of the grid array used in generate_points.

```
var search_end_x: int = min(cell_x + 2, grid_x_axis_size - 1)
var search_end_y: int = min(cell_y + 2, grid_y_axis_size - 1)
```

Figure 5.14: The relevant lines of code in is_valid that reference the grid's x and y dimensions, stored in additional variables as aforementioned.

5.4 Voronoï Cells

The original JavaScript implementation, as mentioned before, had a "randRange" function that was taken out, as it was not needed at all, but there was also an additional "mapSize" parameter in "definePoints" that, in *our* "define_points" function, didn't really need, since the code makes sure the map's dimensions were readily accessible via the "x_tile_range" and "y_tile_range" script variable. Therefore, the second parameter in "define_points" was taken out, as shown in Figure 5.15, and substituted it with "x_tile_range" and "y_tile_range" accordingly, as shown in Figure 5.17.

The type of each Voronoï cell was determined by taking, and then deleting, a value from the "types" array. Said array is local to that function, and it is initialised by duplicating the "trees" array, then appending it with the "buildings" array, making sure the same type cannot be used for a Voronoï cell twice. Duplicating the array before merging it essentially makes sure that the *original* "trees" array is not affected by deletions performed on the "types" array. This computation is shown in Figure 5.16, and the deletion operation is shown in Figure 5.18.

Another addition to our implementation of the algorithm was the choice of using either the Euclidean distance or Manhattan distance for calculating the distance between points that would form cells. This was done with a function "calculate_points_delta", as shown in Figure 5.23 and used in Figure 5.22, which is called on the calculation of "delta" in "define_points". The function takes the contents of the exported variable "distance", as well as the current "x" and "y" coordinates and point ID "p" during the current points delta calculation. It then checks if the String value in "distance" denotes either the Euclidean distance or the Manhattan distance, then it finally returns the appropriate calculation. Using the Manhattan distance instead of the Euclidean distance does indeed yield a considerable performance increase (as well as creating fewer Voronoï cells by using a smaller "random_starting_points" value), which we touch on in the Evaluation chapter.

Furthermore, our implementation would often paint cells in the tile map that were out of bounds, so to mitigate this when painting them, an additional function "_is_in_bounds" was written, as shown in Figure 5.21 and used in Figure 5.20, for checking whether a painted cell (that is, the coordinates of the current point **plus** the delta/difference between it and the closest of the randomly selected starting points to it) is within the boundaries of the tile map. If it is not, then it does not get painted, though it is not deleted from the point's citizens array either.

```
func define_points(num_points: int) -> void:
```

Figure 5.15: The define_points function header, with no argument for the map's size. The num_points value that gets taken in during runtime is determined by the script's export variable random_starting_points.

```
var types: Array[Vector2i] = trees.duplicate()
types.append_array(buildings)
```

Figure 5.16: The types array being initialised in define_points, with its values taken from the trees and buildings arrays, such that no type can be used for a cell twice, while also making sure that the original trees and buildings arrays are not affected by the deletions on types.

```
var x: int = randi_range(0, x_tile_range)
var y: int = randi_range(0, y_tile_range)
```

Figure 5.17: Godot's built-in randi_range function being used in place of a self-defined one in define_points.

```
var type: Vector2i = types.pick_random()
types.erase(type)
```

Figure 5.18: The types of each Voronoï cell being picked and the erased in define_points.

```
const EUCLIDEAN: String = "Euclidean distance"
const MANHATTAN: String = "Manhattan distance"
@export_enum(EUCLIDEAN, MANHATTAN) var distance: String = MANHATTAN
```

Figure 5.19: The applicable values of the exported variable distance.

```
if _is_in_bounds(point["x"], citizen["dx"], point["y"], citizen["dy"]):
    set_cell(0, Vector2(point["x"] + citizen["dx"], point["y"] + citizen["dy"]), 0, point["type"])
```

Figure 5.20: The appropriate block of code in the paint_points function that checks to see if a point would be in bounds or out of bounds before painting it in its relevant tile map cell. It does **not** delete the point if it lies out of bounds.

```
func _is_in_bounds(x: int, dx: int, y: int, dy: int) -> bool:
    return x + dx >= 0 and x + dx < x_tile_range and y + dy >= 0 and y + dy < y_tile_range
```

Figure 5.21: The _is_in_bounds function called in the code snippet in Figure 5.20.

```
var delta: float = calculate_points_delta(x, y, p)
```

Figure 5.23: The calling of calculate_points_delta from Figure 5.22 in define_points, using the current x and y coordinates and point ID p in the iteration when grouping tile map cells together to form Voronoï cells from the randomly selected starting points.

```
func _squared(x: int) -> int:
    return x ** 2

func calculate_points_delta(x: int, y: int, p: int) -> float:
    if distance == EUCLIDEAN:
        return sqrt(_squared(points[p]["x"] - x) + _squared(points[p]["y"] - y))
    return abs(points[p]["x"] - x) + abs(points[p]["y"] - y)
```

Figure 5.22: The calculate_points_delta function being called in Figure 5.23. _squared is a self-defined helper function that is only used for the Euclidean distance calculation; it does as it says (it squares the number taken into it and returns the result).

Chapter 6

Legal, Social, Ethical and Professional Issues

Throughout the course of this project, the author of this report made sure he abode by the principles set out in the Code of Conduct & Code of Good Practice issued by the British Computer Society[53], acting with integrity, honesty and transparency in the way potential licensing issues with the work produced for this dissertation, and the other work used as both reference and inspiration, were all properly handled. Throughout this report, the ways external code, articles and other references were used in both the writing and the software artefacts have been thoroughly discussed and also elaborated on. References and inspirations for code in those artefacts were also clearly and transparently denoted via the inclusion of comments in script files. To further ensure full transparency in the research done behind this project, every single citation, even remotely tangental ones, are cited in the bibliography of this report as appropriate. In this chapter, the ways in which the licenses of code references in the implementations were adequately dealt with are detailed in the following section 6.1, while the details on plans behind the author of this report releasing his *own* code and report, both for public access, are detailed in section 6.2.

6.1 Using Other People's Resources

As this project has been continually worked on, it has been ensured, with confidence, that the resources used were freely available to use in an academic context such as this.

For example, the Unity tutorial used as an inspiration of our Godot Poisson Disk Sampling implementation[33] has its project files under the MIT License[34], a permissive open-source license which means it can be freely used and adapted with, even commercially.[43] This meant that our Godot implementation could use his Unity implementation as a basis without fear of any legal implications. Nonetheless, to act with integrity, it has been denoted properly, in this report and in code comments, that his work has been taken from and adapted, citing it accordingly in the bibliography as well.

As aforementioned in chapter 3.2.5, the JavaScript code example taken from the Procedural Content Generation wiki[16], for the Voronoï Cells implementation, was submitted by an anonymous Wikidot contributor in 2017. Like most if not all of the Wiki's contents, it is licensed under the Creative Commons Attribution-ShareAlike 3.0 License (all contents of the wiki follow this license unless otherwise specified); that is, the article and its contents (including the JavaScript code example) can be freely used and adapted, subject to the condition that the original source is attributed **and** that any transformed work, *like the Godot implementation*, **must** be published under the same or a compatible license.[6] Since there are no listed compatible source code licenses that can be use in lieu of this license[8], the Voronoï cells implementation must therefore abide by the license contents of the original article in the source code, since both it and the original JavaScript code are similar to a noticeable, although not entirely like for like, degree (even though *our* implementation is in GDScript and not JavaScript). We will go over how all of the project artefacts will eventually be released to the public, as well as the LaTeX source code and BIBTeX citations of this report, in the following section 6.2.

Projects that have been used as references on a smaller scale were also accounted for. While the Godot 3 TileMap noise tutorial referenced in chapter 3.2.3 is up on GitHub, it has no readily attached license to it.[22] However, since no substantial code from it whatsoever has been taken from or adapted, and the scripting APIs for Godot 3 and Godot 4 are vastly different, *especially* in the context of tile maps, it is therefore highly justified that *our* implemented code will not pose an issue, and so this, and all the other self-produced artefacts, can be posted on GitHub under conditions that are further explained in the following section 6.2.

To produce the screenshot in Figure 3.6, the author of this report forked an existing Godot 3 project on GitHub[24], taken from a YouTube video tutorial on how to use an L-System to draw line graphics in Godot[23], converted it to Godot 4 and added an additional set of rules to it based on the example lattice grammar featured in chapter 3.1.1.[25] While the conversion was done by the author of this report, and some other own code contributions of his were also to the fork, the author of this report does not regard this as a substantial part of his project, and thus have not included it in the source code listings in chapter C. The person behind the code has previously denoted appreciation of other people's forks of his code, so the lack of readily available code license in his original repository is not believed to be a substantial issue here. Nonetheless, since small parts of his code have been adapted to work with the L-System implementation, he has been emailed him directly for his permission to do both that and add the permissive MIT license to the new fork, and his permission was received from a private email conversation had between the author of this report and him on Tuesday 18$^{\text{th}}$ April 2023:

"Well, haven't done anything with the channel in years, everything is more or less up for grabs. So you have my permission to use anything and/or everything however you like. You can add the MIT license to your fork.

You can refer to this conversation if needed!"

He has been made aware that his work will be properly and clearly cited in both the dissertation and the released artefact. The MIT license has already added the license to the new fork.[25]

Any usage of external screenshots in Figures throughout this document a are properly cited and linked to in the bibliography. Screenshots that were self-produced by the author of this report are clearly denoted as such in Figure captions. Any usage of code snippets in Figures were written by him, and any external references used as bases for these snippets were clearly and properly cited as such.

6.2 How The Author's Own Artefacts Will Be Released

The author of this report has planned for his source code to both the dissertation and artefacts to be released on GitHub for public access. In order to do so, all of his repositories must be properly assigned licenses so that his usage intentions and any repository usage conditions are clearly defined.

For both the report and all the artefacts, the Creative Commons Attribution-Sharealike 4.0 license, as described in the previous section, was chosen and assigned. The aforementioned license allows for commercial and non-commercial usage on the condition that (1) the concerned product is properly attributed to when used and (2) any adaptations of and modifications to this work are released under the same license (or a compatible one, or a later revision of it).[7] The concerned product can be used *verbatim* (i.e. as is) without having to share their work under this license, but when it is adapted upon, *then* the share-alike conditions apply.

Although it is not widely considered good practice to apply Creative Commons licenses to code[9], the author still believes that CC-BY-SA-4.0 is the best license for his project overall. As well as resolving any complications with the Voronoï cells implementation, again as discussed in the previous section (CC-BY-SA-4.0 is compatible with the CC-BY-SA-3.0 license used in the original JavaScript code, as CC-BY-SA-3.0 allows for licensing newly adapted works under later versions of the license[8]), it also ensures that all of the original Godot implementations are still available for public viewing and modification while also ensuring others can still modify it and everyone else who *isn't* modifying it has the freedom to view these modifications. In that regard, it *is* similar to copyleft licenses such as the GPL and LGPL, though, as of the time of this publication, only the former is listed as compatible with CC-BY-SA-4.0.[8]

In an academic context, it ensures that all of the work and code that has been put into this report by the author of this report, as well as the implementations, are still available when the code is taken and then built upon by someone else (and that these modifications are also made available for others to see how the author's work was built upon). Applying the license to this dissertation, as well as the artefacts, allows him to ensure that all aspects of the work the author has done are viewable by everyone, and any improvements made to this work, by him or anyone else, are also viewable and publicly recognised.

Note on the Godot Project Used to Create Some Screenshots in Chapter 3.1.1 & Why It Has Been Included With The Rest of The Artefacts

Some of the self-produced screenshots, specifically the ones in Figures 3.1, 3.2, 3.3 and 3.4, are taken from a Godot project created by the author of this report early on, during the learning process on how L-Systems worked for this dissertation. It has eventually been decided to include it in the source code listings in chapter C, primarily because much of the code in there (specifically in the script file DemoNode.gd) is used in the final project, but also because the commit history shows the process the author of this report initially went through in building an L-System that could handle multiple grammar rules, as detailed in chapters 3.1.1, 3.2.2 and 5.1. Do note, however, that it is not as important to the project's motives as the main four algorithm implementations.

Chapter 7

Results & Evaluation

Here, we will discuss how the implementations of the algorithms in our scenario were tested and ensured that they ran as they should.

7.1 Software Testing

Due to the nature of the project (being several implementations of a computer game), the testing behind this project has solely revolved around trial-and-error, messing around with the exported variables in the Godot editor to see how things worked and what configurations worked best for our scenario. This involved taking many screenshots of generated levels and examining things by eye, seeing how layouts compared across implementations.

Despite this, it was eventually decided to run some simple performance tests to see how long each algorithm ran. These tests took some of the custom export variables from the scene scripts and ran them several times, with an average time calculated to the nearest millisecond. The results of these tests are all in table in the Appendix.

7.2 Comparing the Different Algorithms and Drawing Conclusions on Which Ones Are Best

7.2.1 Performance

With the L-System implementation, there were no problems whatsoever running the game very quickly on the author of this report's machine, and quickly got satisfactory results. The table in Figure A.3 shows that the processing times remained miniscule even as the length of the axiom increased.

While the Noise implementation was slower than the L-System implementation by a magnitude, it was still satisfactorily quick. I ran some timed tests in which I tested how some of the properties affected the time it took to generate the noise. These tests are referred to in tables A.1, in which each noise algorithm was paired with each cellular distance function, and A.2, in which each noise algorithm was paired with each fractal type.

With Poisson Disk Sampling, the higher the number of rejection samples (that is, the higher the maximum number of times a cell was sampled before it was either accepted or ultimately rejected), the longer it took to generate a complete level layout, and even, due to the nature of the tile map compared to the algorithm's *usual* use (of scattering dots on a plane), it was not maximal (not all points had cells painted for them; some cells had their tiles overwritten as well). Using 8 rejection samples was usually enough to yield a satisfactory level layout while also keeping level creation times to a minimum.

Voronoï Cells took the longest to compute on average. Computations with the Euclidean distance measurement took longer than those measured with the Manhattan distance.

7.2.2 Layouts

Of the 4 implementations that were made for this project, the Noise and Poisson Disk Sampling implementation were by far the most similar, followed by the L-System implementation, and then the Voronoï Cells implementation, which was far and away the most unique.

While the noise implementations varied greatly depending on what settings were used, and the way the implementation was designed allowed for very many possibilities as to how the noise would turn out (and how it would affect the final level), the results that were returned produced the most similar results to that of the Poisson Disk Sampling implementation had the following configurations:

- Noise Type ("noise_type"): Simplex Smooth
- Fractal Type ("fractal_type"): Fractal None
- Cellular Distance Type/Function ("cellular_distance_type"): Distance Euclidean
- Noise Frequency ("noise_frequency"): 0.894
- Tree Cap ("tree_cap"): -0.048
- Building Cap ("building_cap"): -0.252

- Building Overtakes Tree ("building_overtakes_tree"): 0.12

The default noise frequency in "FastNoiseLite" is 0.01, which results in smoother and less disparate noise. As seen in Figure 7.2, the smoother noise and lower frequency results in a distinct kind of level layout in which represents some of the noise values in the image very clearly, such that tiles (both buildings and trees) are bunched together in partially interconnected groups, forming long, large lines of painted tiles. To describe this as best as possible, it is easy to discern that the level layout was determined from a noise image. Using a higher noise frequency to produce rougher noise, and more disparate level layouts, yields results like in Figure 7.1, which makes it very similar to the layouts yielded in the Poisson Disk Sampling implementation and, to a lesser extent, the L-System implementation. While the author of this report's personal tastes are fond of the former kind of level layout, part of the aim of this project was to compare in terms of which could produce the most similar, and, compared to the L-System and Poisson Disk Sampling implementations, the Noise implementation with the frequency set to 0.01 was far too distinct, hence the want to change it up.

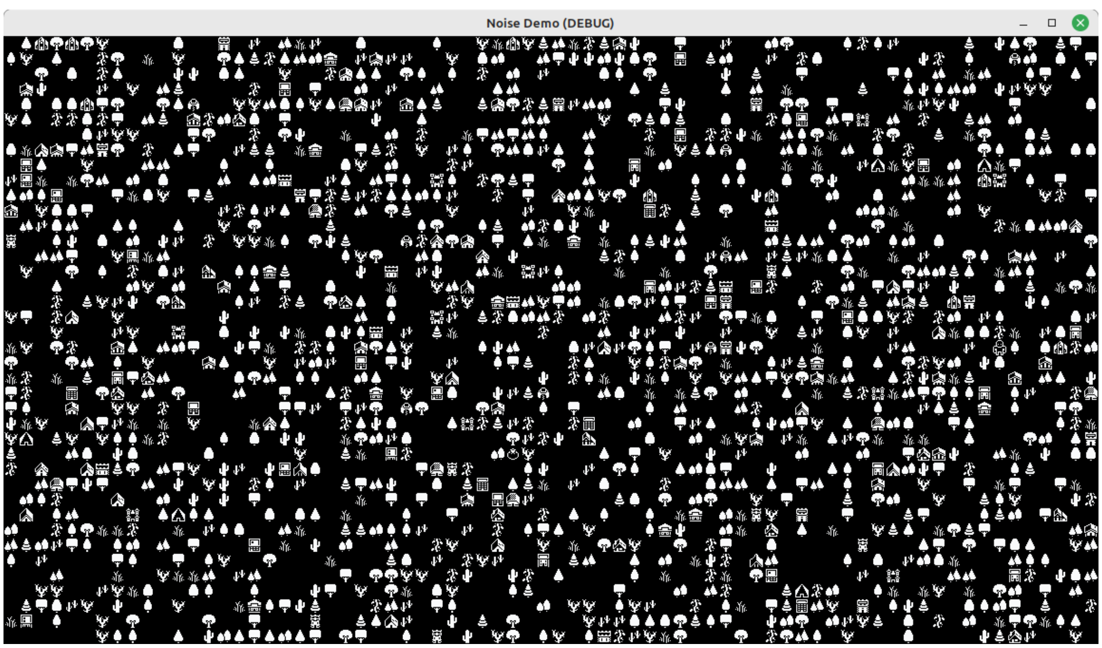

Figure 7.1: A level of our scenario generated in the Simplex Noise implementation, using all of the default values shown here. The level took a total of 99 milliseconds to be made.

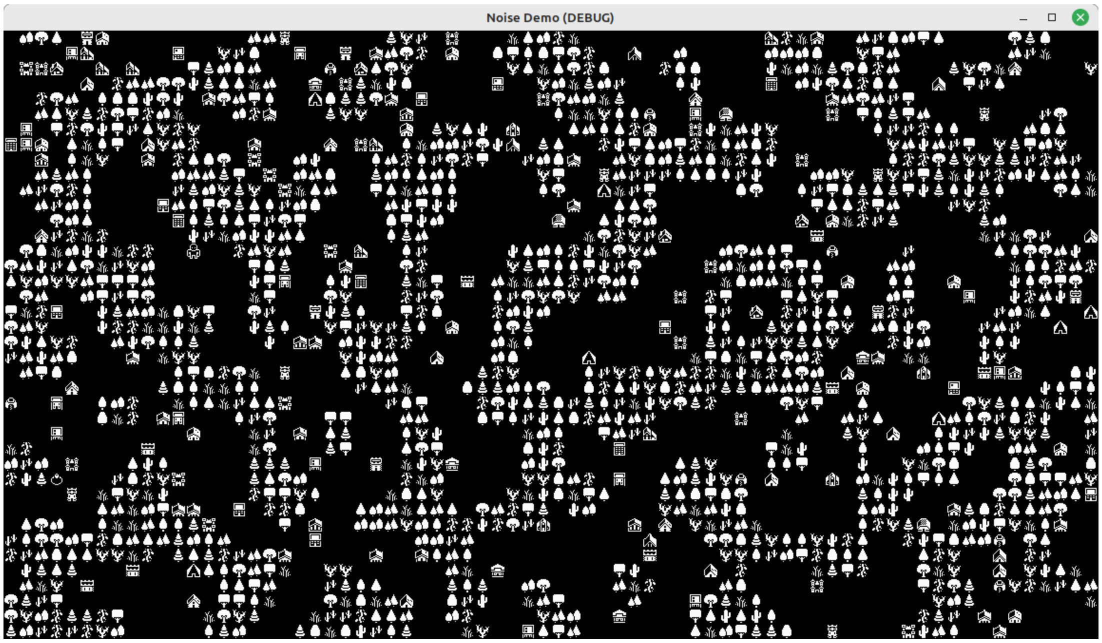

Figure 7.2: A level of our scenario generated in the Simplex Noise implementation, setting the noise frequency to 0.01 (the default value for noise frequency in "FastNoiseLite") and using the rest of the defaults shown here. The level took a total of 104 milliseconds to be made.

The

Chapter 8

Conclusion and Future Work

To conclude, the author of this report gained a wealth of knowledge about the way some of the most popular procedural content generation algorithms work, and how they are typically integrated into working games. He also learnt how he could leverage the features of the Godot game engine for some of them; for example, the "FastNoiseLite" class allows a Godot game developer to generate noise textures in Value, Perlin and even Simplex noise and then modify them accordingly with additional frequency settings, fractal types and cellular distance functions. By implementing them in a self-designed 2D tiled RPG scenario, he was able to get 4 procedural generation algorithms well-integrated into working games, proving Godot's technical proficiency in making these kinds of games work, and proving his own abilities as a games programmer. He was also able to compare the implementations of his chosen algorithms in such a way that the differences, in terms of both performance times and the kinds of levels they produced, could very easily be discerned. The motives of this project can be pushed still further by measuring and comparing the performances of these algorithms in Big-O notation, including even more ontogenic algorithms such as Worley Noise, the Diamond-Square algorithm, Markov Chains and Cellular Automata, as well as telelogical algorithms such as the Rain Drop algorithm and Reaction-Diffusion systems, using a larger tile map on all of these algorithms and even using a different, more intensive scenario entirely, such as a 3D walking simulator/open-world game. With procedural generation for level design, the possibilites are practically endless.

References

[1] Joseph Bakulikira. Poisson disc sampling in pygame. `https://github.com/Josephbakulikira/Poisson-Disc-Sampling-with-pygame`. [Online; accessed 15-April-2023].

[2] Paul Bourke. L-system user notes. `http://paulbourke.net/fractals/lsys`, July 1991.

[3] Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, page 22–es, New York, NY, USA, 2007. Association for Computing Machinery.

[4] Mark Brown. How (and why) spelunky makes its own levels. `https://youtu.be/Uqk5Zf0tw3o`, April 2016. [Online-accessed 10-April-2023].

[5] Chris Bradfield and contributors. Exploring procedural generation algorithms in godot. `https://github.com/kidscancode/godot3_procgen_demos`.

[6] Creative Commons. Attribution-sharealike 3.0 unported (cc by-sa 3.0). `https://creativecommons.org/licenses/by-sa/3.0/`. [Online, Accessed 18-April-2023].

[7] Creative Commons. Attribution-sharealike 4.0 international (cc by-sa 4.0). `https://creativecommons.org/licenses/by-sa/4.0/`. [Online, Accessed 18-April-2023].

[8] Creative Commons. Compatible licenses. `https://creativecommons.org/share-your-work/licensing-considerations/compatible-licenses`. [Online, Accessed 18-April-2023].

[9] Creative Commons. Frequently asked questions — can i apply a creative commons license to software? `https://creativecommons.org/faq/#can-i-apply-a-creative-commons-license-to-software`. [Online, Accessed 18-April-2023].

[10] Timm (Stack Exchange Contributor). "benefit of perlin noise over value noise" - stack exchange answer. https://computergraphics.stackexchange.com/a/3609, June 2016. [Online, Accessed 16-April-2023].

[11] RogueBasin Contributors. Rogue - roguebasin. http://www.roguebasin.com/index.php/Rogue. [Online, Accessed 20-April-2023].

[12] Wikidot contributors. Ontogenetic - procedural content generation wiki. http://pcg.wikidot.com/pcg-algorithm:ontogenetic.

[13] Wikidot contributors. Perlin noise - procedural content generation wiki. http://pcg.wikidot.com/pcg-algorithm:perlin-noise.

[14] Wikidot contributors. Simplex noise - procedural content generation wiki. http://pcg.wikidot.com/pcg-algorithm:simplex-noise.

[15] Wikidot contributors. Telelogical - procedural content generation wiki. http://pcg.wikidot.com/pcg-algorithm:teleological.

[16] Wikidot contributors. Voronoi diagram - procedural content generation wiki. http://pcg.wikidot.com/pcg-algorithm:voronoi-diagram.

[17] Mohamed S. Ebeida, Andrew A. Davidson, Anjul Patney, Patrick M. Knupp, Scott A. Mitchell, and John D. Owens. Efficient maximal poisson-disk sampling. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, New York, NY, USA, 2011. Association for Computing Machinery.

[18] Joan Fons. Simplex noise lands in godot 3.1. https://godotengine.org/article/simplex-noise-lands-godot-31, September 2018.

[19] Jonas Freiknecht and Wolfgang Effelsberg. A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*, 1:27, 10 2017.

[20] Jonas Freiknecht and Wolfgang Effelsberg. A survey on the procedural generation of virtual worlds. *Multimodal Technologies and Interaction*, 1(4):27, 2017.

[21] Gingerageous Games. Autotiles opensimplex noise procedural generation godot 3.1 tutorial. https://youtu.be/SBDs8hbs43w, August 2019. [Online, Accessed 11-March-2023].

[22] Gingerageous Games. Opensimplexnoisetilemaptutorial. https://github.com/gingerageous/OpenSimplexNoiseTilemapTutorial, March 2020. [Online, Accessed 11-March-2023].

[23] Alexander Gillberg. Code that: L-system. https://youtu.be/eY9XkJERiG0, 2020.

[24] Alexander Gillberg and Christian Bartsch. An implementation of the rewriting system: Lindenmayer system. https://github.com/codatproduction/Godots-L-System, April 2020. [Online; accessed 10-April-2023].

[25] Alexander Gillberg, Zishan Rahman, and Christian Bartsch. An implementation of the rewriting system: Lindenmayer system. https://github.com/Zishan-Rahman/Godots-L-System, March 2023. [Online; accessed 10-April-2023, License MIT].

[26] Hernan J. González. "why is gradient noise better quality than value noise?" - stack exckange answer. https://math.stackexchange.com/a/184153, August 2012. [Online, Accessed 16-April-2023, Stack Exchange Username "leonbloy"].

[27] Hristo Hristov. An introduction to the voronoi diagram. https://www.baeldung.com/cs/voronoi-diagram, November 2022. [Online, Accessed 18-April-2023].

[28] Ariel Manzur Juan Linietsky and the Godot community. Opensimplexnoise — godot engine (3.6) documentation in english. https://docs.godotengine.org/en/3.6/classes/class_opensimplexnoise.html. [Online, Accessed 17-April-2023, License CC-BY-3.0].

[29] KDotJPG. Noise! https://web.archive.org/web/20160529190516/http://uniblock.tumblr.com/post/97868843242/noise, September 2014. [Online, Accessed 17-April-2023].

[30] KdotJPG and contributors. Opensimplexnoise. https://github.com/KdotJPG/OpenSimplex2, 2014-present. [Online, Accessed 17-April-2023].

[31] Kenney. 1-bit pack (version 1.2). https://kenney.nl/assets/1-bit-pack, November 2021. [Online, Accessed 23-March-2023, License CC0-1.0].

[32] Marek Kopel and Grzegorz Maciejewski. Comparison of procedural noise-based environment generation methods. In Ngoc Thanh Nguyen, Bao Hung Hoang, Cong Phap Huynh, Dosam Hwang, Bogdan Trawiński, and Gottfried Vossen, editors, *Computational Collective Intelligence*, pages 878–887, Cham, 2020. Springer International Publishing.

[33] Sebastian Lague. [unity] procedural object placement (e01: poisson disc sampling). https://youtu.be/7WcmyxyFO7o, November 2018. [Online, Accessed 11-April-2023].

[34] Sebastian Lague. Poisson-disc-sampling. https://github.com/SebLague/Poisson-Disc-Sampling, November 2018-2020. [Online, Accessed 11-April-2023, License MIT].

[35] Aristid Lindenmayer. Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology*, 18(3):300–315, 1968.

[36] Juan Linietsky, Ariel Manzur, and the Godot community. Fastnoiselite — godot engine (stable) documentation in english. https://docs.godotengine.org/en/4.0/classes/class_fastnoiselite.html.

[37] Juan Linietsky, Ariel Manzur, and the Godot community. Gdscript — godot engine (4.0) documentation in english. https://docs.godotengine.org/en/4.0/classes/class_gdscript.html.

[38] Juan Linietsky, Ariel Manzur, and the Godot community. Projectsettings — godot engine (stable) documentation in english. https://docs.godotengine.org/en/4.0/classes/class_projectsettings.html#properties. [Online, Accessed 16-April-2023, GDScript Warnings Classed as Properties, License CC-BY-3.0].

[39] Don P. Mitchell. Spectrally optimal sampling for distribution ray tracing. *SIGGRAPH Comput. Graph.*, 25(4):157–164, jul 1991.

[40] Don P. Mitchell. Spectrally optimal sampling for distribution ray tracing. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, page 157–164, New York, NY, USA, 1991. Association for Computing Machinery.

[41] Nathan Lovato and contributors. Godot procedural generation. https://github.com/GDQuest/godot-procedural-generation.

[42] Gabriela Ochoa. An introduction to lindenmayer systems: D0l-system. https://www1.biologie.uni-hamburg.de/b-online/e28_3/lsys.html#D0L-system, February 1998. [Online; accessed 11-April-2023].

[43] Massachusetts Institute of Technology et al. The mit license - open source initiative. https://opensource.org/license/mit/. [Online, Accessed 18-April-2023].

[44] Guilherme Oliveira. “intro to procedural levels in godot 3.1: Perlin noise tutorial”. https://youtu.be/m6mu4uPGrMk, November 2018.

[45] Kenneth H. Perlin. Standard for perlin noise. https://patents.google.com/patent/US6867776, January 2002. [Online patent document, Accessed 16-April-2023].

[46] Michael Powell. Spirit of iron: Simplectic noise. https://web.archive.org/web/20160404092918/http://www.spiritofiron.com/2015/01/simplectic-noise.html, January 2015. [Online, Accessed 17-April-2023].

[47] Adarsh Pyarelal, Aditya Banerjee, and Kobus Barnard. Modular procedural generation for voxel maps. In Nikolos Gurney and Gita Sukthankar, editors, *Computational Theory of Mind for Human-Machine Teams*, pages 85–101, Cham, 2022. Springer Nature Switzerland.

[48] Philippe Rivière and Jacob Rus. Poisson disk sampling functions. https://observablehq.com/@fil/poisson-distribution-generators, June 2022. [Online, Accessed 17-April-2023].

[49] Martin Roberts. Maximal poisson disk sampling: an improved version of bridson’s algorithm. http://extremelearning.com.au/an-improved-version-of-bridsons-algorithm-n-for-poisson-disc-sampling/, November 2019. [Online, Accessed 17-April-2023].

[50] Robson. An interactive webpage for generating images that simulate white noise. https://robson.plus/white-noise-image-generator/, April 2021. [Online, Accessed 16-April-2023].

[51] Jacob Rus. A fork of roberts’s tweak to bridson’s algorithm for poisson disc sampling. https://observablehq.com/@jrus/bridson-fork/2, June 2022. [Online, Accessed 17-April-2023].

[52] Tim Smith. Infiniteness and boundedness in 0l, dt0l, and t0l systems. In Adrian-Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications*, pages 535–546, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[53] British Computer Society. Code of conduct for bcs members, version 8. https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf, June 2022. [Online, Accessed 18-April-2023].

[54] Corina Strößner. Criteria for naturalness in conceptual spaces. *Synthese*, 200, 04 2022.

[55] Matt Tillman. Manhattan voronoi approximation. http://www.pixelninja.design/manhattan-voronoi-approximation/, April 2017.

[56] Eric W. Weisstein. ”voronoi diagram.” from mathworld–a wolfram web resource. https://mathworld.wolfram.com/VoronoiDiagram.html. [Online, Accessed 18-April-2023].

[57] Mick West. The inner product: A shattered reality. https://ubm-twvideo01.s3.amazonaws.com/o1/vault/GD_Mag_Archives/GDM_August_2006.pdf, August 2006. [Online, Accessed 17-April-2023, Pages 34-38].

[58] Mick West. The inner product: Random scattering. https://ubm-twvideo01.s3.amazonaws.com/o1/vault/GD_Mag_Archives/GDM_JuneJuly_2007.pdf, June 2007. [Online, Accessed 19-April-2023, Pages 33-35].

[59] Mick West. Random scattering: Creating realistic landscapes. http://web.archive.org/web/20080826103432/http://www.gamasutra.com/view/feature/1648/random_scattering_creating_.php?page=2, August 2008.

[60] Kate Xagoraris. Using l-systems. https://www.katexagoraris.com/houdini-l-systems. [Online, Accessed 19-April-2023].

Appendix A

Extra Information

A.1 Tables & Test Cases

This section contains tables and test cases mentioned in the Evaluation section of this report. They are included here as they remain peripheral to the main body of this report, and would break up the theme and flow of the text if it appeared in the body, as would the source code listings featured in chapter C and the detailed user guide featured in chapter B.

	Noise type				
Cellular Distance Function	Simplex Smooth	Simplex	Perlin	Value	Value Cubic
Euclidean	81ms 84ms 83ms AVG: 83ms	83ms 88ms 83ms AVG: 85ms	80ms 74ms 74ms AVG: 76ms	82ms 84ms 83ms AVG: 83ms	92ms 105ms 74ms AVG: 90ms
Euclidean Squared	81ms 77ms 84ms AVG: 81ms	74ms 91ms 79ms AVG: 81ms	81ms 73ms 80ms AVG: 78ms	90ms 119ms 93ms AVG: 101ms	76ms 109ms 82ms AVG: 89ms
Manhattan	83ms 107ms 82ms AVG: 91ms	93ms 101ms 87ms AVG: 94ms	82ms 72ms 80ms AVG: 78ms	80ms 81ms 94ms AVG: 85ms	81ms 72ms 101ms AVG: 85ms
Hybrid (Euclidean & Manhattan)	77ms 96ms 82ms AVG: 85ms	87ms 122ms 87ms AVG: 99ms	85ms 74ms 83ms AVG: 81ms	85ms 85ms 87ms AVG: 86ms	76ms 77ms 77ms AVG: 77ms

Figure A.1: A table denoting some performance tests done with comparing the noise algorithms, the cellular distance functions and the combination pairs between them. This was done on the Noise implementation of the game; the "tests" were simply checking how long it took to create levels on the author of this report's computer, and all the other script variables were assigned to their default values as described in chapter 4. Each noise type and cellular distance function pair was run 3 times, with the mean time (including potential outliers) calculated afterwards *to the nearest integer*. Be advised that the author of this report did these tests on his computer, so on different computers, results can, and likely *will*, vary.

	Noise type				
Fractal Type	Simplex Smooth	Simplex	Perlin	Value	Value Cubic
None	92ms 86ms 123ms AVG: 100ms	89ms 152ms 99ms AVG: 113ms	84ms 98ms 95ms AVG: 92ms	101ms 86ms 97ms AVG: 95ms	77ms 88ms 86ms AVG: 84ms
FBM (Fractional Brownian Motion)	77ms 93ms 87ms AVG: 86ms	81ms 87ms 93ms AVG: 87ms	73ms 79ms 73ms AVG: 75ms	78ms 137ms 82ms AVG: 99ms	68ms 64ms 87ms AVG: 73ms
Ridged	23ms 25ms 23ms AVG: 24ms	74ms 69ms 70ms AVG: 71ms	15ms 16ms 16ms AVG: 16ms	27ms 28ms 26ms AVG: 27ms	14ms 9ms 11ms AVG: 11ms
Ping Pong	59ms 54ms 58ms AVG: 57ms	67ms 77ms 64ms AVG: 69ms	108ms 105ms 111ms AVG: 108ms	128ms 71ms 72ms AVG: 90ms	163ms 172ms 164ms AVG: 166ms

Figure A.2: A table denoting some performance tests done with comparing the noise algorithms, the fractal types and the combination pairs between them. This was done on the Noise implementation of the game; the “tests” were simply checking how long it took to create levels on the author of this report’s computer, and all the other script variables were assigned to their default values as described in chapter 4. Each noise type and fractal type pair was run 3 times, with the mean time (including potential outliers) calculated afterwards *to the nearest integer*. Be advised that the author of this report did these tests on his computer, so on different computers, results can, and likely *will*, vary.

use_custom_axiom = false axiom = "OWB"	use_custom_axiom = true upper_limit = 3	use_custom_axiom = true upper_limit = 10	use_custom_axiom = true upper_limit = 25
21ms	25ms (length = 2)	20ms (length = 4)	9ms (length = 25)
17ms	13ms (length = 2)	11ms (length = 9)	14ms (length = 2)
21ms	21ms (length = 1)	21ms (length = 8)	21ms (length = 21)
20ms	16ms (length = 1)	18ms (length = 5)	20ms (length = 24)
20ms	11ms (length = 2)	11ms (length = 4)	15ms (length = 14)
AVG: 20ms	AVG: 17ms	AVG: 16ms	AVG: 16ms

Figure A.3: A table denoting some performance tests done with comparing the lengths of axioms used in L-Systems. Obviously, this was done on the L-System implementation of the game; the "tests" were simply checking how long it took to create levels on the author of this report's computer, as well as how long the randomly generated axioms were (where appropriate), and all the other script variables were assigned to their default values. Each of the shown settings were run 5 times, with the mean time (including potential outliers) calculated afterwards *to the nearest integer*. Be advised that the author of this report did these tests on his computer, so on different computers, results can, and likely *will*, vary.

rejection_samples	3	8	13	18
time	170ms	337ms	444ms	503ms
	103ms	224ms	392ms	505ms
	111ms	242ms	388ms	670ms
	AVG: 128ms	AVG: 268ms	AVG: 408ms	AVG: 559ms

Figure A.4: A table denoting some performance tests done with comparing the number of rejection samples used for Poisson Disk Sampling. Obviously, this was done on the Poisson Disk Sampling/Distribution implementation of the game; the "tests" were simply checking how long it took to create levels on the author of this report's computer, and all the other script variables were assigned to their default values. Each of the shown settings were run 3 times, with the mean time (including potential outliers) calculated afterwards *to the nearest integer*. The bottom cell of the rightmost column, with tests done with 18 rejection samples, is highlighted red because, while testing with 18 rejection samples, at one time the program hung without returning any cell points within 10 seconds. The test had to be retaken another time. Be advised that the author of this report did these tests on his computer, so on different computers, results can, and likely *will*, vary.

	Random Starting Points			
Distance type	15	20	30	40
Euclidean distance	393ms 385ms 362ms AVG: 380ms	496ms 504ms 497ms AVG: 499ms	775ms 744ms 723ms AVG: 747ms	968ms 970ms 967ms AVG: 968ms
Manhattan Distance	364ms 346ms 346ms AVG: 352ms	441ms 470ms 437ms AVG:449ms	645ms 630ms 650ms AVG: 642ms	843ms 835ms 908ms AVG: 862ms

Figure A.5: A table denoting some performance tests done with comparing the distance calculation algorithms, the number of random starting points and the combination pairs between them. This was done on the Voronoï Cells implementation of the game, and thus the number of random starting points corresponds directly to the number of unique Voronoï cells generated for each level. The "tests" were simply checking how long it took to create levels on the author of this report's computer. Each of the shown settings were run 3 times, with the mean time (including potential outliers) calculated afterwards *to the nearest integer*. Be advised that the author of this report did these tests on his computer, so on different computers, results can, and likely *will*, vary.

L-System	Perlin/Simplex Noise	Poisson Disk Sampling/Distribution	Voronoï Cells
24ms	78ms	176ms	502ms
10ms	78ms	190ms	425ms
14ms	82ms	210ms	455ms
14ms	91ms	216ms	449ms
17ms	75ms	193ms	442ms
AVG: 16ms	AVG: 81ms	AVG: 197ms	AVG: 455ms

Figure A.6: A table denoting some performance tests done with comparing all of the algorithms implemented with the chosen scenario. This was done on every single implementation implementation of the game; the "tests" were simply checking how long it took to create levels on the author of this report's computer, and every implementation was tested with its default variable values. Each of the implementations were run 5 times, with the mean time (including potential outliers) calculated afterwards *to the nearest integer*. Be advised that the author of this report did these tests on his computer, so on different computers, results can, and likely *will*, vary.

Appendix B

User Guide

B.1 Opening Godot

To run the projects in the .zip file, extract the projects into one folder. Then open Godot 4 (all projects in the source code listings folder are Godot 4 projects, **not** Godot 3 projects). When you start Godot for the first time, the project manager should be completely empty, without any projects, as described in Figure B.1. Projects have to be imported either one-by-one (by clicking "Import" and going to the relevant project and opening it) or by clicking "Scan", then going to a folder of Godot projects and selecting it. The projects can then be opened in the project manager and edited as needed in Godot. Click "Scan", then go to the folder where you extracted the projects, then click the "Select Current Folder" button, as shown in Figure B.2, and all the projects should show up in the editor, as shown in Figure B.3. You can then double click on any one project (or click on it once and click the "Edit" button) to open it in the Godot editor, an example of which is shown in Figure B.4. Alternatively, to run the project itself without opening the editor, using the currently saved values for exported script variables where appropriate, click on the project *once* and click the "Run" button.

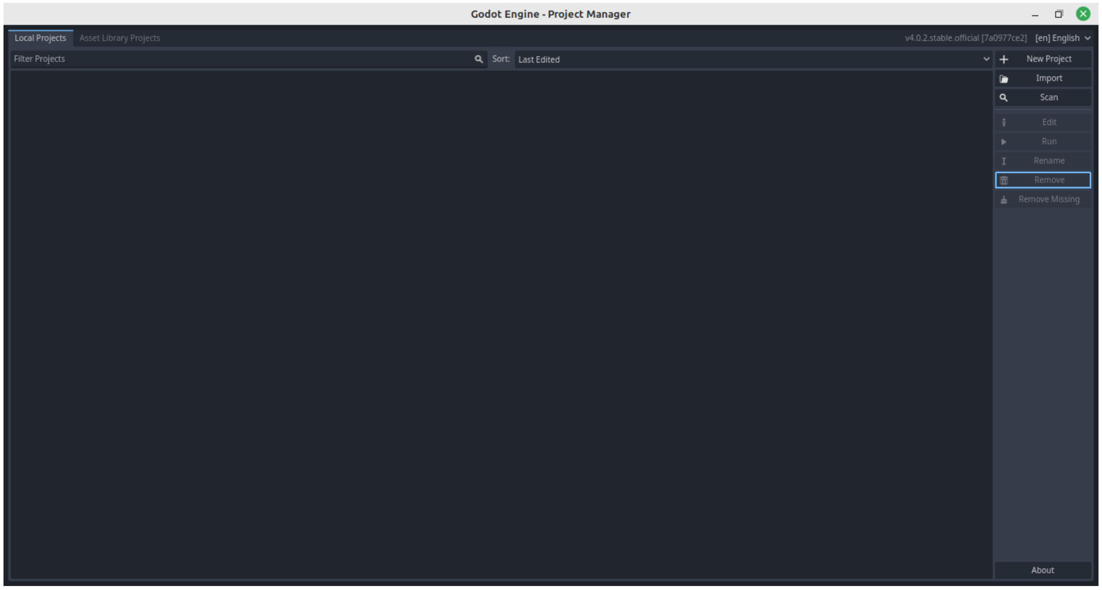

Figure B.1: The Godot editor, when it is opened for the first time, does not show any projects in the editor (the Steam version bundles several example projects). Projects need to be imported either one-by-one or by scanning a folder of Godot projects.

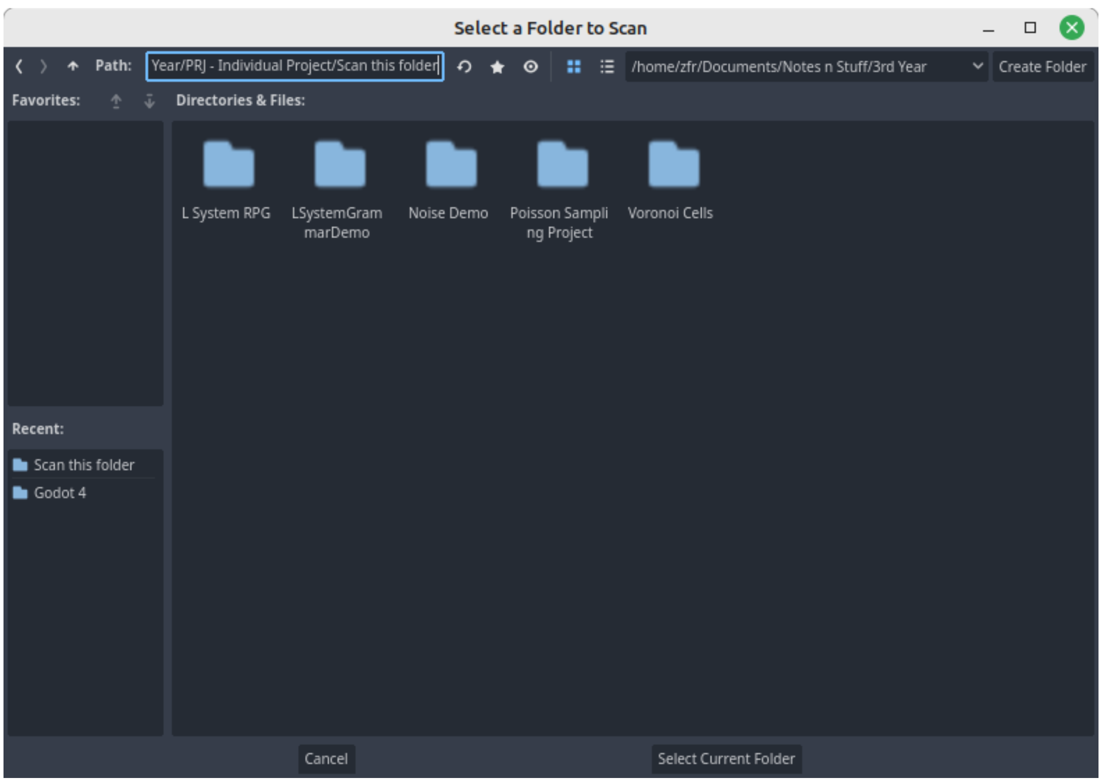

Figure B.2: You can click the ”Scan” button in the project manager (in Figure B.1), then go to the relevant folder where your project are in Godot’s built-in file explorer. Here, the artefacts behind this project have been exported into a separate folder called “Scan this folder” as an example.

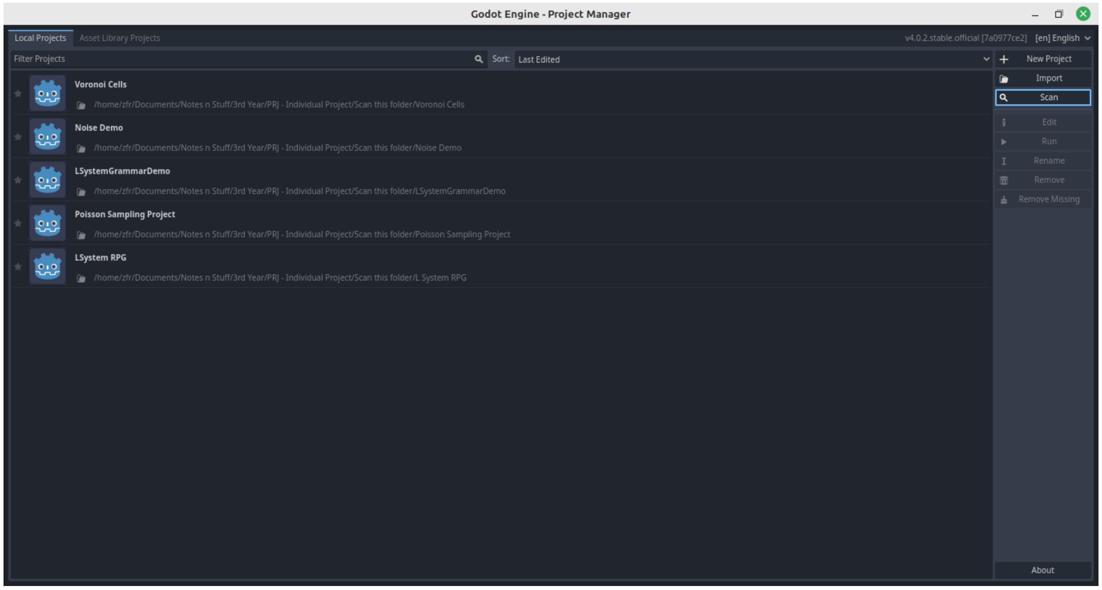

Figure B.3: Once some Godot projects have been imported into the project manager, you should be able to easily view the list and double-click on any one of the projects to edit them, which will open the editor after closing the project manager. You could also click the “Edit” button, or click “Run” to run the game without having to open the editor itself.

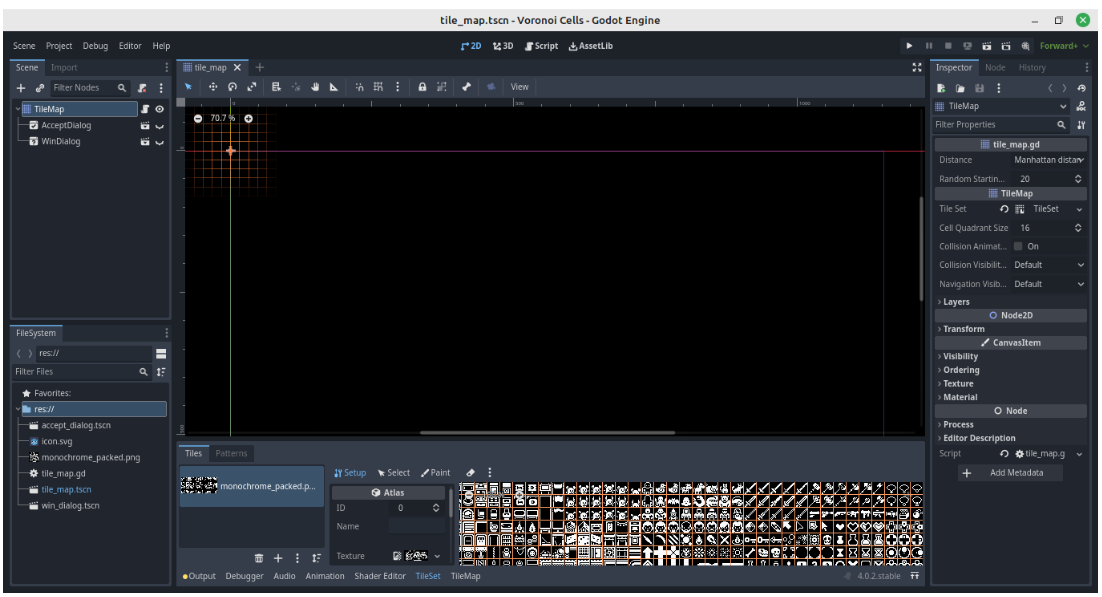

Figure B.4: The Godot editor open with the Voronoi cells project as an example. A visual description of the editor’s contents is in chapter B.2.

B.2 The Godot Editor

As you open up the Godot editor, you will see the main scene view in the center, as shown in Figure B.4, using the Voronoi cells implementation as an example. The left-hand side shows the scene tree at the top, and the file system (from the root folder of the project) at the bottom. Meanwhile, the right hand side shows the currently selected node's export variables, *including the custom export variables* defined in the node's script file, and two other tabs, “Node” (which shows a list of signals for the scene that could be called in a script) and “History” (which shows the sequence of recent actions performed on the scene during the current session). Above this is also a set of buttons which can be used for playing the project and/or the current scene. We go over how to run the current project in chapter B.4.

B.3 Custom Export Variables

When you click on some of the scenes in the projects, there may be some ”exported” variables from scripts that are visible to you in the editor (examples include the ”Distance” and ”Random Starting Points” variables in the Voronoi Cells project). You can hover over the variable names in the editor and it will show a brief description of what the variable correlates to in the script. We will now go over the different export variables across all of the provided artefacts in this section.

B.3.1 Lindenmayer System

All of the custom export variables defined in scene scripts for your use are in the child node “LSystem” (it is saved into its own scene file, but it is cheifly a child node of the root node “TileMap”). Open the “TileMap” scene (if it is not already opened for you when you launch the Godot editor) and click on the “LSystem” node in the scene tree to edit it.

1. axiom
    - **Type:** String
    - **Documentation:** The starting string from which the grammar starts applying its rules. Here it may be self defined, or randomly defined when “use_random_axiom” is true.
    - **Default value:** “OWB”
2. use_random_axiom
    - **Type:** Boolean (bool)
    - **Documentation:** Uses a random axiom with the currently set grammar, computed upon runtime, with a length up to (but not strictly) the value of upper_limit. For

example, if upper_limit is set to 15, the generated axiom can be 15 characters, or it can be just 5 characters.

- **Default value:** true

3. upper_limit

- **Type:** Integer (int)
- **Documentation:** Defines how many characters a random axiom can have MAXIMUM. Only used when "use_random_axiom" is true.
- **Default value:** 10

4. use_custom_ruleset

- **Type:** Boolean (bool)
- **Documentation:** Allows the use of a customly defined ruleset made through amending the rules array in the editor.
- **Default value:** false

5. ruleset

- **Type:** String enumeration of the following choices:
  (a) "Default"
  (b) "More Buildings (IMPOSSIBLE)"
  (c) "More Trees"
  (d) "More Space"
- **Documentation:** Denotes a series of pre-defined rulesets for this L-System grammar, of alphabet O (blank space), W (trees and fauna) and B (buildings), that can be chosen and then used on runtime. Can choose between a default ruleset, a ruleset that produces more buildings, a ruleset that produces more trees and a ruleset that produces more empty space.
- **Default value:** "Default"

6. rules

- **Type:** Array of dictionaries

- **Documentation:** The set of rules that the L-System grammar uses. Shows the “default” ruleset in the Godot editor for the user to see. If “use_custom_ruleset” is set to true, this array can be edited with a custom defined ruleset that will be used on runtime, so long as it adheres to the alphabet of O (blank space), W (trees and fauna) and B (buildings).
- **Additional information:** The “_get_ruleset” function uses the String value in “ruleset” to set the value for “rules” on runtime, *if* “use_custom_ruleset” is false (which it *is* by default).
- **Default value:** The “Default” grammar, as shown in Figure B.5.

```
[
  {
     "from": "O",
     "to": "OWO"
  },
  {
     "from": "W",
     "to": "WB"
  },
  {
     "from": "B",
     "to": "BWO"
  }
]
```

Figure B.5: The “Default” grammar used for the “rules” export variable, stored in the constant “DEFAULT” in l_system.gd. See l_system.gd for the other grammars represented as arrays of dictionaries.

B.3.2 Perlin/Simplex Noise

1. noise_type

   - **Type:** String enumeration of the following choices:

(a) “Perlin”

(b) “Simplex”

(c) “Simplex Smooth”

(d) “Value”

(e) “Value Cubic”

- **Documentation:** Defines the type of noise generation algorithm to use. Equates to the “noise_type” property in FastNoiseLite.
- **Default value:** “Value Cubic”

2. fractal_type

- **Type:** Enumeration of the following choices:

(a) “Fractal None”

(b) “Fractal FBM”

(c) “Fractal Ridged”

(d) “Fractal Ping Pong”

- **Documentation:** Defines the type of method used to combine octaves of a noise image into a fractal. Directly equates to the FractalType enumeration in FastNoiseLite.
- **Default value:** “Fractal None”

3. cellular_distance_type

- **Type:** Enumeration of the following choices:

(a) “Distance Euclidean”

(b) “Distance Euclidean Squared”

(c) “Distance Manhattan”

(d) “Distance Hybrid”

- **Documentation:** Defines the function used to calculate the distance between the nearest/second-nearest point(s). Directly equates to the CellularDistanceFunction enumeration in FastNoiseLite.
- **Default value:** “Distance Euclidean”

4. noise_frequency

- **Type:** Floating point number (float) between 0.0 and 1.0 inclusive

- **Documentation:** Defines the frequency of the generated noise, the higher the frequency, the rougher and more granular the noise.
- **Additional information:** The default value for “frequency” in the “FastNoiseClass” is 0.01, resulting in very smooth and distinct noise.
- **Default value:** 0.894

5. tree_cap

- **Type:** Floating point number (float) between -1.0 and 1.0 inclusive
- **Documentation:** Defines the upper limit to set for painting a tree tile on a specific noise pixel. If the value returned by the “get_noise_2d” method (in FastNoiseLite) is smaller than this, then it gets painted.
- **Default value:** -0.048

6. building_cap

- **Type:** Floating point number (float) between -1.0 and 1.0 inclusive
- **Documentation:** Defines the upper limit to set for painting a building tile on a specific noise pixel. If the value returned by the “get_noise_2d” method (in FastNoiseLite) is smaller than this, then it gets painted. If the value of “building_cap” is smaller than “tree_cap,” then decide whether or not to paint a building cell there with “building_overtakes_tree.”
- **Default value:** -0.252

7. building_overtakes_tree

- **Type:** Floating point number (float) between 0.0 and 0.5 inclusive
- **Documentation:** Only used when “building_cap” is smaller than “tree_cap.” Determines the probability that a building tile would be painted in a cell where a tree tile was, or could be, also painted. Whether or not the cell actually is painted over is decided on computation time.
- **Default value:** 0.12

B.3.3 Poisson Disk Sampling

1. paint_building_probability

- **Type:** Floating point number (float) between 0.0 and 1.0 inclusive
- **Documentation:** The probability that a building gets painted at a cell in lieu of a tree. The higher this probability, the more likely a building tile gets painted instead of a tree tile.
- **Default value:** 0.125

2. point_radius
   - **Type:** Floating point number (float) between 0.5 and 2.5 inclusive
   - **Documentation:** The radius value used to measure distances between points for the algorithm. The longer the radius, the further apart points are during the algorithm's processing, and the further apart painted cells are in the game.
   - **Default value:** 1.0

3. region_size
   - **Type:** Vector2
   - **Documentation:** The size of the region in which the algorithm is performed. Set to the ”default” tile map size (72, 40) in the script, shown as (0, 0) in the Godot editor. Can be changed to use a smaller region for the algorithm itself, of course resulting in less cells covered within the boundaries set for this game, though the algorithm will perform faster due to less cells being checked.
   - **Default value:**
     - **x**: The value in “x_tile_range” (72)
     - **y**: The value in “y_tile_range” (40)

4. rejection_samples
   - **Type:** Integer (int) between 1 andc 50 inclusive
   - **Documentation:** The maximum number of times a cell is checked before it is ignored. A cell can be accepted and painted on before this maximum number is reached. The higher this value, the more times a cell is checked, therefore the higher the algorithm's processing time.
   - **Default value:** 8

B.3.4 Voronoï Cells

1. distance

    - **Type:** String enumeration of the following choices:

        (a) "Euclidean distance"

        (b) "Manhattan distance"

    - **Documentation:** Determines whether or not the Euclidean or Manhattan distance formula is used for calculation of the deltas between points within Voronoi cells.

    - **Default value:** "Manhattan distance"

2. random_starting_points

    - **Type:** Integer (int) between 15 and 40 inclusive

    - **Documentation:** Determines the number of points randomly picked from at the start. Therefore, it also determines the number of cells in our Voronoi tesselation.

    - **Default value:** 20

B.3.5 The Basic L-System Demo Used to Create the Screenshots in Chapter 3.1.1

There is only one export variable for this: "choices", which allows you to choose which one of the three provided rulesets to use. "choices" is the default ruleset, and either "deterministic" or "basic" can be chosen. It is in the "DemoNode" scene, the only scene of this Godot project. Quoting the documentation comment, this export variable "Allows you to decide which ruleset to use. See the script file for the sources of said rulesets." The ruleset is assigned with the "set_values" function.

B.4 Running the Godot Projects

To *run* the current project in the Godot editor, go to the bar above the Inspector, Node and History tabs on the right-hand side. You will find a ▶ button which will play the main scene of the project (in the artefacts, the main scenes have already been set; if it were not already set, you would have been asked to set one). If closing the window to stop the project does not work, hit the ■ button to end it. If you want to replay the current project without having to stop it, hit the ▶ button again. Although **both** the icon **and** the colour of the ▶ button will have changed by then, it will be in the same position as before.

As described in section 3.2.1, only the close button of both the popup dialogs in the game seems to work properly for the moment, but this does not adversely affect the game functioning properly, nor does it adversely affect this project, so this is trivial.

Appendix C

Source Code

Contents

C.1 Instructions

If needed, use the table of contents provided to browse through the code listings in this section. Each listing folder will have a description, a link to its public GitHub repository, and a listing for each readable source file. Use the .zip folder containing the project artefacts to edit and run them in Godot.

"I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary." *- Zishan Rahman, 21st April 2023*

C.2 ProcGenRPG (L-System)

C.2.1 .gitattributes

```
# Normalize EOL for all files that Git considers text files.
* text=auto eol=lf
```

C.2.2 .gitignore

```
# Godot 4+ specific ignores
.godot/
```

C.2.3 project.godot

```
; Engine configuration file.
; It's best edited using the editor UI and not directly,
; since the parameters that go here are not all obvious.
;
; Format:
;   [section] ; section goes between []
;   param=value ; assign values to parameters

config_version=5

[application]

config/name="LSystem RPG"
run/main_scene="res://tile_map.tscn"
config/features=PackedStringArray("4.0", "Forward Plus")
config/icon="res://icon.svg"

```

```
[display]

window/size/viewport_height=640

[rendering]

environment/defaults/default_clear_color=Color(0, 0, 0, 1)
```

C.2.4 l_system.tscn

```
[gd_scene load_steps=2 format=3 uid="uid://d0v18e7ms571f"]

[ext_resource type="Script" path="res://l_system.gd" id="1_elydp"]

[node name="LSystem" type="Node"]
script = ExtResource("1_elydp")
```

C.2.5 l_system.gd

```
extends TileMap

@onready var l_system: LSystem = $LSystem

var x_tile_range: int = ProjectSettings.get_setting("display/window/size/viewport_width") / tile_set.tile_size.x
var y_tile_range: int = ProjectSettings.get_setting("display/window/size/viewport_height") / tile_set.tile_size.y

const PLAYER_SPRITE: Vector2i = Vector2i(24, 7)
var player_placement_cell: Vector2i
const rings: Array[Vector2i] = [
```

```
    Vector2i(43, 6),
    Vector2i(44, 6),
    Vector2i(45, 6),
    Vector2i(46, 6)
]
var ring_placement_cell: Vector2i

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
    randomize()
    var start_time: float = Time.get_ticks_msec()
    l_system.paint()
    place_player()
    place_ring()
    var new_time: float = Time.get_ticks_msec() - start_time
    print("Time taken: " + str(new_time) + "ms")
    $AcceptDialog.dialog_text = "You're a hollow Golem who seeks the ultimate treasure; a ring that's got something on top of it. It's somewhere in this large village and barely visible to your naked eyes, which took us " + str(new_time) + " milliseconds to generate (" + str(new_time / 1000.0) + " seconds), but you'll stop at nothing to get what you want. You can chow down every tree and fauna that stands in your way of the ring, but your Achilles heel is any bricks and mortar, which WILL make you stop at your tracks. Since it's easy to get lost in here, we'll tell you that you're in position " + str(player_placement_cell) + " in this big village of size " + str(Vector2i(x_tile_range, y_tile_range)) + ". The ring is in position " + str(ring_placement_cell) + ", but it is YOUR job to traverse the village, chow down the trees and get it for yourself, so are you ready to attain the treasure that is rightfully yours?!"
    $AcceptDialog.visible = true
```

```
        $AcceptDialog.confirmed.connect(_on_AcceptDialog_closed)
        $AcceptDialog.canceled.connect(_on_AcceptDialog_closed)
        $WinDialog.confirmed.connect(_on_WinDialog_confirmed)
        $WinDialog.canceled.connect(_on_WinDialog_canceled)
        get_tree().paused = true

func _on_WinDialog_confirmed() -> void:
    get_tree().reload_current_scene()

func _on_WinDialog_canceled() -> void:
    get_tree().quit()

func _on_AcceptDialog_closed() -> void:
    $AcceptDialog.visible = false
    get_tree().paused = false

func _get_random_placement_cell() -> Vector2i:
    return Vector2i(randi() % x_tile_range, randi() % y_tile_range)

func place_player() -> void:
    player_placement_cell = _get_random_placement_cell()
    while get_used_cells(0).has(player_placement_cell):
        player_placement_cell = _get_random_placement_cell()
    set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)

func place_ring() -> void:
    ring_placement_cell = _get_random_placement_cell()
    while get_used_cells(0).has(ring_placement_cell):
        ring_placement_cell = _get_random_placement_cell()
    set_cell(0, ring_placement_cell, 0, rings.pick_random())

func _is_not_out_of_bounds(cell: Vector2i) -> bool:
    return cell.x >= 0 and cell.x < x_tile_range and cell.y >= 0 and
```

```
        cell.y < y_tile_range

func _physics_process(_delta: float) -> void:
    var previous_cell: Vector2i = player_placement_cell
    var direction: Vector2i = Vector2i.ZERO
    if Input.is_action_pressed("ui_up"): direction = Vector2i.UP
    elif Input.is_action_pressed("ui_down"): direction = Vector2i.DOWN
    elif Input.is_action_pressed("ui_left"): direction = Vector2i.LEFT
    elif Input.is_action_pressed("ui_right"): direction = Vector2i.RIGHT
    var new_placement_cell: Vector2i = player_placement_cell + direction
    if (not get_used_cells(0).has(new_placement_cell) or l_system.trees.has(get_cell_atlas_coords(0, new_placement_cell)) or new_placement_cell == ring_placement_cell) and _is_not_out_of_bounds(new_placement_cell):
        player_placement_cell = new_placement_cell
        set_cell(0, previous_cell, 0) # deletes contents of previous cell (atlas_coords = Vector2i(-1, -1))
        set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)
        if player_placement_cell == ring_placement_cell:
            $WinDialog.visible = true
            get_tree().paused = true

# ALGORITHM AND CUSTOM EXPORT VARIABLES ARE IN LSYSTEM NODE
```

C.2.6 tile_map.tscn

```
[gd_scene load_steps=6 format=3 uid="uid://bwhvtqld3yo8m"]

```

```
[ext_resource type="TileSet" uid="uid://c168x78r0tful" path="res://Tiles.tres" id="1_l3nwg"]
[ext_resource type="Script" path="res://tile_map.gd" id="2_wrxl8"]
[ext_resource type="PackedScene" uid="uid://d0v18e7ms571f" path="res://l_system.tscn" id="3_ktw1n"]
[ext_resource type="PackedScene" uid="uid://cau5jgogdnf53" path="res://accept_dialog.tscn" id="4_060oh"]
[ext_resource type="PackedScene" uid="uid://b5q8ovcigrvyr" path="res://win_dialog.tscn" id="5_3s48a"]

[node name="TileMap" type="TileMap"]
tile_set = ExtResource("1_l3nwg")
format = 2
layer_0/name = "Things"
script = ExtResource("2_wrxl8")

[node name="LSystem" parent="." instance=ExtResource("3_ktw1n")]

[node name="AcceptDialog" parent="." instance=ExtResource("4_060oh")]

[node name="WinDialog" parent="." instance=ExtResource("5_3s48a")]
```

C.2.7 tile_map.gd

```
extends TileMap

@onready var l_system: LSystem = $LSystem

var x_tile_range: int = ProjectSettings.get_setting("display/window/size/viewport_width") / tile_set.tile_size.x
var y_tile_range: int = ProjectSettings.get_setting("display/window
```

```
/size/viewport_height") / tile_set.tile_size.y

const PLAYER_SPRITE: Vector2i = Vector2i(24, 7)
var player_placement_cell: Vector2i
const rings: Array[Vector2i] = [
	Vector2i(43, 6),
	Vector2i(44, 6),
	Vector2i(45, 6),
	Vector2i(46, 6)
]
var ring_placement_cell: Vector2i

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
	randomize()
	var start_time: float = Time.get_ticks_msec()
	l_system.paint()
	place_player()
	place_ring()
	var new_time: float = Time.get_ticks_msec() - start_time
	print("Time taken: " + str(new_time) + "ms")
	$AcceptDialog.dialog_text = "You're a hollow Golem who seeks the ultimate treasure; a ring that's got something on top of it. It's somewhere in this large village and barely visible to your naked eyes, which took us " + str(new_time) + " milliseconds to generate (" + str(new_time / 1000.0) + " seconds), but you'll stop at nothing to get what you want. You can chow down every tree and fauna that stands in your way of the ring, but your Achilles heel is any bricks and mortar, which WILL make you stop at your tracks. Since it's easy to get lost in here, we'll tell you that you're in position " + str(player_placement_cell) + " in this big village of size " + str(Vector2i(x_tile_range, y_tile_range)
```

```
	) + ". The ring is in position " + str(ring_placement_cell)
	+ ", but it is YOUR job to traverse the village, chow down
	the trees and get it for yourself, so are you ready to
	attain the treasure that is rightfully yours?!"
	$AcceptDialog.visible = true
	$AcceptDialog.confirmed.connect(_on_AcceptDialog_closed)
	$AcceptDialog.canceled.connect(_on_AcceptDialog_closed)
	$WinDialog.confirmed.connect(_on_WinDialog_confirmed)
	$WinDialog.canceled.connect(_on_WinDialog_canceled)
	get_tree().paused = true

func _on_WinDialog_confirmed() -> void:
	get_tree().reload_current_scene()

func _on_WinDialog_canceled() -> void:
	get_tree().quit()

func _on_AcceptDialog_closed() -> void:
	$AcceptDialog.visible = false
	get_tree().paused = false

func _get_random_placement_cell() -> Vector2i:
	return Vector2i(randi() % x_tile_range, randi() % y_tile_range)

func place_player() -> void:
	player_placement_cell = _get_random_placement_cell()
	while get_used_cells(0).has(player_placement_cell):
		player_placement_cell = _get_random_placement_cell()
	set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)

func place_ring() -> void:
	ring_placement_cell = _get_random_placement_cell()
	while get_used_cells(0).has(ring_placement_cell):
```

```
		ring_placement_cell = _get_random_placement_cell()
	set_cell(0, ring_placement_cell, 0, rings.pick_random())

func _is_not_out_of_bounds(cell: Vector2i) -> bool:
	return cell.x >= 0 and cell.x < x_tile_range and cell.y >= 0 and
		cell.y < y_tile_range

func _physics_process(_delta: float) -> void:
	var previous_cell: Vector2i = player_placement_cell
	var direction: Vector2i = Vector2i.ZERO
	if Input.is_action_pressed("ui_up"): direction = Vector2i.UP
	elif Input.is_action_pressed("ui_down"): direction = Vector2i.
		DOWN
	elif Input.is_action_pressed("ui_left"): direction = Vector2i.
		LEFT
	elif Input.is_action_pressed("ui_right"): direction = Vector2i.
		RIGHT
	var new_placement_cell: Vector2i = player_placement_cell +
		direction
	if (not get_used_cells(0).has(new_placement_cell) or l_system.
		trees.has(get_cell_atlas_coords(0, new_placement_cell)) or
		new_placement_cell == ring_placement_cell) and
		_is_not_out_of_bounds(new_placement_cell):
		player_placement_cell = new_placement_cell
		set_cell(0, previous_cell, 0) # deletes contents of previous
			cell (atlas_coords = Vector2i(-1, -1))
		set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)
		if player_placement_cell == ring_placement_cell:
			$WinDialog.visible = true
			get_tree().paused = true

# ALGORITHM AND CUSTOM EXPORT VARIABLES ARE IN LSYSTEM NODE
```

C.2.8 accept_dialog.tscn

```
[gd_scene format=3 uid="uid://cau5jgogdnf53"]

[node name="AcceptDialog" type="AcceptDialog"]
title = "Tree-Munching Time!"
position = Vector2i(326, 100)
size = Vector2i(500, 421)
mouse_passthrough = true
ok_button_text = "Bring it on!"
dialog_text = "You're a hollow Golem who seeks the ultimate treasure; a ring that's got something on top of it. It's somewhere in this large village and barely visible to your naked eyes, but you'll stop at nothing to get what you want. You can chow down every tree and fauna that stands in your way of the ring, but your Achilles heel is any bricks and mortar, which will make you stop at your tracks. Are you ready to attain your treasure?w Golem in a black-and-white world, in search for your most desired treasure. It's a ring with something on top of it. And you'll stop at nothing to get what you want. You can chow down every tree and fauna that stands in your way of the ring, but your Achilles heel is any bricks and mortar, which will make you stop at your tracks. Are you ready to attain the treasure that is rightfully yours?!"
dialog_autowrap = true
```

C.2.9 win_dialog.tscn

```
[gd_scene format=3 uid="uid://b5q8ovcigrvyr"]

[node name="WinDialog" type="ConfirmationDialog"]
title = "You Found the Treasure!"
```

```
position = Vector2i(326, 100)
size = Vector2i(500, 421)
mouse_passthrough = true
ok_button_text = "Get Me a New Village"
dialog_text = "You found your treasure! Well done, you!

Would you like to travel to a new village in the hopes of finding
    another ring? Or would you like to take your treasure home now?
    "
dialog_autowrap = true
cancel_button_text = "Get Me Out of Here"
```

C.2.10 icon.svg.import

```
[remap]

importer="texture"
type="CompressedTexture2D"
uid="uid://b45qexb3wmhym"
path="res://.godot/imported/icon.svg-218
    a8f2b3041327d8a5756f3a245f83b.ctex"
metadata={
"vram_texture": false
}

[deps]

source_file="res://icon.svg"
dest_files=["res://.godot/imported/icon.svg-218
    a8f2b3041327d8a5756f3a245f83b.ctex"]

[params]
```

```

compress/mode=0
compress/high_quality=false
compress/lossy_quality=0.7
compress/hdr_compression=1
compress/normal_map=0
compress/channel_pack=0
mipmaps/generate=false
mipmaps/limit=-1
roughness/mode=0
roughness/src_normal=""
process/fix_alpha_border=true
process/premult_alpha=false
process/normal_map_invert_y=false
process/hdr_as_srgb=false
process/hdr_clamp_exposure=false
process/size_limit=0
detect_3d/compress_to=1
svg/scale=1.0
editor/scale_with_editor_scale=false
editor/convert_colors_with_editor_theme=false
```

C.2.11 roguelikeSheet_transparent.png.import

```
[remap]

importer="texture"
type="CompressedTexture2D"
uid="uid://13ktp0qup5xb"
path="res://.godot/imported/roguelikeSheet_transparent.png-22
    f6b70da04549e371d1f15fe9d96005.ctex"
metadata={
```

```
"vram_texture": false
}

[deps]

source_file="res://roguelikeSheet_transparent.png"
dest_files=["res://.godot/imported/roguelikeSheet_transparent.png
    -22f6b70da04549e371d1f15fe9d96005.ctex"]

[params]

compress/mode=0
compress/high_quality=false
compress/lossy_quality=0.7
compress/hdr_compression=1
compress/normal_map=0
compress/channel_pack=0
mipmaps/generate=false
mipmaps/limit=-1
roughness/mode=0
roughness/src_normal=""
process/fix_alpha_border=true
process/premult_alpha=false
process/normal_map_invert_y=false
process/hdr_as_srgb=false
process/hdr_clamp_exposure=false
process/size_limit=0
detect_3d/compress_to=1
```

C.3 VoronoiCellsGD4

C.3.1 .gitattributes

```
# Normalize EOL for all files that Git considers text files.
* text=auto eol=lf
```

C.3.2 .gitignore

```
# Godot 4+ specific ignores
.godot/
```

C.3.3 project.godot

```
; Engine configuration file.
; It's best edited using the editor UI and not directly,
; since the parameters that go here are not all obvious.
;
; Format:
;   [section] ; section goes between []
;   param=value ; assign values to parameters

config_version=5

[application]

config/name="Voronoi Cells"
run/main_scene="res://tile_map.tscn"
config/features=PackedStringArray("4.0", "Forward Plus")
config/icon="res://icon.svg"

[display]

window/size/viewport_height=640

```

```
[input]

reset_position={
"deadzone": 0.5,
"events": [Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":-1,"window_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_pressed":false,"pressed":false,"keycode":71,"physical_keycode":0,"key_label":0,"unicode":103,"echo":false,"script":null)
, Object(InputEventMouseButton,"resource_local_to_scene":false,"resource_name":"","device":-1,"window_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_pressed":false,"button_mask":2,"position":Vector2(75, 12),"global_position":Vector2(78, 44),"factor":1.0,"button_index":2,"pressed":true,"double_click":false,"script":null)
]
}

[rendering]

environment/defaults/default_clear_color=Color(0, 0, 0, 1)
```

C.3.4 tile_map.tscn

```
[gd_scene load_steps=7 format=3 uid="uid://d6lxnr5bdh1w"]

[ext_resource type="Texture2D" uid="uid://cpign73sfbsrt" path="res://monochrome_packed.png" id="1_o183d"]
[ext_resource type="Script" path="res://tile_map.gd" id="2_lf4lw"]
[ext_resource type="PackedScene" path="res://accept_dialog.tscn" id="3_y081j"]
[ext_resource type="PackedScene" path="res://win_dialog.tscn" id="4
```

```
    _fkys0"]

[sub_resource type="TileSetAtlasSource" id="
    TileSetAtlasSource_6h0bd"]
texture = ExtResource("1_o183d")
0:0/0 = 0
1:0/0 = 0
2:0/0 = 0
3:0/0 = 0
4:0/0 = 0
5:0/0 = 0
6:0/0 = 0
7:0/0 = 0
8:0/0 = 0
9:0/0 = 0
10:0/0 = 0
11:0/0 = 0
12:0/0 = 0
13:0/0 = 0
14:0/0 = 0
15:0/0 = 0
16:0/0 = 0
17:0/0 = 0
18:0/0 = 0
19:0/0 = 0
20:0/0 = 0
21:0/0 = 0
22:0/0 = 0
23:0/0 = 0
24:0/0 = 0
25:0/0 = 0
26:0/0 = 0
27:0/0 = 0
```

```
28:0/0 = 0
29:0/0 = 0
30:0/0 = 0
31:0/0 = 0
32:0/0 = 0
33:0/0 = 0
34:0/0 = 0
35:0/0 = 0
36:0/0 = 0
37:0/0 = 0
38:0/0 = 0
39:0/0 = 0
40:0/0 = 0
41:0/0 = 0
42:0/0 = 0
43:0/0 = 0
44:0/0 = 0
45:0/0 = 0
46:0/0 = 0
47:0/0 = 0
48:0/0 = 0
0:1/0 = 0
1:1/0 = 0
2:1/0 = 0
3:1/0 = 0
4:1/0 = 0
5:1/0 = 0
6:1/0 = 0
7:1/0 = 0
8:1/0 = 0
9:1/0 = 0
10:1/0 = 0
11:1/0 = 0
```

```
12:1/0 = 0
13:1/0 = 0
14:1/0 = 0
15:1/0 = 0
16:1/0 = 0
17:1/0 = 0
18:1/0 = 0
19:1/0 = 0
20:1/0 = 0
21:1/0 = 0
22:1/0 = 0
23:1/0 = 0
24:1/0 = 0
25:1/0 = 0
26:1/0 = 0
27:1/0 = 0
28:1/0 = 0
29:1/0 = 0
30:1/0 = 0
31:1/0 = 0
32:1/0 = 0
33:1/0 = 0
34:1/0 = 0
35:1/0 = 0
36:1/0 = 0
37:1/0 = 0
38:1/0 = 0
39:1/0 = 0
40:1/0 = 0
41:1/0 = 0
42:1/0 = 0
43:1/0 = 0
44:1/0 = 0
```

```
45:1/0 = 0
46:1/0 = 0
47:1/0 = 0
48:1/0 = 0
0:2/0 = 0
1:2/0 = 0
2:2/0 = 0
3:2/0 = 0
4:2/0 = 0
5:2/0 = 0
6:2/0 = 0
7:2/0 = 0
8:2/0 = 0
9:2/0 = 0
10:2/0 = 0
11:2/0 = 0
12:2/0 = 0
13:2/0 = 0
14:2/0 = 0
15:2/0 = 0
16:2/0 = 0
17:2/0 = 0
18:2/0 = 0
19:2/0 = 0
20:2/0 = 0
21:2/0 = 0
22:2/0 = 0
23:2/0 = 0
24:2/0 = 0
25:2/0 = 0
26:2/0 = 0
27:2/0 = 0
28:2/0 = 0
```

```
29:2/0 = 0
30:2/0 = 0
31:2/0 = 0
32:2/0 = 0
33:2/0 = 0
34:2/0 = 0
35:2/0 = 0
36:2/0 = 0
37:2/0 = 0
38:2/0 = 0
39:2/0 = 0
40:2/0 = 0
41:2/0 = 0
42:2/0 = 0
43:2/0 = 0
44:2/0 = 0
45:2/0 = 0
46:2/0 = 0
47:2/0 = 0
48:2/0 = 0
0:3/0 = 0
1:3/0 = 0
2:3/0 = 0
3:3/0 = 0
4:3/0 = 0
5:3/0 = 0
6:3/0 = 0
7:3/0 = 0
8:3/0 = 0
9:3/0 = 0
10:3/0 = 0
11:3/0 = 0
12:3/0 = 0
```

```
13:3/0 = 0
14:3/0 = 0
15:3/0 = 0
16:3/0 = 0
17:3/0 = 0
18:3/0 = 0
19:3/0 = 0
20:3/0 = 0
21:3/0 = 0
22:3/0 = 0
23:3/0 = 0
24:3/0 = 0
25:3/0 = 0
26:3/0 = 0
27:3/0 = 0
28:3/0 = 0
29:3/0 = 0
30:3/0 = 0
31:3/0 = 0
32:3/0 = 0
33:3/0 = 0
34:3/0 = 0
35:3/0 = 0
36:3/0 = 0
37:3/0 = 0
38:3/0 = 0
39:3/0 = 0
40:3/0 = 0
41:3/0 = 0
42:3/0 = 0
43:3/0 = 0
44:3/0 = 0
45:3/0 = 0
```

```
46:3/0 = 0
47:3/0 = 0
48:3/0 = 0
0:4/0 = 0
1:4/0 = 0
2:4/0 = 0
3:4/0 = 0
4:4/0 = 0
5:4/0 = 0
6:4/0 = 0
7:4/0 = 0
8:4/0 = 0
9:4/0 = 0
10:4/0 = 0
11:4/0 = 0
12:4/0 = 0
13:4/0 = 0
14:4/0 = 0
15:4/0 = 0
16:4/0 = 0
17:4/0 = 0
18:4/0 = 0
19:4/0 = 0
20:4/0 = 0
21:4/0 = 0
22:4/0 = 0
23:4/0 = 0
24:4/0 = 0
25:4/0 = 0
26:4/0 = 0
27:4/0 = 0
28:4/0 = 0
29:4/0 = 0
```

```
30:4/0 = 0
31:4/0 = 0
32:4/0 = 0
33:4/0 = 0
34:4/0 = 0
35:4/0 = 0
36:4/0 = 0
37:4/0 = 0
38:4/0 = 0
39:4/0 = 0
40:4/0 = 0
41:4/0 = 0
42:4/0 = 0
43:4/0 = 0
44:4/0 = 0
45:4/0 = 0
46:4/0 = 0
47:4/0 = 0
48:4/0 = 0
0:5/0 = 0
1:5/0 = 0
2:5/0 = 0
3:5/0 = 0
4:5/0 = 0
5:5/0 = 0
6:5/0 = 0
7:5/0 = 0
8:5/0 = 0
9:5/0 = 0
10:5/0 = 0
11:5/0 = 0
12:5/0 = 0
13:5/0 = 0
```

```
14:5/0 = 0
15:5/0 = 0
16:5/0 = 0
17:5/0 = 0
18:5/0 = 0
19:5/0 = 0
20:5/0 = 0
21:5/0 = 0
22:5/0 = 0
23:5/0 = 0
24:5/0 = 0
25:5/0 = 0
26:5/0 = 0
27:5/0 = 0
28:5/0 = 0
29:5/0 = 0
30:5/0 = 0
31:5/0 = 0
32:5/0 = 0
33:5/0 = 0
34:5/0 = 0
35:5/0 = 0
36:5/0 = 0
37:5/0 = 0
38:5/0 = 0
39:5/0 = 0
40:5/0 = 0
41:5/0 = 0
42:5/0 = 0
43:5/0 = 0
44:5/0 = 0
45:5/0 = 0
46:5/0 = 0
```

```
47:5/0 = 0
48:5/0 = 0
0:6/0 = 0
1:6/0 = 0
2:6/0 = 0
3:6/0 = 0
4:6/0 = 0
5:6/0 = 0
6:6/0 = 0
7:6/0 = 0
8:6/0 = 0
9:6/0 = 0
10:6/0 = 0
11:6/0 = 0
12:6/0 = 0
13:6/0 = 0
14:6/0 = 0
15:6/0 = 0
16:6/0 = 0
17:6/0 = 0
18:6/0 = 0
19:6/0 = 0
20:6/0 = 0
21:6/0 = 0
22:6/0 = 0
23:6/0 = 0
24:6/0 = 0
25:6/0 = 0
26:6/0 = 0
27:6/0 = 0
28:6/0 = 0
29:6/0 = 0
30:6/0 = 0
```

```
31:6/0 = 0
32:6/0 = 0
33:6/0 = 0
34:6/0 = 0
35:6/0 = 0
36:6/0 = 0
37:6/0 = 0
38:6/0 = 0
39:6/0 = 0
40:6/0 = 0
41:6/0 = 0
42:6/0 = 0
43:6/0 = 0
44:6/0 = 0
45:6/0 = 0
46:6/0 = 0
47:6/0 = 0
48:6/0 = 0
0:7/0 = 0
1:7/0 = 0
2:7/0 = 0
3:7/0 = 0
4:7/0 = 0
5:7/0 = 0
6:7/0 = 0
7:7/0 = 0
8:7/0 = 0
9:7/0 = 0
10:7/0 = 0
11:7/0 = 0
12:7/0 = 0
13:7/0 = 0
14:7/0 = 0
```

```
15:7/0 = 0
16:7/0 = 0
17:7/0 = 0
18:7/0 = 0
19:7/0 = 0
20:7/0 = 0
21:7/0 = 0
22:7/0 = 0
23:7/0 = 0
24:7/0 = 0
25:7/0 = 0
26:7/0 = 0
27:7/0 = 0
28:7/0 = 0
29:7/0 = 0
30:7/0 = 0
31:7/0 = 0
32:7/0 = 0
33:7/0 = 0
34:7/0 = 0
35:7/0 = 0
36:7/0 = 0
37:7/0 = 0
38:7/0 = 0
39:7/0 = 0
40:7/0 = 0
41:7/0 = 0
42:7/0 = 0
43:7/0 = 0
44:7/0 = 0
45:7/0 = 0
46:7/0 = 0
47:7/0 = 0
```

```
48:7/0 = 0
0:8/0 = 0
1:8/0 = 0
2:8/0 = 0
3:8/0 = 0
4:8/0 = 0
5:8/0 = 0
6:8/0 = 0
7:8/0 = 0
8:8/0 = 0
9:8/0 = 0
10:8/0 = 0
11:8/0 = 0
12:8/0 = 0
13:8/0 = 0
14:8/0 = 0
15:8/0 = 0
16:8/0 = 0
17:8/0 = 0
18:8/0 = 0
19:8/0 = 0
20:8/0 = 0
21:8/0 = 0
22:8/0 = 0
23:8/0 = 0
24:8/0 = 0
25:8/0 = 0
26:8/0 = 0
27:8/0 = 0
28:8/0 = 0
29:8/0 = 0
30:8/0 = 0
31:8/0 = 0
```

```
32:8/0 = 0
33:8/0 = 0
34:8/0 = 0
35:8/0 = 0
36:8/0 = 0
37:8/0 = 0
38:8/0 = 0
39:8/0 = 0
40:8/0 = 0
41:8/0 = 0
42:8/0 = 0
43:8/0 = 0
44:8/0 = 0
45:8/0 = 0
46:8/0 = 0
47:8/0 = 0
48:8/0 = 0
0:9/0 = 0
1:9/0 = 0
2:9/0 = 0
3:9/0 = 0
4:9/0 = 0
5:9/0 = 0
6:9/0 = 0
7:9/0 = 0
8:9/0 = 0
9:9/0 = 0
10:9/0 = 0
11:9/0 = 0
12:9/0 = 0
13:9/0 = 0
14:9/0 = 0
15:9/0 = 0
```

```
16:9/0 = 0
17:9/0 = 0
18:9/0 = 0
19:9/0 = 0
20:9/0 = 0
21:9/0 = 0
22:9/0 = 0
23:9/0 = 0
24:9/0 = 0
25:9/0 = 0
26:9/0 = 0
27:9/0 = 0
28:9/0 = 0
29:9/0 = 0
30:9/0 = 0
31:9/0 = 0
32:9/0 = 0
33:9/0 = 0
34:9/0 = 0
35:9/0 = 0
36:9/0 = 0
37:9/0 = 0
38:9/0 = 0
39:9/0 = 0
40:9/0 = 0
41:9/0 = 0
42:9/0 = 0
43:9/0 = 0
44:9/0 = 0
45:9/0 = 0
46:9/0 = 0
47:9/0 = 0
48:9/0 = 0
```

```
0:10/0 = 0
1:10/0 = 0
2:10/0 = 0
3:10/0 = 0
4:10/0 = 0
5:10/0 = 0
6:10/0 = 0
7:10/0 = 0
8:10/0 = 0
9:10/0 = 0
10:10/0 = 0
11:10/0 = 0
12:10/0 = 0
13:10/0 = 0
14:10/0 = 0
15:10/0 = 0
16:10/0 = 0
17:10/0 = 0
18:10/0 = 0
19:10/0 = 0
20:10/0 = 0
21:10/0 = 0
22:10/0 = 0
23:10/0 = 0
24:10/0 = 0
25:10/0 = 0
26:10/0 = 0
27:10/0 = 0
28:10/0 = 0
29:10/0 = 0
30:10/0 = 0
31:10/0 = 0
32:10/0 = 0
```

```
33:10/0 = 0
34:10/0 = 0
35:10/0 = 0
36:10/0 = 0
37:10/0 = 0
38:10/0 = 0
39:10/0 = 0
40:10/0 = 0
41:10/0 = 0
42:10/0 = 0
43:10/0 = 0
44:10/0 = 0
45:10/0 = 0
46:10/0 = 0
47:10/0 = 0
48:10/0 = 0
0:11/0 = 0
1:11/0 = 0
2:11/0 = 0
3:11/0 = 0
4:11/0 = 0
5:11/0 = 0
6:11/0 = 0
7:11/0 = 0
8:11/0 = 0
9:11/0 = 0
10:11/0 = 0
11:11/0 = 0
12:11/0 = 0
13:11/0 = 0
14:11/0 = 0
15:11/0 = 0
16:11/0 = 0
```

```
17:11/0 = 0
18:11/0 = 0
19:11/0 = 0
20:11/0 = 0
21:11/0 = 0
22:11/0 = 0
23:11/0 = 0
24:11/0 = 0
25:11/0 = 0
26:11/0 = 0
27:11/0 = 0
28:11/0 = 0
29:11/0 = 0
30:11/0 = 0
31:11/0 = 0
32:11/0 = 0
33:11/0 = 0
34:11/0 = 0
35:11/0 = 0
36:11/0 = 0
37:11/0 = 0
38:11/0 = 0
39:11/0 = 0
40:11/0 = 0
41:11/0 = 0
42:11/0 = 0
43:11/0 = 0
44:11/0 = 0
45:11/0 = 0
46:11/0 = 0
47:11/0 = 0
48:11/0 = 0
0:12/0 = 0
```

```
1:12/0 = 0
2:12/0 = 0
3:12/0 = 0
4:12/0 = 0
5:12/0 = 0
6:12/0 = 0
7:12/0 = 0
8:12/0 = 0
9:12/0 = 0
10:12/0 = 0
11:12/0 = 0
12:12/0 = 0
13:12/0 = 0
14:12/0 = 0
15:12/0 = 0
16:12/0 = 0
17:12/0 = 0
18:12/0 = 0
19:12/0 = 0
20:12/0 = 0
21:12/0 = 0
22:12/0 = 0
23:12/0 = 0
24:12/0 = 0
25:12/0 = 0
26:12/0 = 0
27:12/0 = 0
28:12/0 = 0
29:12/0 = 0
30:12/0 = 0
31:12/0 = 0
32:12/0 = 0
33:12/0 = 0
```

```
34:12/0 = 0
35:12/0 = 0
36:12/0 = 0
37:12/0 = 0
38:12/0 = 0
39:12/0 = 0
40:12/0 = 0
41:12/0 = 0
42:12/0 = 0
43:12/0 = 0
44:12/0 = 0
45:12/0 = 0
46:12/0 = 0
47:12/0 = 0
48:12/0 = 0
0:13/0 = 0
1:13/0 = 0
2:13/0 = 0
3:13/0 = 0
4:13/0 = 0
5:13/0 = 0
6:13/0 = 0
7:13/0 = 0
8:13/0 = 0
9:13/0 = 0
10:13/0 = 0
11:13/0 = 0
12:13/0 = 0
13:13/0 = 0
14:13/0 = 0
15:13/0 = 0
16:13/0 = 0
17:13/0 = 0
```

```
18:13/0 = 0
19:13/0 = 0
20:13/0 = 0
21:13/0 = 0
22:13/0 = 0
23:13/0 = 0
24:13/0 = 0
25:13/0 = 0
26:13/0 = 0
27:13/0 = 0
28:13/0 = 0
29:13/0 = 0
30:13/0 = 0
31:13/0 = 0
32:13/0 = 0
33:13/0 = 0
34:13/0 = 0
35:13/0 = 0
36:13/0 = 0
37:13/0 = 0
38:13/0 = 0
39:13/0 = 0
40:13/0 = 0
41:13/0 = 0
42:13/0 = 0
43:13/0 = 0
44:13/0 = 0
45:13/0 = 0
46:13/0 = 0
47:13/0 = 0
48:13/0 = 0
0:14/0 = 0
1:14/0 = 0
```

```
2:14/0 = 0
3:14/0 = 0
4:14/0 = 0
5:14/0 = 0
6:14/0 = 0
7:14/0 = 0
8:14/0 = 0
9:14/0 = 0
10:14/0 = 0
11:14/0 = 0
12:14/0 = 0
13:14/0 = 0
14:14/0 = 0
15:14/0 = 0
16:14/0 = 0
17:14/0 = 0
18:14/0 = 0
19:14/0 = 0
20:14/0 = 0
21:14/0 = 0
22:14/0 = 0
23:14/0 = 0
24:14/0 = 0
25:14/0 = 0
26:14/0 = 0
27:14/0 = 0
28:14/0 = 0
29:14/0 = 0
30:14/0 = 0
31:14/0 = 0
32:14/0 = 0
33:14/0 = 0
34:14/0 = 0
```

```
35:14/0 = 0
36:14/0 = 0
37:14/0 = 0
38:14/0 = 0
39:14/0 = 0
40:14/0 = 0
41:14/0 = 0
42:14/0 = 0
43:14/0 = 0
44:14/0 = 0
45:14/0 = 0
46:14/0 = 0
47:14/0 = 0
48:14/0 = 0
0:15/0 = 0
1:15/0 = 0
2:15/0 = 0
3:15/0 = 0
4:15/0 = 0
5:15/0 = 0
6:15/0 = 0
7:15/0 = 0
8:15/0 = 0
9:15/0 = 0
10:15/0 = 0
11:15/0 = 0
12:15/0 = 0
13:15/0 = 0
14:15/0 = 0
15:15/0 = 0
16:15/0 = 0
17:15/0 = 0
18:15/0 = 0
```

```
19:15/0 = 0
20:15/0 = 0
21:15/0 = 0
22:15/0 = 0
23:15/0 = 0
24:15/0 = 0
25:15/0 = 0
26:15/0 = 0
27:15/0 = 0
28:15/0 = 0
29:15/0 = 0
30:15/0 = 0
31:15/0 = 0
32:15/0 = 0
33:15/0 = 0
34:15/0 = 0
35:15/0 = 0
36:15/0 = 0
37:15/0 = 0
38:15/0 = 0
39:15/0 = 0
40:15/0 = 0
41:15/0 = 0
42:15/0 = 0
43:15/0 = 0
44:15/0 = 0
45:15/0 = 0
46:15/0 = 0
47:15/0 = 0
48:15/0 = 0
0:16/0 = 0
1:16/0 = 0
2:16/0 = 0
```

```
3:16/0 = 0
4:16/0 = 0
5:16/0 = 0
6:16/0 = 0
7:16/0 = 0
8:16/0 = 0
9:16/0 = 0
10:16/0 = 0
11:16/0 = 0
12:16/0 = 0
13:16/0 = 0
14:16/0 = 0
15:16/0 = 0
16:16/0 = 0
17:16/0 = 0
18:16/0 = 0
19:16/0 = 0
20:16/0 = 0
21:16/0 = 0
22:16/0 = 0
23:16/0 = 0
24:16/0 = 0
25:16/0 = 0
26:16/0 = 0
27:16/0 = 0
28:16/0 = 0
29:16/0 = 0
30:16/0 = 0
31:16/0 = 0
32:16/0 = 0
33:16/0 = 0
34:16/0 = 0
35:16/0 = 0
```

```
36:16/0 = 0
37:16/0 = 0
38:16/0 = 0
39:16/0 = 0
40:16/0 = 0
41:16/0 = 0
42:16/0 = 0
43:16/0 = 0
44:16/0 = 0
45:16/0 = 0
46:16/0 = 0
47:16/0 = 0
48:16/0 = 0
0:17/0 = 0
1:17/0 = 0
2:17/0 = 0
3:17/0 = 0
4:17/0 = 0
5:17/0 = 0
6:17/0 = 0
7:17/0 = 0
8:17/0 = 0
9:17/0 = 0
10:17/0 = 0
11:17/0 = 0
12:17/0 = 0
13:17/0 = 0
14:17/0 = 0
15:17/0 = 0
16:17/0 = 0
17:17/0 = 0
18:17/0 = 0
19:17/0 = 0
```

```
20:17/0 = 0
21:17/0 = 0
22:17/0 = 0
23:17/0 = 0
24:17/0 = 0
25:17/0 = 0
26:17/0 = 0
27:17/0 = 0
28:17/0 = 0
29:17/0 = 0
30:17/0 = 0
31:17/0 = 0
32:17/0 = 0
33:17/0 = 0
34:17/0 = 0
35:17/0 = 0
36:17/0 = 0
37:17/0 = 0
38:17/0 = 0
39:17/0 = 0
40:17/0 = 0
41:17/0 = 0
42:17/0 = 0
43:17/0 = 0
44:17/0 = 0
45:17/0 = 0
46:17/0 = 0
47:17/0 = 0
48:17/0 = 0
0:18/0 = 0
1:18/0 = 0
2:18/0 = 0
3:18/0 = 0
```

```
4:18/0 = 0
5:18/0 = 0
6:18/0 = 0
7:18/0 = 0
8:18/0 = 0
9:18/0 = 0
10:18/0 = 0
11:18/0 = 0
12:18/0 = 0
13:18/0 = 0
14:18/0 = 0
15:18/0 = 0
16:18/0 = 0
17:18/0 = 0
18:18/0 = 0
19:18/0 = 0
20:18/0 = 0
21:18/0 = 0
22:18/0 = 0
23:18/0 = 0
24:18/0 = 0
25:18/0 = 0
26:18/0 = 0
27:18/0 = 0
28:18/0 = 0
29:18/0 = 0
30:18/0 = 0
31:18/0 = 0
32:18/0 = 0
33:18/0 = 0
34:18/0 = 0
35:18/0 = 0
36:18/0 = 0
```

```
37:18/0 = 0
38:18/0 = 0
39:18/0 = 0
40:18/0 = 0
41:18/0 = 0
42:18/0 = 0
43:18/0 = 0
44:18/0 = 0
45:18/0 = 0
46:18/0 = 0
47:18/0 = 0
48:18/0 = 0
0:19/0 = 0
1:19/0 = 0
2:19/0 = 0
3:19/0 = 0
4:19/0 = 0
5:19/0 = 0
6:19/0 = 0
7:19/0 = 0
8:19/0 = 0
9:19/0 = 0
10:19/0 = 0
11:19/0 = 0
12:19/0 = 0
13:19/0 = 0
14:19/0 = 0
15:19/0 = 0
16:19/0 = 0
17:19/0 = 0
18:19/0 = 0
19:19/0 = 0
20:19/0 = 0
```

```
21:19/0 = 0
22:19/0 = 0
23:19/0 = 0
24:19/0 = 0
25:19/0 = 0
26:19/0 = 0
27:19/0 = 0
28:19/0 = 0
29:19/0 = 0
30:19/0 = 0
31:19/0 = 0
32:19/0 = 0
33:19/0 = 0
34:19/0 = 0
35:19/0 = 0
36:19/0 = 0
37:19/0 = 0
38:19/0 = 0
39:19/0 = 0
40:19/0 = 0
41:19/0 = 0
42:19/0 = 0
43:19/0 = 0
44:19/0 = 0
45:19/0 = 0
46:19/0 = 0
47:19/0 = 0
48:19/0 = 0
0:20/0 = 0
1:20/0 = 0
2:20/0 = 0
3:20/0 = 0
4:20/0 = 0
```

```
5:20/0 = 0
6:20/0 = 0
7:20/0 = 0
8:20/0 = 0
9:20/0 = 0
10:20/0 = 0
11:20/0 = 0
12:20/0 = 0
13:20/0 = 0
14:20/0 = 0
15:20/0 = 0
16:20/0 = 0
17:20/0 = 0
18:20/0 = 0
19:20/0 = 0
20:20/0 = 0
21:20/0 = 0
22:20/0 = 0
23:20/0 = 0
24:20/0 = 0
25:20/0 = 0
26:20/0 = 0
27:20/0 = 0
28:20/0 = 0
29:20/0 = 0
30:20/0 = 0
31:20/0 = 0
32:20/0 = 0
33:20/0 = 0
34:20/0 = 0
35:20/0 = 0
36:20/0 = 0
37:20/0 = 0
```

```
1028 38:20/0 = 0
1029 39:20/0 = 0
1030 40:20/0 = 0
1031 41:20/0 = 0
1032 42:20/0 = 0
1033 43:20/0 = 0
1034 44:20/0 = 0
1035 45:20/0 = 0
1036 46:20/0 = 0
1037 47:20/0 = 0
1038 48:20/0 = 0
1039 0:21/0 = 0
1040 1:21/0 = 0
1041 2:21/0 = 0
1042 3:21/0 = 0
1043 4:21/0 = 0
1044 5:21/0 = 0
1045 6:21/0 = 0
1046 7:21/0 = 0
1047 8:21/0 = 0
1048 9:21/0 = 0
1049 10:21/0 = 0
1050 11:21/0 = 0
1051 12:21/0 = 0
1052 13:21/0 = 0
1053 14:21/0 = 0
1054 15:21/0 = 0
1055 16:21/0 = 0
1056 17:21/0 = 0
1057 18:21/0 = 0
1058 19:21/0 = 0
1059 20:21/0 = 0
1060 21:21/0 = 0
```

```
22:21/0 = 0
23:21/0 = 0
24:21/0 = 0
25:21/0 = 0
26:21/0 = 0
27:21/0 = 0
28:21/0 = 0
29:21/0 = 0
30:21/0 = 0
31:21/0 = 0
32:21/0 = 0
33:21/0 = 0
34:21/0 = 0
35:21/0 = 0
36:21/0 = 0
37:21/0 = 0
38:21/0 = 0
39:21/0 = 0
40:21/0 = 0
41:21/0 = 0
42:21/0 = 0
43:21/0 = 0
44:21/0 = 0
45:21/0 = 0
46:21/0 = 0
47:21/0 = 0
48:21/0 = 0

[sub_resource type="TileSet" id="TileSet_3drs5"]
sources/0 = SubResource("TileSetAtlasSource_6h0bd")

[node name="TileMap" type="TileMap"]
tile_set = SubResource("TileSet_3drs5")
```

```
format = 2
script = ExtResource("2_lf4lw")

[node name="AcceptDialog" parent="." instance=ExtResource("3_y08lj"
    )]

[node name="WinDialog" parent="." instance=ExtResource("4_fkys0")]
```

C.3.5 tile_map.gd

```
extends TileMap

const buildings: Array[Vector2i] = [
   Vector2i(0, 19),
   Vector2i(1, 19),
   Vector2i(2, 19),
   Vector2i(3, 19),
   Vector2i(4, 19),
   Vector2i(5, 19),
   Vector2i(6, 19),
   Vector2i(7, 19),
   Vector2i(8, 20),
   Vector2i(0, 20),
   Vector2i(1, 20),
   Vector2i(2, 20),
   Vector2i(3, 20),
   Vector2i(4, 20),
   Vector2i(5, 20),
   Vector2i(6, 20),
   Vector2i(7, 20),
   Vector2i(8, 20),
   Vector2i(0, 21),
```

```
    Vector2i(1, 21),
    Vector2i(2, 21),
    Vector2i(3, 21),
    Vector2i(4, 21),
    Vector2i(5, 21),
    Vector2i(6, 21),
    Vector2i(7, 21),
    Vector2i(8, 21)
]
const trees: Array[Vector2i] = [
    Vector2i(0,1),
    Vector2i(1,1),
    Vector2i(2,1),
    Vector2i(3,1),
    Vector2i(4,1),
    Vector2i(5,1),
    Vector2i(6,1),
    Vector2i(7,1),
    Vector2i(0,2),
    Vector2i(1,2),
    Vector2i(2,2),
    Vector2i(3,2),
    Vector2i(4,2)
]
const PLAYER_SPRITE: Vector2i = Vector2i(24, 7)
var player_placement_cell: Vector2i
const rings: Array[Vector2i] = [
    Vector2i(43, 6),
    Vector2i(44, 6),
    Vector2i(45, 6),
    Vector2i(46, 6)
]
var ring_placement_cell: Vector2i
```

```

var points: Array[Dictionary] = []
const EUCLIDEAN: String = "Euclidean distance"
const MANHATTAN: String = "Manhattan distance"
@export_enum(EUCLIDEAN, MANHATTAN) var distance: String = MANHATTAN
@export_range(10, 40, 1) var random_starting_points: int = 20
var x_tile_range: int = ProjectSettings.get_setting("display/window
    /size/viewport_width") / tile_set.tile_size.x
var y_tile_range: int = ProjectSettings.get_setting("display/window
    /size/viewport_height") / tile_set.tile_size.y

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
    randomize()
    var start_time: float = Time.get_ticks_msec()
    define_points(random_starting_points)
    paint_points()
    place_player()
    place_ring()
    var new_time: float = Time.get_ticks_msec() - start_time
    print("Time taken: " + str(new_time) + "ms")
    $AcceptDialog.dialog_text = "You're a hollow Golem who seeks the
         ultimate treasure; a ring that's got something on top of it
        . It's somewhere in this large village and barely visible to
         your naked eyes, which took us " + str(new_time) + "
        milliseconds to generate (" + str(new_time / 1000.0) + "
        seconds), but you'll stop at nothing to get what you want.
        You can chow down every tree and fauna that stands in your
        way of the ring, but your Achilles heel is any bricks and
        mortar, which WILL make you stop at your tracks. Since it's
        easy to get lost in here, we'll tell you that you're in
        position " + str(player_placement_cell) + " in this big
        village of size " + str(Vector2i(x_tile_range, y_tile_range)
```

```
        ) + ". It's also easy to get stuck here, so either press the
         G key or right click to spawn somewhere else where there is
         fauna (or even the ring!!), because this game actually
        WANTS you to win it. Ultimately, though, it is YOUR job to
        find the ring, so are you ready to attain the treasure that
        is rightfully yours?!"
    $AcceptDialog.visible = true
    $AcceptDialog.confirmed.connect(_on_AcceptDialog_closed)
    $AcceptDialog.canceled.connect(_on_AcceptDialog_closed)
    $WinDialog.confirmed.connect(_on_WinDialog_confirmed)
    $WinDialog.canceled.connect(_on_WinDialog_canceled)
    get_tree().paused = true

func _on_WinDialog_confirmed() -> void:
    get_tree().reload_current_scene()

func _on_WinDialog_canceled() -> void:
    get_tree().quit()

func _on_AcceptDialog_closed() -> void:
    $AcceptDialog.visible = false
    get_tree().paused = false

func _get_random_placement_cell() -> Vector2i:
    return Vector2i(randi() % x_tile_range, randi() % y_tile_range)

func place_player() -> void:
    player_placement_cell = _get_random_placement_cell()
    while buildings.has(get_cell_atlas_coords(0,
        player_placement_cell)) or player_placement_cell ==
        ring_placement_cell:
        player_placement_cell = _get_random_placement_cell()
    set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)
```

```

func place_ring() -> void:
	ring_placement_cell = _get_random_placement_cell()
	while buildings.has(get_cell_atlas_coords(0, ring_placement_cell)) or ring_placement_cell == player_placement_cell:
		ring_placement_cell = _get_random_placement_cell()
	set_cell(0, ring_placement_cell, 0, rings.pick_random())

func _is_not_out_of_bounds(cell: Vector2i) -> bool:
	return cell.x >= 0 and cell.x < x_tile_range and cell.y >= 0 and cell.y < y_tile_range

func _physics_process(_delta) -> void:
	var previous_cell: Vector2i = player_placement_cell
	var direction: Vector2i = Vector2i.ZERO
	if Input.is_action_pressed("ui_up"): direction = Vector2i.UP
	elif Input.is_action_pressed("ui_down"): direction = Vector2i.DOWN
	elif Input.is_action_pressed("ui_left"): direction = Vector2i.LEFT
	elif Input.is_action_pressed("ui_right"): direction = Vector2i.RIGHT
	elif Input.is_action_just_pressed("reset_position"): # Respawn player in a different part of the map
		player_placement_cell = _get_random_placement_cell()
		while buildings.has(get_cell_atlas_coords(0, player_placement_cell)): # This time, since we're not STARTING the game, we don't care whether or not the player magically lands on the ring
			player_placement_cell = _get_random_placement_cell()
		set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)
		set_cell(0, previous_cell, 0) # replace the previous sprite
		return
```

```
    var new_placement_cell: Vector2i = player_placement_cell + direction
    if (not get_used_cells(0).has(new_placement_cell) or trees.has(get_cell_atlas_coords(0, new_placement_cell)) or new_placement_cell == ring_placement_cell) and _is_not_out_of_bounds(new_placement_cell):
        player_placement_cell = new_placement_cell
        set_cell(0, previous_cell, 0) # deletes contents of previous cell (atlas_coords = Vector2i(-1, -1))
        set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)
        if player_placement_cell == ring_placement_cell:
            $WinDialog.visible = true
            get_tree().paused = true

# ALGORITHM BEGINS HERE

func paint_points() -> void:
    for point in points:
        set_cell(0, Vector2(point["x"], point["y"]), 0, point["type"])
        for citizen in point["citizens"]:
            if _is_in_bounds(point["x"], citizen["dx"], point["y"], citizen["dy"]):
                set_cell(0, Vector2(point["x"] + citizen["dx"], point["y"] + citizen["dy"]), 0, point["type"])

func _is_in_bounds(x: int, dx: int, y: int, dy: int) -> bool:
    return x + dx >= 0 and x + dx < x_tile_range and y + dy >= 0 and y + dy < y_tile_range

func _squared(x: int) -> int:
    return x ** 2

```

```
func calculate_points_delta(x: int, y: int, p: int) -> float:
	if distance == EUCLIDEAN:
		return sqrt(_squared(points[p]["x"] - x) + _squared(points[p
			]["y"] - y))
	return abs(points[p]["x"] - x) + abs(points[p]["y"] - y)

func define_points(num_points: int) -> void:
	var types: Array[Vector2i] = trees.duplicate()
	types.append_array(buildings)
	for i in range(num_points):
		var x: int = randi_range(0, x_tile_range)
		var y: int = randi_range(0, y_tile_range)
		var type: Vector2i = types.pick_random()
		types.erase(type)
		points.append(
			{
				"type": type,
				"x": x,
				"y": y,
				"citizens": []
			}
		)
	for x in range(x_tile_range):
		for y in range(y_tile_range):
			var lowest_delta: Dictionary = {
				"point_id": 0,
				"delta": x_tile_range * y_tile_range
			}
			for p in range(len(points)):
				var delta: float = calculate_points_delta(x, y, p)
				if delta < lowest_delta["delta"]:
					lowest_delta = {
						"point_id": p,
```

```
                    "delta": delta
                }
            var active_point: Dictionary = points[lowest_delta["point_id"]]
            var dx: int = x - active_point["x"]
            var dy: int = y - active_point["y"]
            active_point["citizens"].append(
                {
                    "dx": dx,
                    "dy": dy
                }
            )
```

C.3.6 accept_dialog.tscn

```
[gd_scene format=3 uid="uid://cau5jgogdnf53"]

[node name="AcceptDialog" type="AcceptDialog"]
title = "Tree-Munching Time!"
position = Vector2i(326, 100)
size = Vector2i(500, 421)
mouse_passthrough = true
ok_button_text = "Bring it on!"
dialog_text = "You're a hollow Golem who seeks the ultimate treasure; a ring that's got something on top of it. It's somewhere in this large village and barely visible to your naked eyes, but you'll stop at nothing to get what you want. You can chow down every tree and fauna that stands in your way of the ring, but your Achilles heel is any bricks and mortar, which will make you stop at your tracks. Are you ready to attain your treasure?w Golem in a black-and-white world, in search for your most desired treasure. It's a ring with
```

```
something on top of it. And you'll stop at nothing to get what you want. You can chow down every tree and fauna that stands in your way of the ring, but your Achilles heel is any bricks and mortar, which will make you stop at your tracks. Are you ready to attain the treasure that is rightfully yours?!"
dialog_autowrap = true
```

C.3.7 win_dialog.tscn

```
[gd_scene format=3 uid="uid://b5q8ovcigrvyr"]

[node name="WinDialog" type="ConfirmationDialog"]
title = "You Found the Treasure!"
position = Vector2i(326, 100)
size = Vector2i(500, 421)
mouse_passthrough = true
ok_button_text = "Get Me a New Village"
dialog_text = "You found your treasure! Well done, you!

Would you like to travel to a new village in the hopes of finding another ring? Or would you like to take your treasure home now?
"
dialog_autowrap = true
cancel_button_text = "Get Me Out of Here"
```

C.3.8 icon.svg.import

```
[remap]

importer="texture"
type="CompressedTexture2D"
```

```
uid="uid://du4v6taw8ssax"
path="res://.godot/imported/icon.svg-218a8f2b3041327d8a5756f3a245f83b.ctex"
metadata={
"vram_texture": false
}

[deps]

source_file="res://icon.svg"
dest_files=["res://.godot/imported/icon.svg-218a8f2b3041327d8a5756f3a245f83b.ctex"]

[params]

compress/mode=0
compress/high_quality=false
compress/lossy_quality=0.7
compress/hdr_compression=1
compress/normal_map=0
compress/channel_pack=0
mipmaps/generate=false
mipmaps/limit=-1
roughness/mode=0
roughness/src_normal=""
process/fix_alpha_border=true
process/premult_alpha=false
process/normal_map_invert_y=false
process/hdr_as_srgb=false
process/hdr_clamp_exposure=false
process/size_limit=0
detect_3d/compress_to=1
svg/scale=1.0
```

```
editor/scale_with_editor_scale=false
editor/convert_colors_with_editor_theme=false
```

C.3.9 monochrome_packed.png.import

```
[remap]

importer="texture"
type="CompressedTexture2D"
uid="uid://cpign73sfbsrt"
path="res://.godot/imported/monochrome_packed.png-6b9bd1c64dd50f72acd3afd14d1ac34f.ctex"
metadata={
"vram_texture": false
}

[deps]

source_file="res://monochrome_packed.png"
dest_files=["res://.godot/imported/monochrome_packed.png-6b9bd1c64dd50f72acd3afd14d1ac34f.ctex"]

[params]

compress/mode=0
compress/high_quality=false
compress/lossy_quality=0.7
compress/hdr_compression=1
compress/normal_map=0
compress/channel_pack=0
mipmaps/generate=false
mipmaps/limit=-1
```

```
roughness/mode=0
roughness/src_normal=""
process/fix_alpha_border=true
process/premult_alpha=false
process/normal_map_invert_y=false
process/hdr_as_srgb=false
process/hdr_clamp_exposure=false
process/size_limit=0
detect_3d/compress_to=1
```

C.4 PoissonGD4

C.4.1 .gitattributes

```
# Normalize EOL for all files that Git considers text files.
* text=auto eol=lf
```

C.4.2 .gitignore

```
# Godot 4+ specific ignores
.godot/
```

C.4.3 project.godot

```
; Engine configuration file.
; It's best edited using the editor UI and not directly,
; since the parameters that go here are not all obvious.
;
; Format:
```

```
;   [section] ; section goes between []
;   param=value ; assign values to parameters

config_version=5

[application]

config/name="Voronoi Cells"
run/main_scene="res://tile_map.tscn"
config/features=PackedStringArray("4.0", "Forward Plus")
config/icon="res://icon.svg"

[display]

window/size/viewport_height=640

[input]

reset_position={
"deadzone": 0.5,
"events": [Object(InputEventKey,"resource_local_to_scene":false,"resource_name":"","device":-1,"window_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_pressed":false,"pressed":false,"keycode":71,"physical_keycode":0,"key_label":0,"unicode":103,"echo":false,"script":null)
, Object(InputEventMouseButton,"resource_local_to_scene":false,"resource_name":"","device":-1,"window_id":0,"alt_pressed":false,"shift_pressed":false,"ctrl_pressed":false,"meta_pressed":false,"button_mask":2,"position":Vector2(75, 12),"global_position":Vector2(78, 44),"factor":1.0,"button_index":2,"pressed":true,"double_click":false,"script":null)
]
}
```

```

[rendering]

environment/defaults/default_clear_color=Color(0, 0, 0, 1)
```

C.4.4 tile_map.tscn

```
[gd_scene load_steps=7 format=3 uid="uid://d6lxnr5bdh1w"]

[ext_resource type="Texture2D" uid="uid://cpign73sfbsrt" path="res://monochrome_packed.png" id="1_o183d"]
[ext_resource type="Script" path="res://tile_map.gd" id="2_lf4lw"]
[ext_resource type="PackedScene" path="res://accept_dialog.tscn" id="3_y08lj"]
[ext_resource type="PackedScene" path="res://win_dialog.tscn" id="4_fkys0"]

[sub_resource type="TileSetAtlasSource" id="TileSetAtlasSource_6h0bd"]
texture = ExtResource("1_o183d")
0:0/0 = 0
1:0/0 = 0
2:0/0 = 0
3:0/0 = 0
4:0/0 = 0
5:0/0 = 0
6:0/0 = 0
7:0/0 = 0
8:0/0 = 0
9:0/0 = 0
10:0/0 = 0
11:0/0 = 0
```

```
12:0/0 = 0
13:0/0 = 0
14:0/0 = 0
15:0/0 = 0
16:0/0 = 0
17:0/0 = 0
18:0/0 = 0
19:0/0 = 0
20:0/0 = 0
21:0/0 = 0
22:0/0 = 0
23:0/0 = 0
24:0/0 = 0
25:0/0 = 0
26:0/0 = 0
27:0/0 = 0
28:0/0 = 0
29:0/0 = 0
30:0/0 = 0
31:0/0 = 0
32:0/0 = 0
33:0/0 = 0
34:0/0 = 0
35:0/0 = 0
36:0/0 = 0
37:0/0 = 0
38:0/0 = 0
39:0/0 = 0
40:0/0 = 0
41:0/0 = 0
42:0/0 = 0
43:0/0 = 0
44:0/0 = 0
```

```
45:0/0 = 0
46:0/0 = 0
47:0/0 = 0
48:0/0 = 0
0:1/0 = 0
1:1/0 = 0
2:1/0 = 0
3:1/0 = 0
4:1/0 = 0
5:1/0 = 0
6:1/0 = 0
7:1/0 = 0
8:1/0 = 0
9:1/0 = 0
10:1/0 = 0
11:1/0 = 0
12:1/0 = 0
13:1/0 = 0
14:1/0 = 0
15:1/0 = 0
16:1/0 = 0
17:1/0 = 0
18:1/0 = 0
19:1/0 = 0
20:1/0 = 0
21:1/0 = 0
22:1/0 = 0
23:1/0 = 0
24:1/0 = 0
25:1/0 = 0
26:1/0 = 0
27:1/0 = 0
28:1/0 = 0
```

```
29:1/0 = 0
30:1/0 = 0
31:1/0 = 0
32:1/0 = 0
33:1/0 = 0
34:1/0 = 0
35:1/0 = 0
36:1/0 = 0
37:1/0 = 0
38:1/0 = 0
39:1/0 = 0
40:1/0 = 0
41:1/0 = 0
42:1/0 = 0
43:1/0 = 0
44:1/0 = 0
45:1/0 = 0
46:1/0 = 0
47:1/0 = 0
48:1/0 = 0
0:2/0 = 0
1:2/0 = 0
2:2/0 = 0
3:2/0 = 0
4:2/0 = 0
5:2/0 = 0
6:2/0 = 0
7:2/0 = 0
8:2/0 = 0
9:2/0 = 0
10:2/0 = 0
11:2/0 = 0
12:2/0 = 0
```

```
13:2/0 = 0
14:2/0 = 0
15:2/0 = 0
16:2/0 = 0
17:2/0 = 0
18:2/0 = 0
19:2/0 = 0
20:2/0 = 0
21:2/0 = 0
22:2/0 = 0
23:2/0 = 0
24:2/0 = 0
25:2/0 = 0
26:2/0 = 0
27:2/0 = 0
28:2/0 = 0
29:2/0 = 0
30:2/0 = 0
31:2/0 = 0
32:2/0 = 0
33:2/0 = 0
34:2/0 = 0
35:2/0 = 0
36:2/0 = 0
37:2/0 = 0
38:2/0 = 0
39:2/0 = 0
40:2/0 = 0
41:2/0 = 0
42:2/0 = 0
43:2/0 = 0
44:2/0 = 0
45:2/0 = 0
```

```
46:2/0 = 0
47:2/0 = 0
48:2/0 = 0
0:3/0 = 0
1:3/0 = 0
2:3/0 = 0
3:3/0 = 0
4:3/0 = 0
5:3/0 = 0
6:3/0 = 0
7:3/0 = 0
8:3/0 = 0
9:3/0 = 0
10:3/0 = 0
11:3/0 = 0
12:3/0 = 0
13:3/0 = 0
14:3/0 = 0
15:3/0 = 0
16:3/0 = 0
17:3/0 = 0
18:3/0 = 0
19:3/0 = 0
20:3/0 = 0
21:3/0 = 0
22:3/0 = 0
23:3/0 = 0
24:3/0 = 0
25:3/0 = 0
26:3/0 = 0
27:3/0 = 0
28:3/0 = 0
29:3/0 = 0
```

```
30:3/0 = 0
31:3/0 = 0
32:3/0 = 0
33:3/0 = 0
34:3/0 = 0
35:3/0 = 0
36:3/0 = 0
37:3/0 = 0
38:3/0 = 0
39:3/0 = 0
40:3/0 = 0
41:3/0 = 0
42:3/0 = 0
43:3/0 = 0
44:3/0 = 0
45:3/0 = 0
46:3/0 = 0
47:3/0 = 0
48:3/0 = 0
0:4/0 = 0
1:4/0 = 0
2:4/0 = 0
3:4/0 = 0
4:4/0 = 0
5:4/0 = 0
6:4/0 = 0
7:4/0 = 0
8:4/0 = 0
9:4/0 = 0
10:4/0 = 0
11:4/0 = 0
12:4/0 = 0
13:4/0 = 0
```

```
14:4/0 = 0
15:4/0 = 0
16:4/0 = 0
17:4/0 = 0
18:4/0 = 0
19:4/0 = 0
20:4/0 = 0
21:4/0 = 0
22:4/0 = 0
23:4/0 = 0
24:4/0 = 0
25:4/0 = 0
26:4/0 = 0
27:4/0 = 0
28:4/0 = 0
29:4/0 = 0
30:4/0 = 0
31:4/0 = 0
32:4/0 = 0
33:4/0 = 0
34:4/0 = 0
35:4/0 = 0
36:4/0 = 0
37:4/0 = 0
38:4/0 = 0
39:4/0 = 0
40:4/0 = 0
41:4/0 = 0
42:4/0 = 0
43:4/0 = 0
44:4/0 = 0
45:4/0 = 0
46:4/0 = 0
```

```
47:4/0 = 0
48:4/0 = 0
0:5/0 = 0
1:5/0 = 0
2:5/0 = 0
3:5/0 = 0
4:5/0 = 0
5:5/0 = 0
6:5/0 = 0
7:5/0 = 0
8:5/0 = 0
9:5/0 = 0
10:5/0 = 0
11:5/0 = 0
12:5/0 = 0
13:5/0 = 0
14:5/0 = 0
15:5/0 = 0
16:5/0 = 0
17:5/0 = 0
18:5/0 = 0
19:5/0 = 0
20:5/0 = 0
21:5/0 = 0
22:5/0 = 0
23:5/0 = 0
24:5/0 = 0
25:5/0 = 0
26:5/0 = 0
27:5/0 = 0
28:5/0 = 0
29:5/0 = 0
30:5/0 = 0
```

```
31:5/0 = 0
32:5/0 = 0
33:5/0 = 0
34:5/0 = 0
35:5/0 = 0
36:5/0 = 0
37:5/0 = 0
38:5/0 = 0
39:5/0 = 0
40:5/0 = 0
41:5/0 = 0
42:5/0 = 0
43:5/0 = 0
44:5/0 = 0
45:5/0 = 0
46:5/0 = 0
47:5/0 = 0
48:5/0 = 0
0:6/0 = 0
1:6/0 = 0
2:6/0 = 0
3:6/0 = 0
4:6/0 = 0
5:6/0 = 0
6:6/0 = 0
7:6/0 = 0
8:6/0 = 0
9:6/0 = 0
10:6/0 = 0
11:6/0 = 0
12:6/0 = 0
13:6/0 = 0
14:6/0 = 0
```

```
15:6/0 = 0
16:6/0 = 0
17:6/0 = 0
18:6/0 = 0
19:6/0 = 0
20:6/0 = 0
21:6/0 = 0
22:6/0 = 0
23:6/0 = 0
24:6/0 = 0
25:6/0 = 0
26:6/0 = 0
27:6/0 = 0
28:6/0 = 0
29:6/0 = 0
30:6/0 = 0
31:6/0 = 0
32:6/0 = 0
33:6/0 = 0
34:6/0 = 0
35:6/0 = 0
36:6/0 = 0
37:6/0 = 0
38:6/0 = 0
39:6/0 = 0
40:6/0 = 0
41:6/0 = 0
42:6/0 = 0
43:6/0 = 0
44:6/0 = 0
45:6/0 = 0
46:6/0 = 0
47:6/0 = 0
```

```
48:6/0 = 0
0:7/0 = 0
1:7/0 = 0
2:7/0 = 0
3:7/0 = 0
4:7/0 = 0
5:7/0 = 0
6:7/0 = 0
7:7/0 = 0
8:7/0 = 0
9:7/0 = 0
10:7/0 = 0
11:7/0 = 0
12:7/0 = 0
13:7/0 = 0
14:7/0 = 0
15:7/0 = 0
16:7/0 = 0
17:7/0 = 0
18:7/0 = 0
19:7/0 = 0
20:7/0 = 0
21:7/0 = 0
22:7/0 = 0
23:7/0 = 0
24:7/0 = 0
25:7/0 = 0
26:7/0 = 0
27:7/0 = 0
28:7/0 = 0
29:7/0 = 0
30:7/0 = 0
31:7/0 = 0
```

```
32:7/0 = 0
33:7/0 = 0
34:7/0 = 0
35:7/0 = 0
36:7/0 = 0
37:7/0 = 0
38:7/0 = 0
39:7/0 = 0
40:7/0 = 0
41:7/0 = 0
42:7/0 = 0
43:7/0 = 0
44:7/0 = 0
45:7/0 = 0
46:7/0 = 0
47:7/0 = 0
48:7/0 = 0
0:8/0 = 0
1:8/0 = 0
2:8/0 = 0
3:8/0 = 0
4:8/0 = 0
5:8/0 = 0
6:8/0 = 0
7:8/0 = 0
8:8/0 = 0
9:8/0 = 0
10:8/0 = 0
11:8/0 = 0
12:8/0 = 0
13:8/0 = 0
14:8/0 = 0
15:8/0 = 0
```

```
16:8/0 = 0
17:8/0 = 0
18:8/0 = 0
19:8/0 = 0
20:8/0 = 0
21:8/0 = 0
22:8/0 = 0
23:8/0 = 0
24:8/0 = 0
25:8/0 = 0
26:8/0 = 0
27:8/0 = 0
28:8/0 = 0
29:8/0 = 0
30:8/0 = 0
31:8/0 = 0
32:8/0 = 0
33:8/0 = 0
34:8/0 = 0
35:8/0 = 0
36:8/0 = 0
37:8/0 = 0
38:8/0 = 0
39:8/0 = 0
40:8/0 = 0
41:8/0 = 0
42:8/0 = 0
43:8/0 = 0
44:8/0 = 0
45:8/0 = 0
46:8/0 = 0
47:8/0 = 0
48:8/0 = 0
```

```
0:9/0 = 0
1:9/0 = 0
2:9/0 = 0
3:9/0 = 0
4:9/0 = 0
5:9/0 = 0
6:9/0 = 0
7:9/0 = 0
8:9/0 = 0
9:9/0 = 0
10:9/0 = 0
11:9/0 = 0
12:9/0 = 0
13:9/0 = 0
14:9/0 = 0
15:9/0 = 0
16:9/0 = 0
17:9/0 = 0
18:9/0 = 0
19:9/0 = 0
20:9/0 = 0
21:9/0 = 0
22:9/0 = 0
23:9/0 = 0
24:9/0 = 0
25:9/0 = 0
26:9/0 = 0
27:9/0 = 0
28:9/0 = 0
29:9/0 = 0
30:9/0 = 0
31:9/0 = 0
32:9/0 = 0
```

```
33:9/0 = 0
34:9/0 = 0
35:9/0 = 0
36:9/0 = 0
37:9/0 = 0
38:9/0 = 0
39:9/0 = 0
40:9/0 = 0
41:9/0 = 0
42:9/0 = 0
43:9/0 = 0
44:9/0 = 0
45:9/0 = 0
46:9/0 = 0
47:9/0 = 0
48:9/0 = 0
0:10/0 = 0
1:10/0 = 0
2:10/0 = 0
3:10/0 = 0
4:10/0 = 0
5:10/0 = 0
6:10/0 = 0
7:10/0 = 0
8:10/0 = 0
9:10/0 = 0
10:10/0 = 0
11:10/0 = 0
12:10/0 = 0
13:10/0 = 0
14:10/0 = 0
15:10/0 = 0
16:10/0 = 0
```

```
17:10/0 = 0
18:10/0 = 0
19:10/0 = 0
20:10/0 = 0
21:10/0 = 0
22:10/0 = 0
23:10/0 = 0
24:10/0 = 0
25:10/0 = 0
26:10/0 = 0
27:10/0 = 0
28:10/0 = 0
29:10/0 = 0
30:10/0 = 0
31:10/0 = 0
32:10/0 = 0
33:10/0 = 0
34:10/0 = 0
35:10/0 = 0
36:10/0 = 0
37:10/0 = 0
38:10/0 = 0
39:10/0 = 0
40:10/0 = 0
41:10/0 = 0
42:10/0 = 0
43:10/0 = 0
44:10/0 = 0
45:10/0 = 0
46:10/0 = 0
47:10/0 = 0
48:10/0 = 0
0:11/0 = 0
```

```
1:11/0 = 0
2:11/0 = 0
3:11/0 = 0
4:11/0 = 0
5:11/0 = 0
6:11/0 = 0
7:11/0 = 0
8:11/0 = 0
9:11/0 = 0
10:11/0 = 0
11:11/0 = 0
12:11/0 = 0
13:11/0 = 0
14:11/0 = 0
15:11/0 = 0
16:11/0 = 0
17:11/0 = 0
18:11/0 = 0
19:11/0 = 0
20:11/0 = 0
21:11/0 = 0
22:11/0 = 0
23:11/0 = 0
24:11/0 = 0
25:11/0 = 0
26:11/0 = 0
27:11/0 = 0
28:11/0 = 0
29:11/0 = 0
30:11/0 = 0
31:11/0 = 0
32:11/0 = 0
33:11/0 = 0
```

```
34:11/0 = 0
35:11/0 = 0
36:11/0 = 0
37:11/0 = 0
38:11/0 = 0
39:11/0 = 0
40:11/0 = 0
41:11/0 = 0
42:11/0 = 0
43:11/0 = 0
44:11/0 = 0
45:11/0 = 0
46:11/0 = 0
47:11/0 = 0
48:11/0 = 0
0:12/0 = 0
1:12/0 = 0
2:12/0 = 0
3:12/0 = 0
4:12/0 = 0
5:12/0 = 0
6:12/0 = 0
7:12/0 = 0
8:12/0 = 0
9:12/0 = 0
10:12/0 = 0
11:12/0 = 0
12:12/0 = 0
13:12/0 = 0
14:12/0 = 0
15:12/0 = 0
16:12/0 = 0
17:12/0 = 0
```

```
18:12/0 = 0
19:12/0 = 0
20:12/0 = 0
21:12/0 = 0
22:12/0 = 0
23:12/0 = 0
24:12/0 = 0
25:12/0 = 0
26:12/0 = 0
27:12/0 = 0
28:12/0 = 0
29:12/0 = 0
30:12/0 = 0
31:12/0 = 0
32:12/0 = 0
33:12/0 = 0
34:12/0 = 0
35:12/0 = 0
36:12/0 = 0
37:12/0 = 0
38:12/0 = 0
39:12/0 = 0
40:12/0 = 0
41:12/0 = 0
42:12/0 = 0
43:12/0 = 0
44:12/0 = 0
45:12/0 = 0
46:12/0 = 0
47:12/0 = 0
48:12/0 = 0
0:13/0 = 0
1:13/0 = 0
```

```
2:13/0 = 0
3:13/0 = 0
4:13/0 = 0
5:13/0 = 0
6:13/0 = 0
7:13/0 = 0
8:13/0 = 0
9:13/0 = 0
10:13/0 = 0
11:13/0 = 0
12:13/0 = 0
13:13/0 = 0
14:13/0 = 0
15:13/0 = 0
16:13/0 = 0
17:13/0 = 0
18:13/0 = 0
19:13/0 = 0
20:13/0 = 0
21:13/0 = 0
22:13/0 = 0
23:13/0 = 0
24:13/0 = 0
25:13/0 = 0
26:13/0 = 0
27:13/0 = 0
28:13/0 = 0
29:13/0 = 0
30:13/0 = 0
31:13/0 = 0
32:13/0 = 0
33:13/0 = 0
34:13/0 = 0
```

```
35:13/0 = 0
36:13/0 = 0
37:13/0 = 0
38:13/0 = 0
39:13/0 = 0
40:13/0 = 0
41:13/0 = 0
42:13/0 = 0
43:13/0 = 0
44:13/0 = 0
45:13/0 = 0
46:13/0 = 0
47:13/0 = 0
48:13/0 = 0
0:14/0 = 0
1:14/0 = 0
2:14/0 = 0
3:14/0 = 0
4:14/0 = 0
5:14/0 = 0
6:14/0 = 0
7:14/0 = 0
8:14/0 = 0
9:14/0 = 0
10:14/0 = 0
11:14/0 = 0
12:14/0 = 0
13:14/0 = 0
14:14/0 = 0
15:14/0 = 0
16:14/0 = 0
17:14/0 = 0
18:14/0 = 0
```

```
19:14/0 = 0
20:14/0 = 0
21:14/0 = 0
22:14/0 = 0
23:14/0 = 0
24:14/0 = 0
25:14/0 = 0
26:14/0 = 0
27:14/0 = 0
28:14/0 = 0
29:14/0 = 0
30:14/0 = 0
31:14/0 = 0
32:14/0 = 0
33:14/0 = 0
34:14/0 = 0
35:14/0 = 0
36:14/0 = 0
37:14/0 = 0
38:14/0 = 0
39:14/0 = 0
40:14/0 = 0
41:14/0 = 0
42:14/0 = 0
43:14/0 = 0
44:14/0 = 0
45:14/0 = 0
46:14/0 = 0
47:14/0 = 0
48:14/0 = 0
0:15/0 = 0
1:15/0 = 0
2:15/0 = 0
```

```
3:15/0 = 0
4:15/0 = 0
5:15/0 = 0
6:15/0 = 0
7:15/0 = 0
8:15/0 = 0
9:15/0 = 0
10:15/0 = 0
11:15/0 = 0
12:15/0 = 0
13:15/0 = 0
14:15/0 = 0
15:15/0 = 0
16:15/0 = 0
17:15/0 = 0
18:15/0 = 0
19:15/0 = 0
20:15/0 = 0
21:15/0 = 0
22:15/0 = 0
23:15/0 = 0
24:15/0 = 0
25:15/0 = 0
26:15/0 = 0
27:15/0 = 0
28:15/0 = 0
29:15/0 = 0
30:15/0 = 0
31:15/0 = 0
32:15/0 = 0
33:15/0 = 0
34:15/0 = 0
35:15/0 = 0
```

```
36:15/0 = 0
37:15/0 = 0
38:15/0 = 0
39:15/0 = 0
40:15/0 = 0
41:15/0 = 0
42:15/0 = 0
43:15/0 = 0
44:15/0 = 0
45:15/0 = 0
46:15/0 = 0
47:15/0 = 0
48:15/0 = 0
0:16/0 = 0
1:16/0 = 0
2:16/0 = 0
3:16/0 = 0
4:16/0 = 0
5:16/0 = 0
6:16/0 = 0
7:16/0 = 0
8:16/0 = 0
9:16/0 = 0
10:16/0 = 0
11:16/0 = 0
12:16/0 = 0
13:16/0 = 0
14:16/0 = 0
15:16/0 = 0
16:16/0 = 0
17:16/0 = 0
18:16/0 = 0
19:16/0 = 0
```

```
20:16/0 = 0
21:16/0 = 0
22:16/0 = 0
23:16/0 = 0
24:16/0 = 0
25:16/0 = 0
26:16/0 = 0
27:16/0 = 0
28:16/0 = 0
29:16/0 = 0
30:16/0 = 0
31:16/0 = 0
32:16/0 = 0
33:16/0 = 0
34:16/0 = 0
35:16/0 = 0
36:16/0 = 0
37:16/0 = 0
38:16/0 = 0
39:16/0 = 0
40:16/0 = 0
41:16/0 = 0
42:16/0 = 0
43:16/0 = 0
44:16/0 = 0
45:16/0 = 0
46:16/0 = 0
47:16/0 = 0
48:16/0 = 0
0:17/0 = 0
1:17/0 = 0
2:17/0 = 0
3:17/0 = 0
```

```
4:17/0 = 0
5:17/0 = 0
6:17/0 = 0
7:17/0 = 0
8:17/0 = 0
9:17/0 = 0
10:17/0 = 0
11:17/0 = 0
12:17/0 = 0
13:17/0 = 0
14:17/0 = 0
15:17/0 = 0
16:17/0 = 0
17:17/0 = 0
18:17/0 = 0
19:17/0 = 0
20:17/0 = 0
21:17/0 = 0
22:17/0 = 0
23:17/0 = 0
24:17/0 = 0
25:17/0 = 0
26:17/0 = 0
27:17/0 = 0
28:17/0 = 0
29:17/0 = 0
30:17/0 = 0
31:17/0 = 0
32:17/0 = 0
33:17/0 = 0
34:17/0 = 0
35:17/0 = 0
36:17/0 = 0
```

```
37:17/0 = 0
38:17/0 = 0
39:17/0 = 0
40:17/0 = 0
41:17/0 = 0
42:17/0 = 0
43:17/0 = 0
44:17/0 = 0
45:17/0 = 0
46:17/0 = 0
47:17/0 = 0
48:17/0 = 0
0:18/0 = 0
1:18/0 = 0
2:18/0 = 0
3:18/0 = 0
4:18/0 = 0
5:18/0 = 0
6:18/0 = 0
7:18/0 = 0
8:18/0 = 0
9:18/0 = 0
10:18/0 = 0
11:18/0 = 0
12:18/0 = 0
13:18/0 = 0
14:18/0 = 0
15:18/0 = 0
16:18/0 = 0
17:18/0 = 0
18:18/0 = 0
19:18/0 = 0
20:18/0 = 0
```

```
21:18/0 = 0
22:18/0 = 0
23:18/0 = 0
24:18/0 = 0
25:18/0 = 0
26:18/0 = 0
27:18/0 = 0
28:18/0 = 0
29:18/0 = 0
30:18/0 = 0
31:18/0 = 0
32:18/0 = 0
33:18/0 = 0
34:18/0 = 0
35:18/0 = 0
36:18/0 = 0
37:18/0 = 0
38:18/0 = 0
39:18/0 = 0
40:18/0 = 0
41:18/0 = 0
42:18/0 = 0
43:18/0 = 0
44:18/0 = 0
45:18/0 = 0
46:18/0 = 0
47:18/0 = 0
48:18/0 = 0
0:19/0 = 0
1:19/0 = 0
2:19/0 = 0
3:19/0 = 0
4:19/0 = 0
```

```
5:19/0 = 0
6:19/0 = 0
7:19/0 = 0
8:19/0 = 0
9:19/0 = 0
10:19/0 = 0
11:19/0 = 0
12:19/0 = 0
13:19/0 = 0
14:19/0 = 0
15:19/0 = 0
16:19/0 = 0
17:19/0 = 0
18:19/0 = 0
19:19/0 = 0
20:19/0 = 0
21:19/0 = 0
22:19/0 = 0
23:19/0 = 0
24:19/0 = 0
25:19/0 = 0
26:19/0 = 0
27:19/0 = 0
28:19/0 = 0
29:19/0 = 0
30:19/0 = 0
31:19/0 = 0
32:19/0 = 0
33:19/0 = 0
34:19/0 = 0
35:19/0 = 0
36:19/0 = 0
37:19/0 = 0
```

```
38:19/0 = 0
39:19/0 = 0
40:19/0 = 0
41:19/0 = 0
42:19/0 = 0
43:19/0 = 0
44:19/0 = 0
45:19/0 = 0
46:19/0 = 0
47:19/0 = 0
48:19/0 = 0
0:20/0 = 0
1:20/0 = 0
2:20/0 = 0
3:20/0 = 0
4:20/0 = 0
5:20/0 = 0
6:20/0 = 0
7:20/0 = 0
8:20/0 = 0
9:20/0 = 0
10:20/0 = 0
11:20/0 = 0
12:20/0 = 0
13:20/0 = 0
14:20/0 = 0
15:20/0 = 0
16:20/0 = 0
17:20/0 = 0
18:20/0 = 0
19:20/0 = 0
20:20/0 = 0
21:20/0 = 0
```

```
22:20/0 = 0
23:20/0 = 0
24:20/0 = 0
25:20/0 = 0
26:20/0 = 0
27:20/0 = 0
28:20/0 = 0
29:20/0 = 0
30:20/0 = 0
31:20/0 = 0
32:20/0 = 0
33:20/0 = 0
34:20/0 = 0
35:20/0 = 0
36:20/0 = 0
37:20/0 = 0
38:20/0 = 0
39:20/0 = 0
40:20/0 = 0
41:20/0 = 0
42:20/0 = 0
43:20/0 = 0
44:20/0 = 0
45:20/0 = 0
46:20/0 = 0
47:20/0 = 0
48:20/0 = 0
0:21/0 = 0
1:21/0 = 0
2:21/0 = 0
3:21/0 = 0
4:21/0 = 0
5:21/0 = 0
```

```
6:21/0 = 0
7:21/0 = 0
8:21/0 = 0
9:21/0 = 0
10:21/0 = 0
11:21/0 = 0
12:21/0 = 0
13:21/0 = 0
14:21/0 = 0
15:21/0 = 0
16:21/0 = 0
17:21/0 = 0
18:21/0 = 0
19:21/0 = 0
20:21/0 = 0
21:21/0 = 0
22:21/0 = 0
23:21/0 = 0
24:21/0 = 0
25:21/0 = 0
26:21/0 = 0
27:21/0 = 0
28:21/0 = 0
29:21/0 = 0
30:21/0 = 0
31:21/0 = 0
32:21/0 = 0
33:21/0 = 0
34:21/0 = 0
35:21/0 = 0
36:21/0 = 0
37:21/0 = 0
38:21/0 = 0
```

```
39:21/0 = 0
40:21/0 = 0
41:21/0 = 0
42:21/0 = 0
43:21/0 = 0
44:21/0 = 0
45:21/0 = 0
46:21/0 = 0
47:21/0 = 0
48:21/0 = 0

[sub_resource type="TileSet" id="TileSet_3drs5"]
sources/0 = SubResource("TileSetAtlasSource_6h0bd")

[node name="TileMap" type="TileMap"]
tile_set = SubResource("TileSet_3drs5")
format = 2
script = ExtResource("2_lf4lw")

[node name="AcceptDialog" parent="." instance=ExtResource("3_y08lj"
    )]

[node name="WinDialog" parent="." instance=ExtResource("4_fkys0")]
```

C.4.5 tile_map.gd

```
extends TileMap

const buildings: Array[Vector2i] = [
    Vector2i(0, 19),
    Vector2i(1, 19),
    Vector2i(2, 19),
```

```
    Vector2i(3, 19),
    Vector2i(4, 19),
    Vector2i(5, 19),
    Vector2i(6, 19),
    Vector2i(7, 19),
    Vector2i(8, 20),
    Vector2i(0, 20),
    Vector2i(1, 20),
    Vector2i(2, 20),
    Vector2i(3, 20),
    Vector2i(4, 20),
    Vector2i(5, 20),
    Vector2i(6, 20),
    Vector2i(7, 20),
    Vector2i(8, 20),
    Vector2i(0, 21),
    Vector2i(1, 21),
    Vector2i(2, 21),
    Vector2i(3, 21),
    Vector2i(4, 21),
    Vector2i(5, 21),
    Vector2i(6, 21),
    Vector2i(7, 21),
    Vector2i(8, 21)
]
const trees: Array[Vector2i] = [
    Vector2i(0,1),
    Vector2i(1,1),
    Vector2i(2,1),
    Vector2i(3,1),
    Vector2i(4,1),
    Vector2i(5,1),
    Vector2i(6,1),
```

```
    Vector2i(7,1),
    Vector2i(0,2),
    Vector2i(1,2),
    Vector2i(2,2),
    Vector2i(3,2),
    Vector2i(4,2)
]
const PLAYER_SPRITE: Vector2i = Vector2i(24, 7)
var player_placement_cell: Vector2i
const rings: Array[Vector2i] = [
    Vector2i(43, 6),
    Vector2i(44, 6),
    Vector2i(45, 6),
    Vector2i(46, 6)
]
var ring_placement_cell: Vector2i

var points: Array[Dictionary] = []
const EUCLIDEAN: String = "Euclidean distance"
const MANHATTAN: String = "Manhattan distance"
@export_enum(EUCLIDEAN, MANHATTAN) var distance: String = MANHATTAN
@export_range(10, 40, 1) var random_starting_points: int = 20
var x_tile_range: int = ProjectSettings.get_setting("display/window/size/viewport_width") / tile_set.tile_size.x
var y_tile_range: int = ProjectSettings.get_setting("display/window/size/viewport_height") / tile_set.tile_size.y

# Called when the node enters the scene tree for the first time.
func _ready() -> void:
    randomize()
    var start_time: float = Time.get_ticks_msec()
    define_points(random_starting_points)
    paint_points()
```

```
    place_player()
    place_ring()
    var new_time: float = Time.get_ticks_msec() - start_time
    print("Time taken: " + str(new_time) + "ms")
    $AcceptDialog.dialog_text = "You're a hollow Golem who seeks the
        ultimate treasure; a ring that's got something on top of it
        . It's somewhere in this large village and barely visible to
        your naked eyes, which took us " + str(new_time) + "
        milliseconds to generate (" + str(new_time / 1000.0) + "
        seconds), but you'll stop at nothing to get what you want.
        You can chow down every tree and fauna that stands in your
        way of the ring, but your Achilles heel is any bricks and
        mortar, which WILL make you stop at your tracks. Since it's
        easy to get lost in here, we'll tell you that you're in
        position " + str(player_placement_cell) + " in this big
        village of size " + str(Vector2i(x_tile_range, y_tile_range)
        ) + ". It's also easy to get stuck here, so either press the
        G key or right click to spawn somewhere else where there is
        fauna (or even the ring!!), because this game actually
        WANTS you to win it. Ultimately, though, it is YOUR job to
        find the ring, so are you ready to attain the treasure that
        is rightfully yours?!"
    $AcceptDialog.visible = true
    $AcceptDialog.confirmed.connect(_on_AcceptDialog_closed)
    $AcceptDialog.canceled.connect(_on_AcceptDialog_closed)
    $WinDialog.confirmed.connect(_on_WinDialog_confirmed)
    $WinDialog.canceled.connect(_on_WinDialog_canceled)
    get_tree().paused = true

func _on_WinDialog_confirmed() -> void:
    get_tree().reload_current_scene()

func _on_WinDialog_canceled() -> void:
```

```
    get_tree().quit()

func _on_AcceptDialog_closed() -> void:
    $AcceptDialog.visible = false
    get_tree().paused = false

func _get_random_placement_cell() -> Vector2i:
    return Vector2i(randi() % x_tile_range, randi() % y_tile_range)

func place_player() -> void:
    player_placement_cell = _get_random_placement_cell()
    while buildings.has(get_cell_atlas_coords(0,
        player_placement_cell)) or player_placement_cell ==
        ring_placement_cell:
        player_placement_cell = _get_random_placement_cell()
    set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)

func place_ring() -> void:
    ring_placement_cell = _get_random_placement_cell()
    while buildings.has(get_cell_atlas_coords(0, ring_placement_cell
        )) or ring_placement_cell == player_placement_cell:
        ring_placement_cell = _get_random_placement_cell()
    set_cell(0, ring_placement_cell, 0, rings.pick_random())

func _is_not_out_of_bounds(cell: Vector2i) -> bool:
    return cell.x >= 0 and cell.x < x_tile_range and cell.y >= 0 and
         cell.y < y_tile_range

func _physics_process(_delta) -> void:
    var previous_cell: Vector2i = player_placement_cell
    var direction: Vector2i = Vector2i.ZERO
    if Input.is_action_pressed("ui_up"): direction = Vector2i.UP
    elif Input.is_action_pressed("ui_down"): direction = Vector2i.
```

```
DOWN
	elif Input.is_action_pressed("ui_left"): direction = Vector2i.LEFT
	elif Input.is_action_pressed("ui_right"): direction = Vector2i.RIGHT
	elif Input.is_action_just_pressed("reset_position"): # Respawn player in a different part of the map
		player_placement_cell = _get_random_placement_cell()
		while buildings.has(get_cell_atlas_coords(0, player_placement_cell)): # This time, since we're not STARTING the game, we don't care whether or not the player magically lands on the ring
			player_placement_cell = _get_random_placement_cell()
		set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)
		set_cell(0, previous_cell, 0) # replace the previous sprite
		return
	var new_placement_cell: Vector2i = player_placement_cell + direction
	if (not get_used_cells(0).has(new_placement_cell) or trees.has(get_cell_atlas_coords(0, new_placement_cell)) or new_placement_cell == ring_placement_cell) and _is_not_out_of_bounds(new_placement_cell):
		player_placement_cell = new_placement_cell
		set_cell(0, previous_cell, 0) # deletes contents of previous cell (atlas_coords = Vector2i(-1, -1))
		set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)
		if player_placement_cell == ring_placement_cell:
			$WinDialog.visible = true
			get_tree().paused = true

# ALGORITHM BEGINS HERE

func paint_points() -> void:
```

```
	for point in points:
		set_cell(0, Vector2(point["x"], point["y"]), 0, point["type"])
		for citizen in point["citizens"]:
			if _is_in_bounds(point["x"], citizen["dx"], point["y"], citizen["dy"]):
				set_cell(0, Vector2(point["x"] + citizen["dx"], point["y"] + citizen["dy"]), 0, point["type"])

func _is_in_bounds(x: int, dx: int, y: int, dy: int) -> bool:
	return x + dx >= 0 and x + dx < x_tile_range and y + dy >= 0 and y + dy < y_tile_range

func _squared(x: int) -> int:
	return x ** 2

func calculate_points_delta(x: int, y: int, p: int) -> float:
	if distance == EUCLIDEAN:
		return sqrt(_squared(points[p]["x"] - x) + _squared(points[p]["y"] - y))
	return abs(points[p]["x"] - x) + abs(points[p]["y"] - y)

func define_points(num_points: int) -> void:
	var types: Array[Vector2i] = trees.duplicate()
	types.append_array(buildings)
	for i in range(num_points):
		var x: int = randi_range(0, x_tile_range)
		var y: int = randi_range(0, y_tile_range)
		var type: Vector2i = types.pick_random()
		types.erase(type)
		points.append(
			{
				"type": type,
```

```
                "x": x,
                "y": y,
                "citizens": []
            }
        )
    for x in range(x_tile_range):
        for y in range(y_tile_range):
            var lowest_delta: Dictionary = {
                "point_id": 0,
                "delta": x_tile_range * y_tile_range
            }
            for p in range(len(points)):
                var delta: float = calculate_points_delta(x, y, p)
                if delta < lowest_delta["delta"]:
                    lowest_delta = {
                        "point_id": p,
                        "delta": delta
                    }
                var active_point: Dictionary = points[lowest_delta["
                    point_id"]]
                var dx: int = x - active_point["x"]
                var dy: int = y - active_point["y"]
                active_point["citizens"].append(
                    {
                        "dx": dx,
                        "dy": dy
                    }
                )
```

C.4.6 accept_dialog.tscn

```
[gd_scene format=3 uid="uid://cau5jgogdnf53"]
```

```

[node name="AcceptDialog" type="AcceptDialog"]
title = "Tree-Munching Time!"
position = Vector2i(326, 100)
size = Vector2i(500, 421)
mouse_passthrough = true
ok_button_text = "Bring it on!"
dialog_text = "You're a hollow Golem who seeks the ultimate
    treasure; a ring that's got something on top of it. It's
    somewhere in this large village and barely visible to your
    naked eyes, but you'll stop at nothing to get what you want.
    You can chow down every tree and fauna that stands in your way
    of the ring, but your Achilles heel is any bricks and mortar,
    which will make you stop at your tracks. Are you ready to
    attain your treasure?w Golem in a black-and-white world, in
    search for your most desired treasure. It's a ring with
    something on top of it. And you'll stop at nothing to get what
    you want. You can chow down every tree and fauna that stands in
     your way of the ring, but your Achilles heel is any bricks and
     mortar, which will make you stop at your tracks. Are you ready
     to attain the treasure that is rightfully yours?!"
dialog_autowrap = true
```

C.4.7 win_dialog.tscn

```
[gd_scene format=3 uid="uid://b5q8ovcigrvyr"]

[node name="WinDialog" type="ConfirmationDialog"]
title = "You Found the Treasure!"
position = Vector2i(326, 100)
size = Vector2i(500, 421)
mouse_passthrough = true
```

```
ok_button_text = "Get Me a New Village"
dialog_text = "You found your treasure! Well done, you!

Would you like to travel to a new village in the hopes of finding
    another ring? Or would you like to take your treasure home now?
    "
dialog_autowrap = true
cancel_button_text = "Get Me Out of Here"
```

C.4.8 icon.svg.import

```
[remap]

importer="texture"
type="CompressedTexture2D"
uid="uid://uotfe6soknht"
path="res://.godot/imported/icon.svg-218
    a8f2b3041327d8a5756f3a245f83b.ctex"
metadata={
"vram_texture": false
}

[deps]

source_file="res://icon.svg"
dest_files=["res://.godot/imported/icon.svg-218
    a8f2b3041327d8a5756f3a245f83b.ctex"]

[params]

compress/mode=0
compress/high_quality=false
```

```
compress/lossy_quality=0.7
compress/hdr_compression=1
compress/normal_map=0
compress/channel_pack=0
mipmaps/generate=false
mipmaps/limit=-1
roughness/mode=0
roughness/src_normal=""
process/fix_alpha_border=true
process/premult_alpha=false
process/normal_map_invert_y=false
process/hdr_as_srgb=false
process/hdr_clamp_exposure=false
process/size_limit=0
detect_3d/compress_to=1
svg/scale=1.0
editor/scale_with_editor_scale=false
editor/convert_colors_with_editor_theme=false
```

C.4.9 monochrome_packed.png.import

```
[remap]

importer="texture"
type="CompressedTexture2D"
uid="uid://c3bpsm4r8t504"
path="res://.godot/imported/monochrome_packed.png-6
    b9bd1c64dd50f72acd3afd14d1ac34f.ctex"
metadata={
"vram_texture": false
}

```

```
[deps]

source_file="res://monochrome_packed.png"
dest_files=["res://.godot/imported/monochrome_packed.png-6
    b9bd1c64dd50f72acd3afd14d1ac34f.ctex"]

[params]

compress/mode=0
compress/high_quality=false
compress/lossy_quality=0.7
compress/hdr_compression=1
compress/normal_map=0
compress/channel_pack=0
mipmaps/generate=false
mipmaps/limit=-1
roughness/mode=0
roughness/src_normal=""
process/fix_alpha_border=true
process/premult_alpha=false
process/normal_map_invert_y=false
process/hdr_as_srgb=false
process/hdr_clamp_exposure=false
process/size_limit=0
detect_3d/compress_to=1
```

C.5 Noise Demo

C.5.1 .gitattributes

```
# Normalize EOL for all files that Git considers text files.
* text=auto eol=lf
```

C.5.2 .gitignore

```
# Godot 4+ specific ignores
.godot/
```

C.5.3 project.godot

```
; Engine configuration file.
; It's best edited using the editor UI and not directly,
; since the parameters that go here are not all obvious.
;
; Format:
;   [section] ; section goes between []
;   param=value ; assign values to parameters

config_version=5

[application]

config/name="Noise Demo"
run/main_scene="res://tile_map.tscn"
config/features=PackedStringArray("4.0", "Forward Plus")
config/icon="res://icon.svg"

[display]

window/size/viewport_height=640

[rendering]

environment/defaults/default_clear_color=Color(0, 0, 0, 1)
```

C.5.4 tile_map.tscn

```
[gd_scene load_steps=7 format=3 uid="uid://d4jdcavluwx6s"]

[ext_resource type="Texture2D" uid="uid://m662wwd4prmn" path="res://monochrome_packed.png" id="1_ld7xx"]
[ext_resource type="Script" path="res://tile_map.gd" id="2_o1dn1"]
[ext_resource type="PackedScene" uid="uid://cau5jgogdnf53" path="res://accept_dialog.tscn" id="3_e0ur6"]
[ext_resource type="PackedScene" uid="uid://b5q8ovcigrvyr" path="res://win_dialog.tscn" id="4_ecfaa"]

[sub_resource type="TileSetAtlasSource" id="TileSetAtlasSource_1e80b"]
texture = ExtResource("1_ld7xx")
0:0/0 = 0
1:0/0 = 0
2:0/0 = 0
3:0/0 = 0
4:0/0 = 0
5:0/0 = 0
6:0/0 = 0
7:0/0 = 0
8:0/0 = 0
9:0/0 = 0
10:0/0 = 0
11:0/0 = 0
12:0/0 = 0
13:0/0 = 0
14:0/0 = 0
15:0/0 = 0
16:0/0 = 0
17:0/0 = 0
```

```
18:0/0 = 0
19:0/0 = 0
20:0/0 = 0
21:0/0 = 0
22:0/0 = 0
23:0/0 = 0
24:0/0 = 0
25:0/0 = 0
26:0/0 = 0
27:0/0 = 0
28:0/0 = 0
29:0/0 = 0
30:0/0 = 0
31:0/0 = 0
32:0/0 = 0
33:0/0 = 0
34:0/0 = 0
35:0/0 = 0
36:0/0 = 0
37:0/0 = 0
38:0/0 = 0
39:0/0 = 0
40:0/0 = 0
41:0/0 = 0
42:0/0 = 0
43:0/0 = 0
44:0/0 = 0
45:0/0 = 0
46:0/0 = 0
47:0/0 = 0
48:0/0 = 0
0:1/0 = 0
1:1/0 = 0
```

```
2:1/0 = 0
3:1/0 = 0
4:1/0 = 0
5:1/0 = 0
6:1/0 = 0
7:1/0 = 0
8:1/0 = 0
9:1/0 = 0
10:1/0 = 0
11:1/0 = 0
12:1/0 = 0
13:1/0 = 0
14:1/0 = 0
15:1/0 = 0
16:1/0 = 0
17:1/0 = 0
18:1/0 = 0
19:1/0 = 0
20:1/0 = 0
21:1/0 = 0
22:1/0 = 0
23:1/0 = 0
24:1/0 = 0
25:1/0 = 0
26:1/0 = 0
27:1/0 = 0
28:1/0 = 0
29:1/0 = 0
30:1/0 = 0
31:1/0 = 0
32:1/0 = 0
33:1/0 = 0
34:1/0 = 0
```

```
35:1/0 = 0
36:1/0 = 0
37:1/0 = 0
38:1/0 = 0
39:1/0 = 0
40:1/0 = 0
41:1/0 = 0
42:1/0 = 0
43:1/0 = 0
44:1/0 = 0
45:1/0 = 0
46:1/0 = 0
47:1/0 = 0
48:1/0 = 0
0:2/0 = 0
1:2/0 = 0
2:2/0 = 0
3:2/0 = 0
4:2/0 = 0
5:2/0 = 0
6:2/0 = 0
7:2/0 = 0
8:2/0 = 0
9:2/0 = 0
10:2/0 = 0
11:2/0 = 0
12:2/0 = 0
13:2/0 = 0
14:2/0 = 0
15:2/0 = 0
16:2/0 = 0
17:2/0 = 0
18:2/0 = 0
```

```
19:2/0 = 0
20:2/0 = 0
21:2/0 = 0
22:2/0 = 0
23:2/0 = 0
24:2/0 = 0
25:2/0 = 0
26:2/0 = 0
27:2/0 = 0
28:2/0 = 0
29:2/0 = 0
30:2/0 = 0
31:2/0 = 0
32:2/0 = 0
33:2/0 = 0
34:2/0 = 0
35:2/0 = 0
36:2/0 = 0
37:2/0 = 0
38:2/0 = 0
39:2/0 = 0
40:2/0 = 0
41:2/0 = 0
42:2/0 = 0
43:2/0 = 0
44:2/0 = 0
45:2/0 = 0
46:2/0 = 0
47:2/0 = 0
48:2/0 = 0
0:3/0 = 0
1:3/0 = 0
2:3/0 = 0
```

```
3:3/0 = 0
4:3/0 = 0
5:3/0 = 0
6:3/0 = 0
7:3/0 = 0
8:3/0 = 0
9:3/0 = 0
10:3/0 = 0
11:3/0 = 0
12:3/0 = 0
13:3/0 = 0
14:3/0 = 0
15:3/0 = 0
16:3/0 = 0
17:3/0 = 0
18:3/0 = 0
19:3/0 = 0
20:3/0 = 0
21:3/0 = 0
22:3/0 = 0
23:3/0 = 0
24:3/0 = 0
25:3/0 = 0
26:3/0 = 0
27:3/0 = 0
28:3/0 = 0
29:3/0 = 0
30:3/0 = 0
31:3/0 = 0
32:3/0 = 0
33:3/0 = 0
34:3/0 = 0
35:3/0 = 0
```

```
36:3/0 = 0
37:3/0 = 0
38:3/0 = 0
39:3/0 = 0
40:3/0 = 0
41:3/0 = 0
42:3/0 = 0
43:3/0 = 0
44:3/0 = 0
45:3/0 = 0
46:3/0 = 0
47:3/0 = 0
48:3/0 = 0
0:4/0 = 0
1:4/0 = 0
2:4/0 = 0
3:4/0 = 0
4:4/0 = 0
5:4/0 = 0
6:4/0 = 0
7:4/0 = 0
8:4/0 = 0
9:4/0 = 0
10:4/0 = 0
11:4/0 = 0
12:4/0 = 0
13:4/0 = 0
14:4/0 = 0
15:4/0 = 0
16:4/0 = 0
17:4/0 = 0
18:4/0 = 0
19:4/0 = 0
```

```
20:4/0 = 0
21:4/0 = 0
22:4/0 = 0
23:4/0 = 0
24:4/0 = 0
25:4/0 = 0
26:4/0 = 0
27:4/0 = 0
28:4/0 = 0
29:4/0 = 0
30:4/0 = 0
31:4/0 = 0
32:4/0 = 0
33:4/0 = 0
34:4/0 = 0
35:4/0 = 0
36:4/0 = 0
37:4/0 = 0
38:4/0 = 0
39:4/0 = 0
40:4/0 = 0
41:4/0 = 0
42:4/0 = 0
43:4/0 = 0
44:4/0 = 0
45:4/0 = 0
46:4/0 = 0
47:4/0 = 0
48:4/0 = 0
0:5/0 = 0
1:5/0 = 0
2:5/0 = 0
3:5/0 = 0
```

```
4:5/0 = 0
5:5/0 = 0
6:5/0 = 0
7:5/0 = 0
8:5/0 = 0
9:5/0 = 0
10:5/0 = 0
11:5/0 = 0
12:5/0 = 0
13:5/0 = 0
14:5/0 = 0
15:5/0 = 0
16:5/0 = 0
17:5/0 = 0
18:5/0 = 0
19:5/0 = 0
20:5/0 = 0
21:5/0 = 0
22:5/0 = 0
23:5/0 = 0
24:5/0 = 0
25:5/0 = 0
26:5/0 = 0
27:5/0 = 0
28:5/0 = 0
29:5/0 = 0
30:5/0 = 0
31:5/0 = 0
32:5/0 = 0
33:5/0 = 0
34:5/0 = 0
35:5/0 = 0
36:5/0 = 0
```

```
37:5/0 = 0
38:5/0 = 0
39:5/0 = 0
40:5/0 = 0
41:5/0 = 0
42:5/0 = 0
43:5/0 = 0
44:5/0 = 0
45:5/0 = 0
46:5/0 = 0
47:5/0 = 0
48:5/0 = 0
0:6/0 = 0
1:6/0 = 0
2:6/0 = 0
3:6/0 = 0
4:6/0 = 0
5:6/0 = 0
6:6/0 = 0
7:6/0 = 0
8:6/0 = 0
9:6/0 = 0
10:6/0 = 0
11:6/0 = 0
12:6/0 = 0
13:6/0 = 0
14:6/0 = 0
15:6/0 = 0
16:6/0 = 0
17:6/0 = 0
18:6/0 = 0
19:6/0 = 0
20:6/0 = 0
```

```
21:6/0 = 0
22:6/0 = 0
23:6/0 = 0
24:6/0 = 0
25:6/0 = 0
26:6/0 = 0
27:6/0 = 0
28:6/0 = 0
29:6/0 = 0
30:6/0 = 0
31:6/0 = 0
32:6/0 = 0
33:6/0 = 0
34:6/0 = 0
35:6/0 = 0
36:6/0 = 0
37:6/0 = 0
38:6/0 = 0
39:6/0 = 0
40:6/0 = 0
41:6/0 = 0
42:6/0 = 0
43:6/0 = 0
44:6/0 = 0
45:6/0 = 0
46:6/0 = 0
47:6/0 = 0
48:6/0 = 0
0:7/0 = 0
1:7/0 = 0
2:7/0 = 0
3:7/0 = 0
4:7/0 = 0
```

```
5:7/0 = 0
6:7/0 = 0
7:7/0 = 0
8:7/0 = 0
9:7/0 = 0
10:7/0 = 0
11:7/0 = 0
12:7/0 = 0
13:7/0 = 0
14:7/0 = 0
15:7/0 = 0
16:7/0 = 0
17:7/0 = 0
18:7/0 = 0
19:7/0 = 0
20:7/0 = 0
21:7/0 = 0
22:7/0 = 0
23:7/0 = 0
24:7/0 = 0
25:7/0 = 0
26:7/0 = 0
27:7/0 = 0
28:7/0 = 0
29:7/0 = 0
30:7/0 = 0
31:7/0 = 0
32:7/0 = 0
33:7/0 = 0
34:7/0 = 0
35:7/0 = 0
36:7/0 = 0
37:7/0 = 0
```

```
38:7/0 = 0
39:7/0 = 0
40:7/0 = 0
41:7/0 = 0
42:7/0 = 0
43:7/0 = 0
44:7/0 = 0
45:7/0 = 0
46:7/0 = 0
47:7/0 = 0
48:7/0 = 0
0:8/0 = 0
1:8/0 = 0
2:8/0 = 0
3:8/0 = 0
4:8/0 = 0
5:8/0 = 0
6:8/0 = 0
7:8/0 = 0
8:8/0 = 0
9:8/0 = 0
10:8/0 = 0
11:8/0 = 0
12:8/0 = 0
13:8/0 = 0
14:8/0 = 0
15:8/0 = 0
16:8/0 = 0
17:8/0 = 0
18:8/0 = 0
19:8/0 = 0
20:8/0 = 0
21:8/0 = 0
```

```
22:8/0 = 0
23:8/0 = 0
24:8/0 = 0
25:8/0 = 0
26:8/0 = 0
27:8/0 = 0
28:8/0 = 0
29:8/0 = 0
30:8/0 = 0
31:8/0 = 0
32:8/0 = 0
33:8/0 = 0
34:8/0 = 0
35:8/0 = 0
36:8/0 = 0
37:8/0 = 0
38:8/0 = 0
39:8/0 = 0
40:8/0 = 0
41:8/0 = 0
42:8/0 = 0
43:8/0 = 0
44:8/0 = 0
45:8/0 = 0
46:8/0 = 0
47:8/0 = 0
48:8/0 = 0
0:9/0 = 0
1:9/0 = 0
2:9/0 = 0
3:9/0 = 0
4:9/0 = 0
5:9/0 = 0
```

```
6:9/0 = 0
7:9/0 = 0
8:9/0 = 0
9:9/0 = 0
10:9/0 = 0
11:9/0 = 0
12:9/0 = 0
13:9/0 = 0
14:9/0 = 0
15:9/0 = 0
16:9/0 = 0
17:9/0 = 0
18:9/0 = 0
19:9/0 = 0
20:9/0 = 0
21:9/0 = 0
22:9/0 = 0
23:9/0 = 0
24:9/0 = 0
25:9/0 = 0
26:9/0 = 0
27:9/0 = 0
28:9/0 = 0
29:9/0 = 0
30:9/0 = 0
31:9/0 = 0
32:9/0 = 0
33:9/0 = 0
34:9/0 = 0
35:9/0 = 0
36:9/0 = 0
37:9/0 = 0
38:9/0 = 0
```

```
39:9/0 = 0
40:9/0 = 0
41:9/0 = 0
42:9/0 = 0
43:9/0 = 0
44:9/0 = 0
45:9/0 = 0
46:9/0 = 0
47:9/0 = 0
48:9/0 = 0
0:10/0 = 0
1:10/0 = 0
2:10/0 = 0
3:10/0 = 0
4:10/0 = 0
5:10/0 = 0
6:10/0 = 0
7:10/0 = 0
8:10/0 = 0
9:10/0 = 0
10:10/0 = 0
11:10/0 = 0
12:10/0 = 0
13:10/0 = 0
14:10/0 = 0
15:10/0 = 0
16:10/0 = 0
17:10/0 = 0
18:10/0 = 0
19:10/0 = 0
20:10/0 = 0
21:10/0 = 0
22:10/0 = 0
```

```
23:10/0 = 0
24:10/0 = 0
25:10/0 = 0
26:10/0 = 0
27:10/0 = 0
28:10/0 = 0
29:10/0 = 0
30:10/0 = 0
31:10/0 = 0
32:10/0 = 0
33:10/0 = 0
34:10/0 = 0
35:10/0 = 0
36:10/0 = 0
37:10/0 = 0
38:10/0 = 0
39:10/0 = 0
40:10/0 = 0
41:10/0 = 0
42:10/0 = 0
43:10/0 = 0
44:10/0 = 0
45:10/0 = 0
46:10/0 = 0
47:10/0 = 0
48:10/0 = 0
0:11/0 = 0
1:11/0 = 0
2:11/0 = 0
3:11/0 = 0
4:11/0 = 0
5:11/0 = 0
6:11/0 = 0
```

```
7:11/0 = 0
8:11/0 = 0
9:11/0 = 0
10:11/0 = 0
11:11/0 = 0
12:11/0 = 0
13:11/0 = 0
14:11/0 = 0
15:11/0 = 0
16:11/0 = 0
17:11/0 = 0
18:11/0 = 0
19:11/0 = 0
20:11/0 = 0
21:11/0 = 0
22:11/0 = 0
23:11/0 = 0
24:11/0 = 0
25:11/0 = 0
26:11/0 = 0
27:11/0 = 0
28:11/0 = 0
29:11/0 = 0
30:11/0 = 0
31:11/0 = 0
32:11/0 = 0
33:11/0 = 0
34:11/0 = 0
35:11/0 = 0
36:11/0 = 0
37:11/0 = 0
38:11/0 = 0
39:11/0 = 0
```

```
40:11/0 = 0
41:11/0 = 0
42:11/0 = 0
43:11/0 = 0
44:11/0 = 0
45:11/0 = 0
46:11/0 = 0
47:11/0 = 0
48:11/0 = 0
0:12/0 = 0
1:12/0 = 0
2:12/0 = 0
3:12/0 = 0
4:12/0 = 0
5:12/0 = 0
6:12/0 = 0
7:12/0 = 0
8:12/0 = 0
9:12/0 = 0
10:12/0 = 0
11:12/0 = 0
12:12/0 = 0
13:12/0 = 0
14:12/0 = 0
15:12/0 = 0
16:12/0 = 0
17:12/0 = 0
18:12/0 = 0
19:12/0 = 0
20:12/0 = 0
21:12/0 = 0
22:12/0 = 0
23:12/0 = 0
```

```
24:12/0 = 0
25:12/0 = 0
26:12/0 = 0
27:12/0 = 0
28:12/0 = 0
29:12/0 = 0
30:12/0 = 0
31:12/0 = 0
32:12/0 = 0
33:12/0 = 0
34:12/0 = 0
35:12/0 = 0
36:12/0 = 0
37:12/0 = 0
38:12/0 = 0
39:12/0 = 0
40:12/0 = 0
41:12/0 = 0
42:12/0 = 0
43:12/0 = 0
44:12/0 = 0
45:12/0 = 0
46:12/0 = 0
47:12/0 = 0
48:12/0 = 0
0:13/0 = 0
1:13/0 = 0
2:13/0 = 0
3:13/0 = 0
4:13/0 = 0
5:13/0 = 0
6:13/0 = 0
7:13/0 = 0
```

```
8:13/0 = 0
9:13/0 = 0
10:13/0 = 0
11:13/0 = 0
12:13/0 = 0
13:13/0 = 0
14:13/0 = 0
15:13/0 = 0
16:13/0 = 0
17:13/0 = 0
18:13/0 = 0
19:13/0 = 0
20:13/0 = 0
21:13/0 = 0
22:13/0 = 0
23:13/0 = 0
24:13/0 = 0
25:13/0 = 0
26:13/0 = 0
27:13/0 = 0
28:13/0 = 0
29:13/0 = 0
30:13/0 = 0
31:13/0 = 0
32:13/0 = 0
33:13/0 = 0
34:13/0 = 0
35:13/0 = 0
36:13/0 = 0
37:13/0 = 0
38:13/0 = 0
39:13/0 = 0
40:13/0 = 0
```

```
41:13/0 = 0
42:13/0 = 0
43:13/0 = 0
44:13/0 = 0
45:13/0 = 0
46:13/0 = 0
47:13/0 = 0
48:13/0 = 0
0:14/0 = 0
1:14/0 = 0
2:14/0 = 0
3:14/0 = 0
4:14/0 = 0
5:14/0 = 0
6:14/0 = 0
7:14/0 = 0
8:14/0 = 0
9:14/0 = 0
10:14/0 = 0
11:14/0 = 0
12:14/0 = 0
13:14/0 = 0
14:14/0 = 0
15:14/0 = 0
16:14/0 = 0
17:14/0 = 0
18:14/0 = 0
19:14/0 = 0
20:14/0 = 0
21:14/0 = 0
22:14/0 = 0
23:14/0 = 0
24:14/0 = 0
```

```
25:14/0 = 0
26:14/0 = 0
27:14/0 = 0
28:14/0 = 0
29:14/0 = 0
30:14/0 = 0
31:14/0 = 0
32:14/0 = 0
33:14/0 = 0
34:14/0 = 0
35:14/0 = 0
36:14/0 = 0
37:14/0 = 0
38:14/0 = 0
39:14/0 = 0
40:14/0 = 0
41:14/0 = 0
42:14/0 = 0
43:14/0 = 0
44:14/0 = 0
45:14/0 = 0
46:14/0 = 0
47:14/0 = 0
48:14/0 = 0
0:15/0 = 0
1:15/0 = 0
2:15/0 = 0
3:15/0 = 0
4:15/0 = 0
5:15/0 = 0
6:15/0 = 0
7:15/0 = 0
8:15/0 = 0
```

```
9:15/0 = 0
10:15/0 = 0
11:15/0 = 0
12:15/0 = 0
13:15/0 = 0
14:15/0 = 0
15:15/0 = 0
16:15/0 = 0
17:15/0 = 0
18:15/0 = 0
19:15/0 = 0
20:15/0 = 0
21:15/0 = 0
22:15/0 = 0
23:15/0 = 0
24:15/0 = 0
25:15/0 = 0
26:15/0 = 0
27:15/0 = 0
28:15/0 = 0
29:15/0 = 0
30:15/0 = 0
31:15/0 = 0
32:15/0 = 0
33:15/0 = 0
34:15/0 = 0
35:15/0 = 0
36:15/0 = 0
37:15/0 = 0
38:15/0 = 0
39:15/0 = 0
40:15/0 = 0
41:15/0 = 0
```

```
42:15/0 = 0
43:15/0 = 0
44:15/0 = 0
45:15/0 = 0
46:15/0 = 0
47:15/0 = 0
48:15/0 = 0
0:16/0 = 0
1:16/0 = 0
2:16/0 = 0
3:16/0 = 0
4:16/0 = 0
5:16/0 = 0
6:16/0 = 0
7:16/0 = 0
8:16/0 = 0
9:16/0 = 0
10:16/0 = 0
11:16/0 = 0
12:16/0 = 0
13:16/0 = 0
14:16/0 = 0
15:16/0 = 0
16:16/0 = 0
17:16/0 = 0
18:16/0 = 0
19:16/0 = 0
20:16/0 = 0
21:16/0 = 0
22:16/0 = 0
23:16/0 = 0
24:16/0 = 0
25:16/0 = 0
```

```
26:16/0 = 0
27:16/0 = 0
28:16/0 = 0
29:16/0 = 0
30:16/0 = 0
31:16/0 = 0
32:16/0 = 0
33:16/0 = 0
34:16/0 = 0
35:16/0 = 0
36:16/0 = 0
37:16/0 = 0
38:16/0 = 0
39:16/0 = 0
40:16/0 = 0
41:16/0 = 0
42:16/0 = 0
43:16/0 = 0
44:16/0 = 0
45:16/0 = 0
46:16/0 = 0
47:16/0 = 0
48:16/0 = 0
0:17/0 = 0
1:17/0 = 0
2:17/0 = 0
3:17/0 = 0
4:17/0 = 0
5:17/0 = 0
6:17/0 = 0
7:17/0 = 0
8:17/0 = 0
9:17/0 = 0
```

```
10:17/0 = 0
11:17/0 = 0
12:17/0 = 0
13:17/0 = 0
14:17/0 = 0
15:17/0 = 0
16:17/0 = 0
17:17/0 = 0
18:17/0 = 0
19:17/0 = 0
20:17/0 = 0
21:17/0 = 0
22:17/0 = 0
23:17/0 = 0
24:17/0 = 0
25:17/0 = 0
26:17/0 = 0
27:17/0 = 0
28:17/0 = 0
29:17/0 = 0
30:17/0 = 0
31:17/0 = 0
32:17/0 = 0
33:17/0 = 0
34:17/0 = 0
35:17/0 = 0
36:17/0 = 0
37:17/0 = 0
38:17/0 = 0
39:17/0 = 0
40:17/0 = 0
41:17/0 = 0
42:17/0 = 0
```

```
43:17/0 = 0
44:17/0 = 0
45:17/0 = 0
46:17/0 = 0
47:17/0 = 0
48:17/0 = 0
0:18/0 = 0
1:18/0 = 0
2:18/0 = 0
3:18/0 = 0
4:18/0 = 0
5:18/0 = 0
6:18/0 = 0
7:18/0 = 0
8:18/0 = 0
9:18/0 = 0
10:18/0 = 0
11:18/0 = 0
12:18/0 = 0
13:18/0 = 0
14:18/0 = 0
15:18/0 = 0
16:18/0 = 0
17:18/0 = 0
18:18/0 = 0
19:18/0 = 0
20:18/0 = 0
21:18/0 = 0
22:18/0 = 0
23:18/0 = 0
24:18/0 = 0
25:18/0 = 0
26:18/0 = 0
```

```
27:18/0 = 0
28:18/0 = 0
29:18/0 = 0
30:18/0 = 0
31:18/0 = 0
32:18/0 = 0
33:18/0 = 0
34:18/0 = 0
35:18/0 = 0
36:18/0 = 0
37:18/0 = 0
38:18/0 = 0
39:18/0 = 0
40:18/0 = 0
41:18/0 = 0
42:18/0 = 0
43:18/0 = 0
44:18/0 = 0
45:18/0 = 0
46:18/0 = 0
47:18/0 = 0
48:18/0 = 0
0:19/0 = 0
1:19/0 = 0
2:19/0 = 0
3:19/0 = 0
4:19/0 = 0
5:19/0 = 0
6:19/0 = 0
7:19/0 = 0
8:19/0 = 0
9:19/0 = 0
10:19/0 = 0
```

```
11:19/0 = 0
12:19/0 = 0
13:19/0 = 0
14:19/0 = 0
15:19/0 = 0
16:19/0 = 0
17:19/0 = 0
18:19/0 = 0
19:19/0 = 0
20:19/0 = 0
21:19/0 = 0
22:19/0 = 0
23:19/0 = 0
24:19/0 = 0
25:19/0 = 0
26:19/0 = 0
27:19/0 = 0
28:19/0 = 0
29:19/0 = 0
30:19/0 = 0
31:19/0 = 0
32:19/0 = 0
33:19/0 = 0
34:19/0 = 0
35:19/0 = 0
36:19/0 = 0
37:19/0 = 0
38:19/0 = 0
39:19/0 = 0
40:19/0 = 0
41:19/0 = 0
42:19/0 = 0
43:19/0 = 0
```

```
44:19/0 = 0
45:19/0 = 0
46:19/0 = 0
47:19/0 = 0
48:19/0 = 0
0:20/0 = 0
1:20/0 = 0
2:20/0 = 0
3:20/0 = 0
4:20/0 = 0
5:20/0 = 0
6:20/0 = 0
7:20/0 = 0
8:20/0 = 0
9:20/0 = 0
10:20/0 = 0
11:20/0 = 0
12:20/0 = 0
13:20/0 = 0
14:20/0 = 0
15:20/0 = 0
16:20/0 = 0
17:20/0 = 0
18:20/0 = 0
19:20/0 = 0
20:20/0 = 0
21:20/0 = 0
22:20/0 = 0
23:20/0 = 0
24:20/0 = 0
25:20/0 = 0
26:20/0 = 0
27:20/0 = 0
```

```
28:20/0 = 0
29:20/0 = 0
30:20/0 = 0
31:20/0 = 0
32:20/0 = 0
33:20/0 = 0
34:20/0 = 0
35:20/0 = 0
36:20/0 = 0
37:20/0 = 0
38:20/0 = 0
39:20/0 = 0
40:20/0 = 0
41:20/0 = 0
42:20/0 = 0
43:20/0 = 0
44:20/0 = 0
45:20/0 = 0
46:20/0 = 0
47:20/0 = 0
48:20/0 = 0
0:21/0 = 0
1:21/0 = 0
2:21/0 = 0
3:21/0 = 0
4:21/0 = 0
5:21/0 = 0
6:21/0 = 0
7:21/0 = 0
8:21/0 = 0
9:21/0 = 0
10:21/0 = 0
11:21/0 = 0
```

```
12:21/0 = 0
13:21/0 = 0
14:21/0 = 0
15:21/0 = 0
16:21/0 = 0
17:21/0 = 0
18:21/0 = 0
19:21/0 = 0
20:21/0 = 0
21:21/0 = 0
22:21/0 = 0
23:21/0 = 0
24:21/0 = 0
25:21/0 = 0
26:21/0 = 0
27:21/0 = 0
28:21/0 = 0
29:21/0 = 0
30:21/0 = 0
31:21/0 = 0
32:21/0 = 0
33:21/0 = 0
34:21/0 = 0
35:21/0 = 0
36:21/0 = 0
37:21/0 = 0
38:21/0 = 0
39:21/0 = 0
40:21/0 = 0
41:21/0 = 0
42:21/0 = 0
43:21/0 = 0
44:21/0 = 0
```

```
45:21/0 = 0
46:21/0 = 0
47:21/0 = 0
48:21/0 = 0

[sub_resource type="TileSet" id="TileSet_qtrb6"]
sources/0 = SubResource("TileSetAtlasSource_1e80b")

[node name="TileMap" type="TileMap"]
texture_filter = 1
tile_set = SubResource("TileSet_qtrb6")
format = 2
script = ExtResource("2_o1dn1")

[node name="AcceptDialog" parent="." instance=ExtResource("3_e0ur6"
    )]

[node name="WinDialog" parent="." instance=ExtResource("4_ecfaa")]
title = "You Found the Treasure!"
```

C.5.5 tile_map.gd

```
extends TileMap

const buildings: Array[Vector2i] = [
    Vector2i(0, 19),
    Vector2i(1, 19),
    Vector2i(2, 19),
    Vector2i(3, 19),
    Vector2i(4, 19),
    Vector2i(5, 19),
    Vector2i(6, 19),
```

```
        Vector2i(7, 19),
        Vector2i(8, 20),
        Vector2i(0, 20),
        Vector2i(1, 20),
        Vector2i(2, 20),
        Vector2i(3, 20),
        Vector2i(4, 20),
        Vector2i(5, 20),
        Vector2i(6, 20),
        Vector2i(7, 20),
        Vector2i(8, 20),
        Vector2i(0, 21),
        Vector2i(1, 21),
        Vector2i(2, 21),
        Vector2i(3, 21),
        Vector2i(4, 21),
        Vector2i(5, 21),
        Vector2i(6, 21),
        Vector2i(7, 21),
        Vector2i(8, 21)
    ]
    const trees: Array[Vector2i] = [
        Vector2i(0,1),
        Vector2i(1,1),
        Vector2i(2,1),
        Vector2i(3,1),
        Vector2i(4,1),
        Vector2i(5,1),
        Vector2i(6,1),
        Vector2i(7,1),
        Vector2i(0,2),
        Vector2i(1,2),
        Vector2i(2,2),
```

```
        Vector2i(3,2),
        Vector2i(4,2)
    ]
    const PLAYER_SPRITE: Vector2i = Vector2i(24, 7)
    var player_placement_cell: Vector2i
    const rings: Array[Vector2i] = [
        Vector2i(43, 6),
        Vector2i(44, 6),
        Vector2i(45, 6),
        Vector2i(46, 6)
    ]
    var ring_placement_cell: Vector2i

    var noise: FastNoiseLite
    @export_enum("Perlin", "Simplex", "Simplex Smooth", "Value", "Value
         Cubic") var noise_type: String = "Simplex Smooth"
    @export var fractal_type: FastNoiseLite.FractalType
    @export var cellular_distance_type: FastNoiseLite.
        CellularDistanceFunction
    #@export_range(1, 10, 1) var octaves: int = 5
    @export_range(0.0, 1.0) var noise_frequency: float = 0.894

    #@onready var timer: Timer = Timer.new()
    #@export_range(10, 200, 10) var player_movement_speed: int = 100
    @export_range(-1.0, 1.0) var tree_cap: float = -0.048
    @export_range(-1.0, 1.0) var building_cap: float = -0.252
    @export_range(0.0, 0.5) var building_overtakes_tree: float = 0.12
    var x_tile_range: int = ProjectSettings.get_setting("display/window
        /size/viewport_width") / tile_set.tile_size.x
    var y_tile_range: int = ProjectSettings.get_setting("display/window
        /size/viewport_height") / tile_set.tile_size.y

    # Called when the node enters the scene tree for the first time.
```

```
func _ready() -> void:
	randomize()
	var start_time: float = Time.get_ticks_msec()
	set_noise()
	paint_tiles()
	place_player()
	place_ring()
	var new_time: float = Time.get_ticks_msec() - start_time
	print("Time taken: " + str(new_time) + "ms")
	$AcceptDialog.dialog_text = "You're a hollow Golem who seeks the ultimate treasure; a ring that's got something on top of it. It's somewhere in this large village and barely visible to your naked eyes, which took us " + str(new_time) + " milliseconds to generate (" + str(new_time / 1000.0) + " seconds), but you'll stop at nothing to get what you want. You can chow down every tree and fauna that stands in your way of the ring, but your Achilles heel is any bricks and mortar, which WILL make you stop at your tracks. Since it's easy to get lost in here, we'll tell you that you're in position " + str(player_placement_cell) + " in this big village of size " + str(Vector2i(x_tile_range, y_tile_range)) + ". However, it is YOUR job to find the ring, so are you ready to attain the treasure that is rightfully yours?!"
	$AcceptDialog.visible = true
	$AcceptDialog.confirmed.connect(_on_AcceptDialog_closed)
	$AcceptDialog.canceled.connect(_on_AcceptDialog_closed)
	$WinDialog.confirmed.connect(_on_WinDialog_confirmed)
	$WinDialog.canceled.connect(_on_WinDialog_canceled)
	get_tree().paused = true

func _on_WinDialog_confirmed() -> void:
	get_tree().reload_current_scene()

```

```
func _on_WinDialog_canceled() -> void:
    get_tree().quit()

func _on_AcceptDialog_closed() -> void:
    $AcceptDialog.visible = false
    get_tree().paused = false

func _get_random_placement_cell() -> Vector2i:
    return Vector2i(randi() % x_tile_range, randi() % y_tile_range)

func place_player() -> void:
    player_placement_cell = _get_random_placement_cell()
    while get_used_cells(0).has(player_placement_cell):
        player_placement_cell = _get_random_placement_cell()
    set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)

func place_ring() -> void:
    ring_placement_cell = _get_random_placement_cell()
    while get_used_cells(0).has(ring_placement_cell):
        ring_placement_cell = _get_random_placement_cell()
    set_cell(0, ring_placement_cell, 0, rings.pick_random())

func _is_not_out_of_bounds(cell: Vector2i) -> bool:
    return cell.x >= 0 and cell.x < x_tile_range and cell.y >= 0 and cell.y < y_tile_range

func _physics_process(_delta: float) -> void:
    var previous_cell: Vector2i = player_placement_cell
    var direction: Vector2i = Vector2i.ZERO
    if Input.is_action_pressed("ui_up"): direction = Vector2i.UP
    elif Input.is_action_pressed("ui_down"): direction = Vector2i.DOWN
    elif Input.is_action_pressed("ui_left"): direction = Vector2i.
```

```
LEFT
	elif Input.is_action_pressed("ui_right"): direction = Vector2i.RIGHT
	var new_placement_cell: Vector2i = player_placement_cell + direction
	if (not get_used_cells(0).has(new_placement_cell) or trees.has(get_cell_atlas_coords(0, new_placement_cell)) or new_placement_cell == ring_placement_cell) and _is_not_out_of_bounds(new_placement_cell):
		player_placement_cell = new_placement_cell
		set_cell(0, previous_cell, 0) # deletes contents of previous cell (atlas_coords = Vector2i(-1, -1))
		set_cell(0, player_placement_cell, 0, PLAYER_SPRITE)
		if player_placement_cell == ring_placement_cell:
			$WinDialog.visible = true
			get_tree().paused = true

# ALGORITHM BEGINS HERE

func _get_noise_type() -> int:
	match noise_type:
		"Perlin": return 3
		"Simplex": return 0
		"Value": return 5
		"Value Cubic": return 4
		_: return 1 # Return Simplex Smooth by default

func set_noise() -> void:
	noise = FastNoiseLite.new()
	noise.frequency = noise_frequency
	noise.noise_type = _get_noise_type() as FastNoiseLite.NoiseType
	noise.fractal_type = fractal_type
	noise.cellular_distance_function = cellular_distance_type
```

```
#	noise.fractal_octaves = octaves
	noise.seed = randi()

func paint_tiles() -> void:
	for x in range(x_tile_range):
		for y in range(y_tile_range):
			var noise_point: float = noise.get_noise_2d(x * tile_set.tile_size.x, y * tile_set.tile_size.y)
			if noise_point < tree_cap and not get_used_cells(0).has(Vector2i(x, y)):
				set_cell(0, Vector2i(x, y), 0, trees.pick_random())
			if ((building_cap <= tree_cap and randf() < building_overtakes_tree) or (building_cap > tree_cap and noise_point < building_cap)) and not get_used_cells(0).has(Vector2i(x, y)):
				set_cell(0, Vector2i(x, y), 0, buildings.pick_random())
```

C.5.6 accept_dialog.tscn

```
[gd_scene format=3 uid="uid://cau5jgogdnf53"]

[node name="AcceptDialog" type="AcceptDialog"]
title = "Tree-Munching Time!"
position = Vector2i(326, 100)
size = Vector2i(500, 421)
mouse_passthrough = true
ok_button_text = "Bring it on!"
dialog_text = "You're a hollow Golem who seeks the ultimate treasure; a ring that's got something on top of it. It's somewhere in this large village and barely visible to your naked eyes, but you'll stop at nothing to get what you want. You can chow down every tree and fauna that stands in your way
```

```
    of the ring, but your Achilles heel is any bricks and mortar,
    which will make you stop at your tracks. Are you ready to
    attain your treasure?w Golem in a black-and-white world, in
    search for your most desired treasure. It's a ring with
    something on top of it. And you'll stop at nothing to get what
    you want. You can chow down every tree and fauna that stands in
     your way of the ring, but your Achilles heel is any bricks and
     mortar, which will make you stop at your tracks. Are you ready
     to attain the treasure that is rightfully yours?!"
dialog_autowrap = true
```

C.5.7 win_dialog.tscn

```
[gd_scene format=3 uid="uid://b5q8ovcigrvyr"]

[node name="WinDialog" type="ConfirmationDialog"]
title = "Tree-Munching Time!"
position = Vector2i(326, 100)
size = Vector2i(500, 421)
mouse_passthrough = true
ok_button_text = "Get Me a New Village"
dialog_text = "You found your treasure! Well done, you!

Would you like to travel to a new village in the hopes of finding
    another ring? Or would you like to take your treasure home now?
    "
dialog_autowrap = true
cancel_button_text = "Get Me Out of Here"
```

C.5.8 icon.svg.import

```
[remap]

importer="texture"
type="CompressedTexture2D"
uid="uid://crgf6ascxsdt0"
path="res://.godot/imported/icon.svg-218
    a8f2b3041327d8a5756f3a245f83b.ctex"
metadata={
"vram_texture": false
}

[deps]

source_file="res://icon.svg"
dest_files=["res://.godot/imported/icon.svg-218
    a8f2b3041327d8a5756f3a245f83b.ctex"]

[params]

compress/mode=0
compress/high_quality=false
compress/lossy_quality=0.7
compress/hdr_compression=1
compress/normal_map=0
compress/channel_pack=0
mipmaps/generate=false
mipmaps/limit=-1
roughness/mode=0
roughness/src_normal=""
process/fix_alpha_border=true
process/premult_alpha=false
process/normal_map_invert_y=false
process/hdr_as_srgb=false
```

```
process/hdr_clamp_exposure=false
process/size_limit=0
detect_3d/compress_to=1
svg/scale=1.0
editor/scale_with_editor_scale=false
editor/convert_colors_with_editor_theme=false
```

C.5.9 monochrome_packed.png.import

```
[remap]

importer="texture"
type="CompressedTexture2D"
uid="uid://m662wwd4prmn"
path="res://.godot/imported/monochrome_packed.png-6
    b9bd1c64dd50f72acd3afd14d1ac34f.ctex"
metadata={
"vram_texture": false
}

[deps]

source_file="res://monochrome_packed.png"
dest_files=["res://.godot/imported/monochrome_packed.png-6
    b9bd1c64dd50f72acd3afd14d1ac34f.ctex"]

[params]

compress/mode=0
compress/high_quality=false
compress/lossy_quality=0.7
compress/hdr_compression=1
```

```
compress/normal_map=0
compress/channel_pack=0
mipmaps/generate=false
mipmaps/limit=-1
roughness/mode=0
roughness/src_normal=""
process/fix_alpha_border=true
process/premult_alpha=false
process/normal_map_invert_y=false
process/hdr_as_srgb=false
process/hdr_clamp_exposure=false
process/size_limit=0
detect_3d/compress_to=1
```

C.6 LSystemGrammarDemo

C.6.1 .gitattributes

```
# Normalize EOL for all files that Git considers text files.
* text=auto eol=lf
```

C.6.2 .gitignore

```
# Godot 4+ specific ignores
.godot/
```

C.6.3 project.godot

```
; Engine configuration file.
```

```
; It's best edited using the editor UI and not directly,
; since the parameters that go here are not all obvious.
;
; Format:
;   [section] ; section goes between []
;   param=value ; assign values to parameters

config_version=5

[application]

config/name="LSystemGrammarDemo"
run/main_scene="res://DemoNode.tscn"
config/features=PackedStringArray("4.0")

[display]

window/stretch/mode="canvas_items"
window/stretch/aspect="expand"

[gui]

common/drop_mouse_on_gui_input_disabled=true

[physics]

common/enable_pause_aware_picking=true
```

C.6.4 DemoNode.tscn

```
[gd_scene load_steps=2 format=3 uid="uid://bu380we4od0ln"]

```

```
[ext_resource type="Script" path="res://DemoNode.gd" id="1"]

[node name="DemoNode" type="Node"]
script = ExtResource("1")
choices = "deterministic"

[node name="Timer" type="Timer" parent="."]

[node name="TextLabel" type="Label" parent="."]
offset_right = 1152.0
offset_bottom = 23.0
autowrap_mode = 3

[connection signal="timeout" from="Timer" to="." method="_on_Timer_timeout"]
```

C.6.5 DemoNode.gd

```
extends Node

# Basic: https://youtu.be/feNVBEPXAcE?t=77 (L = +)
# Choices: http://paulbourke.net/fractals/lsys/
# Deterministic: https://www1.biologie.uni-hamburg.de/b-online/e28_3/lsys.html#D0L-system

@export_enum("basic", "choices", "deterministic") var choices: String = "choices"
@export var axiom: String
@onready var string: String
@onready var timer = $Timer
@onready var label = $TextLabel
@onready var rules: Array[Dictionary]
```

```

func set_values():
    if choices == "basic":
        rules = [
            {
                "from": "F",
                "to": "F+F"
            }
        ]
        axiom = "F+"
    elif choices == "choices":
        rules = [
            {
                "from": "F",
                "to": "F+--FFFF+F+-FF"
            }
        ]
        axiom = "F+F+F+F"
    elif choices == "deterministic":
        rules = [
            {
                "from": "a",
                "to": "ab"
            },
            {
                "from": "b",
                "to": "a"
            }
        ]
        axiom = "b"

func _ready():
    set_values()
```

```
    string = axiom
    label.size.x = get_viewport().size.x
    label.text = string
    print(len(string))
    timer.start()

func get_new_replacement(character: String) -> String:
    for rule in rules:
        if rule["from"] == character:
            return rule["to"]
    return ""

func _on_Timer_timeout():
    var new_string = ""
    for character in string:
        new_string += get_new_replacement(character)
    string = new_string
    label.text = string
    print(len(string))
```

C.6.6 icon.svg.import

```
[remap]

importer="texture"
type="CompressedTexture2D"
uid="uid://cwnnuqmejj04q"
path="res://.godot/imported/icon.svg-218
    a8f2b3041327d8a5756f3a245f83b.ctex"
metadata={
"vram_texture": false
}
```

```
[deps]

source_file="res://icon.svg"
dest_files=["res://.godot/imported/icon.svg-218a8f2b3041327d8a5756f3a245f83b.ctex"]

[params]

compress/mode=0
compress/high_quality=false
compress/lossy_quality=0.7
compress/hdr_compression=1
compress/normal_map=0
compress/channel_pack=0
mipmaps/generate=false
mipmaps/limit=-1
roughness/mode=0
roughness/src_normal=""
process/fix_alpha_border=true
process/premult_alpha=false
process/normal_map_invert_y=false
process/hdr_as_srgb=false
process/hdr_clamp_exposure=false
process/size_limit=0
detect_3d/compress_to=1
svg/scale=1.0
editor/scale_with_editor_scale=false
editor/convert_colors_with_editor_theme=false
```