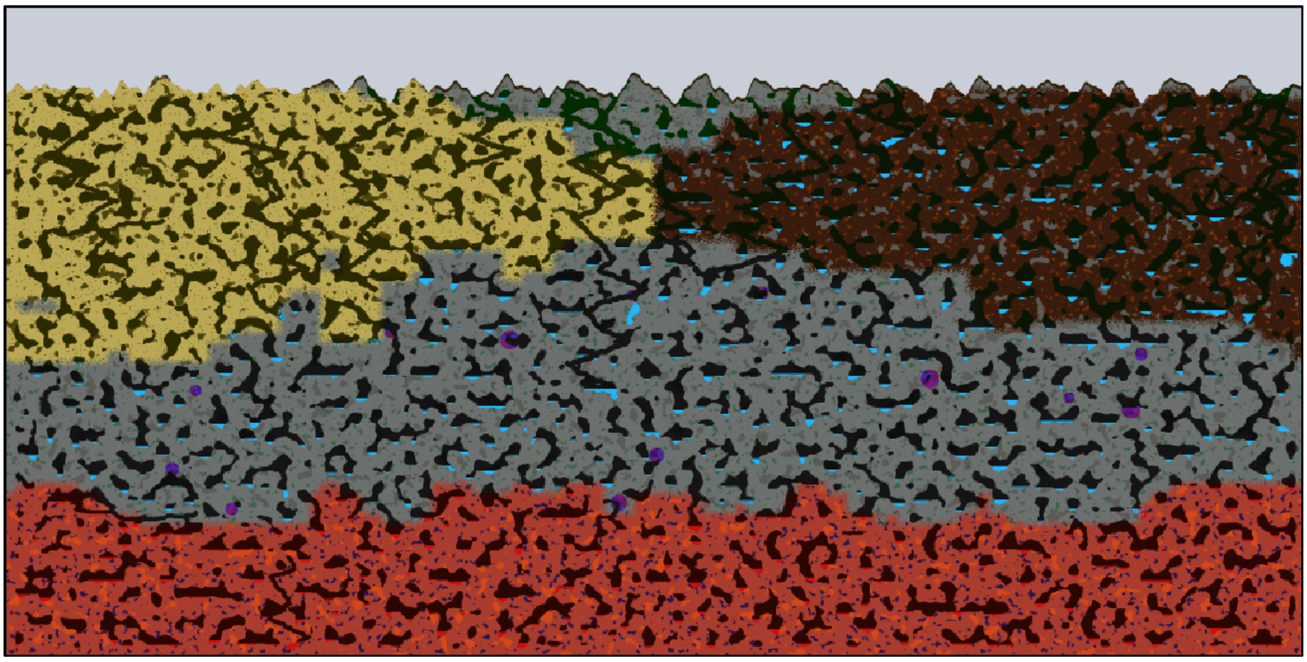
# Comprehensive Creative Technologies Project: Player-Driven Procedural World Generation for 2D Based Games

**Oscar Wilkinson**
Oscar2.Wilkinson@live.uwe.ac.uk
Supervisor: Joshua Hompstead

**Department of Computing and Creative Technology**
University of the West of England
Coldharbour Lane
Bristol BS16 1QY



**Abstract**

This report outlines the implementation of a system that improves upon the idea of procedurally generated maps for 2D video game by involving player driven creative input. This concept allows players to play with maps that cater towards their preferences without the constraint of pre-determined rules in most sandbox games. The system is within a downloadable Unity project which can be altered by developers to restrain or expand the limits of the player's control. The underlying report covers the layers of techniques used such as Perline noise and Wave Function Collapse to aptly convert player creativity into processes.

**Keywords:** Procedural Content Generation, Perlin Noise, Wave Function Collapse, 2D Video Games

## How to access the project

GitHub Link and Wiki: https://github.com/Oscar-Wilko/Player-Driven-Procedural-Generation
Video Showcase: https://www.youtube.com/watch?v=la9h4L493ao
EXE Download: https://uweacuk-my.sharepoint.com/:u:/g/personal/oscar2_wilkinson_live_uwe_ac_uk/ERhl0jFQhp5EiON0DQdmO6wBu642erlMvHiPbVixp8tr-Q?e=TrL8AR

## 1. Introduction

Procedural Content Generation (PCG) tools and systems are constructive processes that create randomised and continuous results with limited or indirect user input (Togelius, Shaker and Nelson, 2016), allowing for repeatable usage. A lot of videogames, specifically sandbox games, utilise these to produce worlds and maps that are different for every playthrough.

These procedural worlds offer players unique experiences, yet their replay ability often depends on the limitations imposed by developers' predefined generation methods. Allowing players to influence world creation could significantly enhance their enjoyment by diversifying the world, increasing player immersion, and expanding creativity.

This project will explore these generative tools to create a system that can convert a players creative input into a large scale map that encompasses their intended design. This will assist in tackling the boundaries of developer-dictated environments. The output will act like the terrain generation in the video game Terraria (Re-Logic 2011), a 2D side-view sandbox video game scenario. This will require a culmination of multiple PCG techniques such as Perlin Noise and Wave Function Collapse while exploring their uses.

### 1.1 Project Objectives

- Develop generative processes that allow for the system to create unique and interesting maps.
- Create an interactive field for the player to alter all the inputs and variables behind the generative process.
- Allow for the developer to limit the range of creative freedom for suiting environments / scenarios.

### 1.2 Key Deliverables

- Both .EXE of scene showcasing the system, and a Unity project containing all the mentioned systems/tools.
- Produce a set of procedurally generative algorithms such as Wave Function Collapse, Perlin noise, Worley noise, and Voronoi diagram.
- Implement an interactive canvas that allow the player to draw the intended location and spread of biomes.
- Develop an interface that allows the player to alter the inputs and variables behind the generative process.
- Implement a file saving system to allow for the user to save certain steps of the process and reload it at any time.

## 2. Research questions

To focus the functionality and scope of the project, the following questions are considered:

*"How can the player's creative input be calculated and utilised in a procedural scenario?"* Values and variables dictate how worlds are procedural generated and altered. Converting the creative input into these values must be accurate enough to maintain the player's ideal generation.

*"What is the extent of player control that also maintains unique and emergent procedural world generation?"* Procedural content generation can lack in control when considering design processes, so it is crucial to prioritise control over the generation processes to allow for player input.

*"What are the expected world generation elements to indicate usage in all scenarios?"* To effectively display the utility of this system, enough unique scenarios must be present to show the wide range of application this can apply to.

## 3. Literature review

### 3.1 Wave Function Collapse & Rulesets

In its principal Wave Function Collapse (WFC), refers to an algorithm used in procedural scenarios that generate maps of cells. There are many applications of it for various fields, including texture generation, pattern generation, and map generation. It involves the simulation of an environment, typically a two-dimensional grid of cells, that contain multiple possible states per cell which gradually collapse into single results based on conditions surrounding them or globally affecting them (Cheng, Han and Fei, 2020). The WFC ruleset is a system that governs the behaviour of the cells and determines the conditions in which collapsing can occur and what cells are limited to what they become. The rulesets contain logic that ensures consistent, reliable results and are more often curated by the developer as it maintains said consistency for their chosen purposes.

The WFC process in detail is a cycle of steps that start with a blank environment of all possibilities and result into an environment in which every cell is isolated to one state or condition.

An example can be described as such.
In a 2D grid based environment of a given size there are types of tiles: grass, dirt, water, and stone. For a visual demonstration each type is associated with a colour: green, brown, blue, and grey. The grid starts with every tile having every possible state available, meaning the map is given the possibility of being anything. However, this map is controlled by rules that state what tile types can neighbour other tile types. Some rules are clarified here: water cannot be above or below a grass tile, grass cannot be to the left or right of a stone tile, dirt cannot neighbour a water tile.

Now that there is a beginning and there are rules governing the system, a cycle of processes can commence. First, find the cell with the lowest entropy, meaning the cell with the lowest number of possibilities to what it can become. If there are multiple cells with the lowest entropy, then it will pick a random one from all that share the lowest entropy. This cell gets collapsed into one tile type based on the possible types it can become. E.g. a tile that can be any tile, will randomly select between grass, dirt, water, or stone. Meaning this tile is locked into being this state and cannot change.

Due to a cell collapsing, it updates surrounding tiles and limits their possible states based on the ruleset earlier created. In this example, if a cell collapses and isolates to a grass tile, then the cells to the left and right cannot become stone tiles, and the cells above and below it cannot become a water tile, removing those tile types from its list of possible states.

The entropy search, collapsing, and cell updating repeat until either all cells have successfully collapsed or until a cell attempts to collapse and has no possible states that follow the ruleset. In the case that the collapse fails as there are no possible types to become, a few possibilities arise which can be decided upon based on the needs of the system. One solution is to use its last known possibilities before it was limited to none, this can produce irregular generations that go against the defined rules but is the fastest method. Another possibility is to revert a given number of cycles and reattempt the generation in a different way to avoid reaching the cell that had zero states, this has a guaranteed level of reliability with the rules but can encounter scenarios that constantly loop with no possible solution. There are many more methods that adhere to this issue, but the appropriate use depends on the scope of the system.
With all these steps into account, **Fig 01** shows the generative process step by step and finalises on the generation of **Fig 02**.

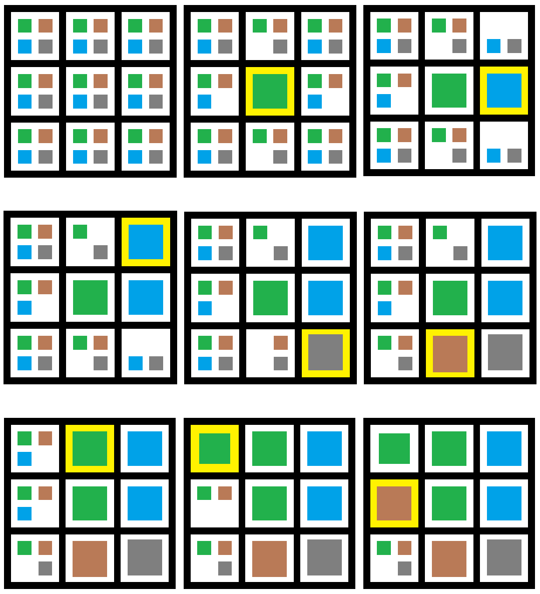


**Fig 01:** WFC process step by step.

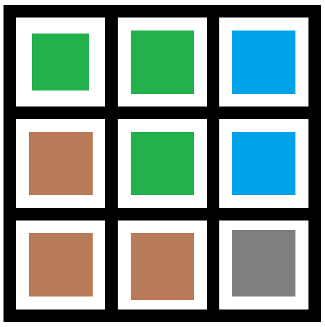


**Fig 02:** Result of 3x3 WFC with given ruleset.

### 3.2 Noises

Noise in programs are generative processes that provide a level of controlled randomness, creating gradients of values with unpredictable patterns, which is why an overwhelming majority of procedural generation uses them. Noises generate either single instances, one-dimensional arrays, two-dimensional arrays, or rarely three-dimensional arrays, that can be used as references within systems to create generative results.

A popular and frequently used noise would be Perlin Noise (Perlin, 1985) which utilises a grid of points that store a pseudorandom gradient vector dictating directions on the grid. A culmination of processes between a given point and the grid vectors generate a map of values limited between 0 and 1 that interpolate into each other, producing no harsh changes in nearby values **(Fig 03)**.

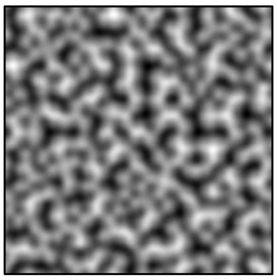


**Fig 03:** Example generation of Perlin Noise.

This does not apply much to the later mentioned Worley and Voronoi, but Perlin Noise by itself can be repetitive or predictable in nature, so it has become common practise to apply further processes to a generated noise map. These processes consist of a few simple steps and complicated steps. The simple steps consist of scaling the X and Y value to stretch or grow the result and shifting by an offset to generate different results from a repeated seed. However, the complicated steps involve multiple noise iterations that gradually multiply its effect (persistence) and frequency (lacunarity) during each iteration. This results in more random and crisp looking patterns that simulate more unnatural compositions. For example, in **Fig 04** they all share the same scale, offset, and seed but gradually look clearer when going from left to right.

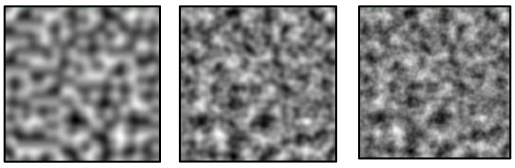


**Fig 04:** Perlin noise with identical variables but 0 additional generations (left), 1 additional generation (middle), and 2 additional generations (right).

Two other generations which are very similar in nature but not as common as Perlin is Worley noise and Voronoi Diagram. Voronoi diagrams are region maps that are split up based on a series of grid positions. It works by placing random points onto a two-dimensional grid to act as region separators. Regions are decided by calculating which point is closest to each pixel of an image and either assigning them a colour or index based on the closest point **(Fig 05)**.

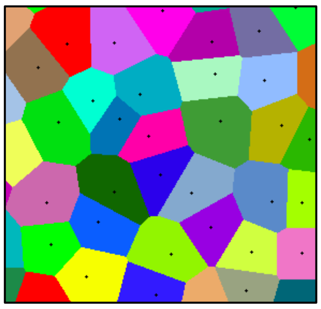


**Fig 05:** Voronoi Diagram.

Worley Noise investigates using these points for their distance instead of their region. Each given pixel of an image finds the ***X*** closest point, where ***X*** is an exchangeable integer that drastically varies the results as shown in **Fig 06**. Both Voronoi and Worley can be used for texture generation and pattern generation.

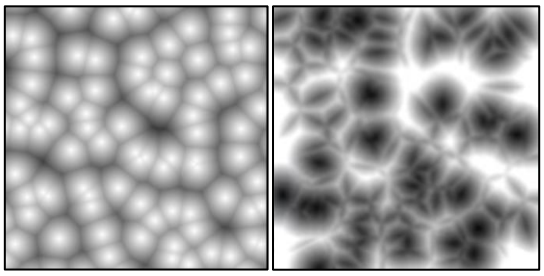


**Fig 06:** Worley noise with an X index of 0 (left) compared to an X index of 1 (right).

These are not the only noises and diagrams that can be used for a procedural context. Another noise that was also discovered by Ken Perlin is Simplex Noise **(Fig 07)** which acts similarly to Perlin but is computationally quicker and can act in higher dimensions. Other noises that are not mentioned were either inadequate for this project or had not been discovered during research.

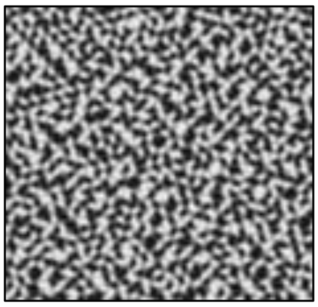


**Fig 07:** Simplex noise

## 3.3 Procedural Content in Existing Games

PCG is defined as the *algorithmic creation of game context with limited or indirect user* (Togelius et al., 2011) that can range from *texture synthesis to atmospheric effect simulation or to landscape geometry specification* (Kirillov, 2018) meaning it can be used systematically with many varying applications for their usage.

A video game example of procedural generation that has a similar setting to this project is Terraria by Re-Logic. It is a 2D side scrolling sandbox game in which the map is generated with minimal input from the user **Fig 08**. It accomplishes their map creation by continuously applying processes onto one single Tilemap to gradually stack components of the environment. These processes consist of digging out caves and tunnels, integrating structures, simulating fluids, and more to have intractability within steps of the generation.

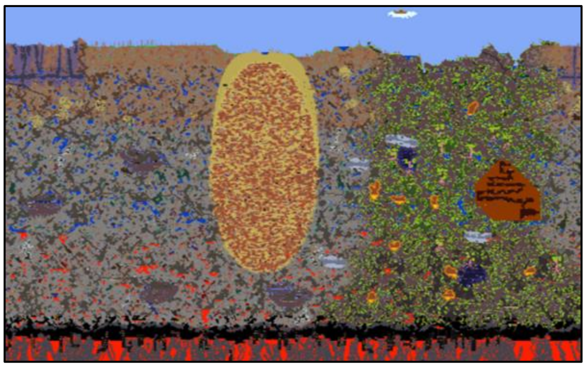


**Fig 08:** Terraria generated world map.

Another famous example that utilises PCG is the 3D sandbox game, Minecraft (Mojang, 2009). It operates in three dimensions yet has caverns, contours, landmasses, and such, by taking randomised values and combining multiple generations together **(Fig 09)**. Both Terraria and Minecraft base most, if not all, of the world on procedural values and thrive off replay ability, and (less so in Terraria due to fixed world sizes) the continuous generation provides seemingly infinite range.

**Fig 09:** Minecraft surface generation.

A less popular application of generative content is the 2D side scrolling rogue-like Noita by Nolla Games. This represents a videogame that does not solely rely on the procedural outcomes for the entire playable world, instead most of the map is predetermined with guaranteed generations through every playthrough and contains sections of randomly generated layouts **(Fig 10)**. This competes against the lack of control that designers have with PCG tools (van der Linden, Lopes, and Bidarra, 2014) by targeting its usage to areas that are not required to be consistent.

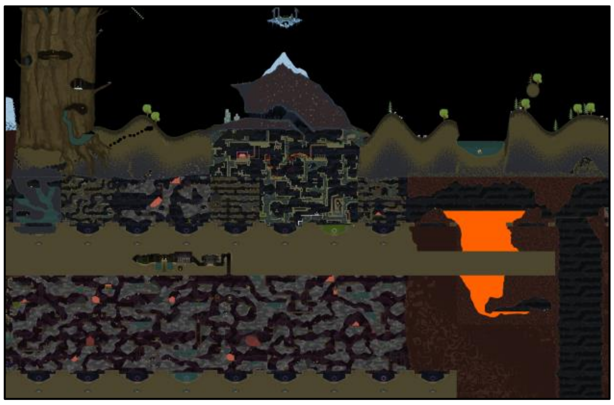


**Fig 10:** Noita map generation.

Additionally, some researchers are investigating incorporating machine learning into PCG to create more tailor-made experiences (Summerville et al., 2018). These approaches have the idea of creating content that adapts to player preferences and behaviour, to direct the world generation to those personised experiences. However, this idea has not been incorporated into public scope yet due it's difficulties and lack of resources put into it so far.

In essence, the varied applications of PCG in existing games show its adaptability and significance in game development. Whether used to craft expansive worlds or enhance replay ability, PCG continues to shape the gaming landscape, offering multiple avenues for immersive gameplay.

## 4. Research methods and Ethics

This project consists of primary research and secondary research to investigate existing procedural content generation in games, primarily investigating academic articles, journals, development logs, research papers, and game engine documentation. Existing public code examples are investigated through GitHub repositories and such to provide as a comparison and guidance on the tools.

For suitable papers and articles, keywords were used when researching topics: Procedural Content Generation, PCG, Procedural, Generation Algorithms, Generation in Video Games, Wave Function Collapse, WFC. By focusing on these keywords only related topics to this project would appear, providing more context on the subject matter to improve upon the procedural knowledge behind the system.

Development frameworks exist as a method of managing projects effectively. Agile frameworks consist of repeating a cycle of planning, production, testing, and feedback. This allows for the project to have an intended goal with room for flexibility to produce a better scenario. Waterfall is a sequential framework which follows similar steps to Agile but have a lot less leeway in iteration between processes, making it less adaptable, which is critical in an independently worked upon project.

Among them, Agile was applied during this project due to its adaptability and iterative approach. Several other development methodologies exist that are useful in certain contexts and with certain personal workflows but were considered less effective for this scenario.

Any code that has been integrated or used as reference for the project is referenced in both code and report. This project does not involve human participants but still has few ethical concerns to consider based on the research conducted.

Depending on the result of this system, if it were to be taken to a larger scope in which the ruleset is generated through large data sets or retrieve rulesets from users and send them to a global server to be analysed or used, then it would raise concerns on the privacy and security of data. On a different topic, procedural content can disrupt the traditional workflow of some developers or studios, potentially resulting in job roles being removed and harming workers financial states and mental wellbeing's. These issued were avoided primarily due to the scope; this system is not reaching levels of database utility and if it were, would be operating through and very different environment and scope.

## 5. Practice

This section encompasses how key components such as the WFC system and map conversion were implemented, elaborating on the successes, setbacks, and changes to plans.

### 5.1 WFC Ruleset Generation

A key concept in the application of WFC is the dictation of rules, to control how certain cells interact with other cells. These rules often state whether a cell of Type A can neighbour a cell of Type B, with most systems advancing that step by controlling which sides they can or cannot neighbour. For this system, the ruleset is necessary to be accurate to the user's needs, however, each user will have entirely different scenarios in which one ruleset will vary to the next, so instead of one predetermined ruleset that is applied to every user, it must be calculated during runtime to best suit their requirements.

For the generated ruleset to adapt to different requirements, as much useful information must be gathered from the user input. To allow for the application to gather an input mimicking cell distribution in a WFC environment, a biome canvas in which the player can interact with is a suitable method as displayed in **Fig 11**.

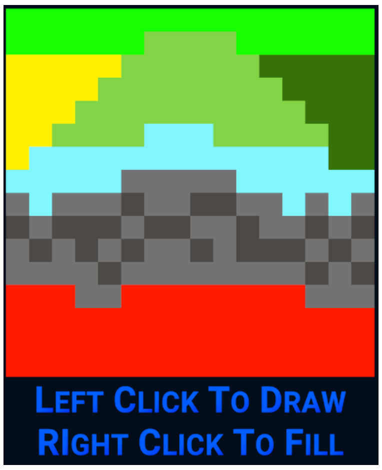


**Fig 11:** Interactive Biome Canvas

The information gathered during the ruleset generation process consists of:

- **Height Variance** - *How likely a cell of Type A can be at that Y position.*
- **Width Variance** - *How likely a cell of Type A can be at that X position.*
- **Diagonal Variance** - *How likely a cell of Type A will be in the diagonal line of Y=X and Y = -X.*
- **Neighbouring Cells** - *What cell Types can neighbour a cell of Type A for each direction.*

The height variance and width variance are calculated by going through every pixel of the player's inputted image and converting a pixel colour to a biome instance at a certain X and Y position, this distinction is shown to the user through a colour palette to solidify the link between colours and biomes. Diagonal variance is similarly accomplished to height and width variance, but instead of calculating based on X or Y positions, it considers both simultaneously. It goes through each pixel and increments the count of the detected biome at an array index of [X + Y] or [X + (height - Y)] which corresponds to counting on a Y=X line and a Y=-X line. The neighbouring cells, however, is generated through a longer process of going through each pixel one by one, checking each pixel surrounding it, and storing the number of times a pixel of biome A will neighbour pixel of biome B in those 4 directions.

These variables are then stored for future reference during the WFC generation. The WFC process is as follows:

1. **Initialise** an array of cells that coincide with each expected biome pixel.
2. Find the cell with the **lowest entropy** (lowest number of possible biomes it can be), if there are multiple cells contesting the lowest entropy, the randomly pick one from the selection.
3. **Collapse** the chosen cell based on the ruleset - *In detail below*.
4. **Update** all neighbouring cells of the collapsed cell - *In detail below*.
5. **Repeat** steps 2,3,4 until every cell has collapsed, resulting in a full map of biome pixels.

During the collapsing of the cell, it calculates the likelihood for each possible biome it can collapse to and randomly select one with weighted priority for certain biomes. Each biome weighting is affected by its height and width variance of that biome at that X and Y position and diagonal position, with an extra step of finding the closest biome pixel to that position; if it is outside a certain range, it will set the weighting to 0, if within that range, it will scale the weighting based on the distance.
When updating neighbouring cells, the 4 neighbouring cells have their possible biomes filtered down based on the ruleset of neighbouring the previously collapsed cell. During this step, any cell that gets updated to an equal biome count possibility as the lowest entropy then it gets added to the list of lowest entropy cells, or if is lower than the current entropy count, then it clears the list and states it as the new lowest entropy cell. This makes the step of finding the lowest entropy cell much quicker as it prevents spending unnecessary time on unimportant cells.

## 5.2 Layering of Procedural Map

The map generation is accomplished through the layering of multiple processes onto one another. The encompassing task for this conversion is to expand the biome map that the user has made into a larger tile map which consists of multiple components for a diverse world whilst keeping the themes of biome types. By making the world generation a step-by-step process of multiple isolated layers it provides multiple benefits to its application and ease of access on the developer side as one faulty generation at the end can quickly be linked to one or two steps of the process.

Starting this conversion is expanding the biome map based on the biome size, creating a transition between neighbouring biomes which is randomised based on their distance from the transitioning line and the transition percent controlled by the user. **Fig 12** shows the difference between 0% and 100%.

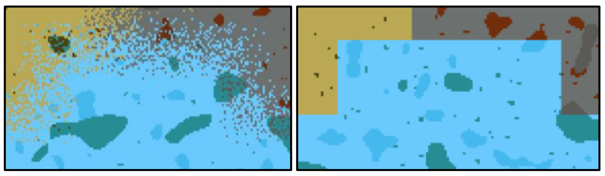


**Fig 12:** Biome Transitions with 100%(left) and 0%(right) Transition Percentages.

A crucial step of the process is the noise maps. The noise maps are what control the overall generation of the map; how caves are formed, how clumps of blocks are gathered, how water is generated, and more that will be further stated. These noise maps are 2D float arrays accomplished by operating 2D Perlin noise generation with varying values of threshold, frequency, and such. Perlin was used over Simplex because it was adequate for the project, and the Math libraries in Unity have Perlin ready to be used whereas Simplex would have needed to be programmed in.

There are five 2D maps and one 1D map, all of which have those differing values distinguish their uses. The only two maps that get extra processes is the cave map and the water map, each pass mentioned here will be elaborated upon further on. The cave map receives a tunnel pass which dig a mixture of horizontal, vertical, and diagonal tunnels into the map and a feather pass which gradually tapers off the map edges to prevent caves/tunnels to lead out of bounds. The water map requires a settling pass so that the generated water is affected by gravity and pooled together.

After these layers have been generated, the full map conversion commences. Each tile receives a selective process which accounts for what biome to generate with and what layers are present. For example, a tile that is being generated in a jungle biome will consider whether it is in a cave layer, if not then it considers the large clump, small clump, and dots layer to differentiate between different block generations. Whereas, if it were in the cave layer, it checks the state of the water layer, deciding if it locks into a water tile, otherwise it detects the above tile and below tile, calculating if it randomises a vine generation or floral generation. This process is effective to reproduce and customise on the developer side as it is located within an independent class that solely converts biome tile generation to a TileID return, abstracting the two systems apart.

Further diversifying the world is the structure pass which takes the existing Tilemap generation and inserts treasure generation and crystal clusters. The treasure generates upon the surface at a customisable rate by the user to accustom the level of difficulty to their needs. Crystal clusters generate with either Voronoi or Worley patterns to act as a procedural texture application, of which can be altered to the players favourable sizes and frequencies.

### 5.3 Map Visualization

Initially, the map generation was shown to the user through a Unity Tilemap visualization. This proved effective in displaying a dynamic Tile to correspond with TileID's, providing the instant possibility of texturing each tile. However, this proved to be an ineffective method, for each map visualization would cost a long processing time, often resulting in minutes of Unity Tilemap generation. This method was scrapped and in turn replaced with a sprite generation system, each TileID still corresponds to a Unity Tile, but only retrieves the colour of the tile to be used. The result of the map generation is a roughly 2048x1024 pixel image where each pixel is the colour of the tile, giving enough context and representation of the map without causing a massive influx in processing time.

Additional functionality includes a frame-by-frame process viewer and looping. The frame-by-frame viewer is used within the WFC step which showed the cells being collapsed one by one, allowing for an appropriate visualisation of the process whilst also acting as a debug tool to discover inconsistencies in the generation. The looping is utilised in both WFC and map generation, this constantly repeats the process showing rapid construction of the process to measure the consistency and speed of the generations.

Upon the map generation step being completed, additional information, such as noise layers, about the independent layers is stored in the class. This is then retrieved by the map visualizer and shown to the user in translucent layers **(Fig 13)** to represent how important generations are accomplished, such as the difference between the cave layer before and after the tunnel pass as shown below.

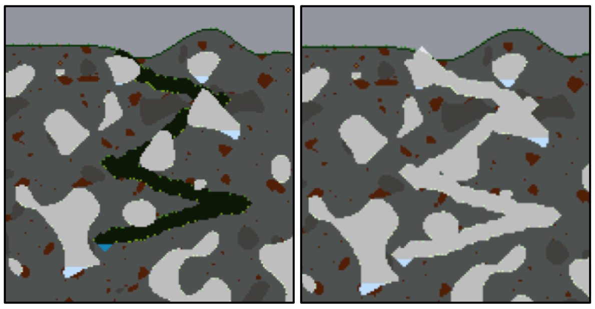


**Fig 13:** Layer viewer of cave layer before (left) and after (right) the tunnel pass.

### 5.4 Interface Environment for User

The user interface provides the connection between the player and the systems, allowing accessible variable altering for map generation logic and canvas sizes.

All available values are shown through a ValueEditor class which provides minimum, maximum, and default values for floats or integers. This coincides with a constant update check to pair the slider and input field to always be identical. These ValueEditor classes are then further applied through fields, starting with a 'Base Field' which allows for an effective communication between one listener and multiple ValueEditors. Of which gets developed into 'Noise Fields' **(Fig 14)** and 'Tunnel Fields' that specialise in those two areas.

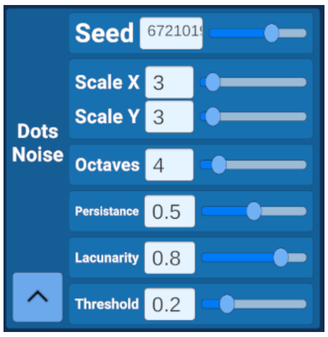

Dots Noise
Seed 672101
Scale X 3
Scale Y 3
Octaves 4
Persistance 0.5
Lacunarity 0.8
Threshold 0.2

**Fig 14:** Noise Field containing multiple ValueEditors.

Previously mentioned are biome palettes that generate a scroll view of biome colours and names which, when pressed, will become the active biome for drawing and filling in for canvas operations **(Fig 15)**. The colours are variable in the draw canvas class which contains a list of all possible biomes and their respective colours.

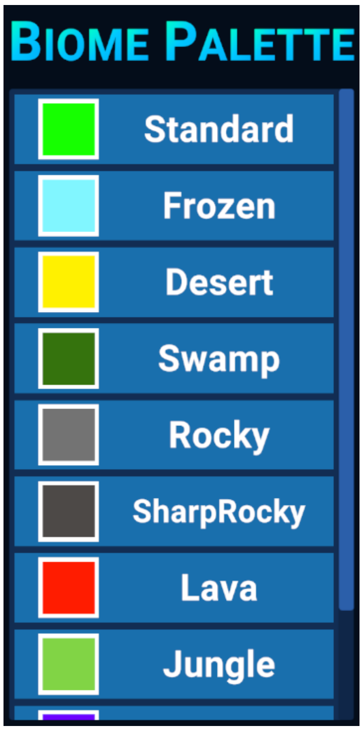


**Fig 15:** Biome Palette of multiple biome types and colours.

Other functionality within the UI is accessibility to saving and loading creations to the user's storage, inserting a given texture into the draw canvas process, toggling between each generative process, and choosing which layers to overlay when viewing the final map generation.

On the developer side of UI control, all ValueEditors can have their minimum, maximum, and default value altered, and integer or float state toggled, biome palettes can be customised with differing colours and additional biomes, and more layers can easily be incorporated into the layer viewer. All providing the possibility of implementing further improvements.

### 5.5 Optimisation, Limiters and Code Base

Optimisation is critical for lengthy processes such as map generations, so calculating time and space complexities of processes greatly assist in improving the optimisation of the system.

**WFC Processing Time:**

| Tile Size | Average Time (sec) | Time Per Tile (sec * 10^-6) |
|---|---|---|
| **32*16** | 0.0055847 | 10.9076 |
| **64*32** | 0.0249099 | 12.1630 |
| **128*64** | 0.0878601 | 10.7251 |
| **256*128** | 0.3214263 | 9.80815 |

**Map Generation Processing Time:**

| Tile Size | Average Time (sec) | Time Per Tile (sec * 10^-6) |
|---|---|---|
| **128*64** | 0.0953369 | 11.6378 |
| **256*128** | 0.2038269 | 6.2203 |
| **512*256** | 0.4157409 | 3.1715 |
| **1024*512** | 1.2499085 | 2.3840 |
| **2048*1024** | 6.0654907 | 2.8922 |
| **4096*2048** | 33.2074870 | 3.9586 |
| **8192*4096** | 200.0903667 | 5.9632 |

For a map generation of size 2048*1024 the time disparities are: 2% biome map expanding, 61% layer generation with each layer being roughly identical timing except for the cave and water layers having slightly more processes, 6% tile conversion when using layers as reference, 30% structure generation, and the final 1% of other processes.

From all this information it can be stated that the WFC system has a relatively consistent process time when paired to total tiles to generate, classifying as a O(n) time notation. However, the map generation is showing an interesting curve of time complexity, it starts with a higher time per tile value, then massively decreases to a quarter of that value, but then slowly increases the more tiles that need to be generation, classifying as a O($n^2$) notation past the 8 million tile mark, indicating that a process has a fault that scales with tiles upon reaching high quantities.

All code is online within a GitHub repository through the software of GitKraken, which allowed for the project to be accessible anywhere, whilst providing a timestamped version history of when features were implemented, changed, or fixed. The repository also has a README.md and Wiki about the features and application of use to allow for further development from other users that are not familiar with the system.

## 6. Discussion of outcomes

To discuss the application of this system in the intended environment, a diverse set of generations must be analysed considering both successes and faults within them and elaborate further onto the limitations causing these. The result of the system heavily depends on both the WFC result and the map generation.

### 6.1 Generated Results of WFC

An accurate method of evaluating the WFC's successes and failures is with varying types of inputs. Four input types were chosen to portray diverse possible uses are: uniform layouts, irregular layouts, diagonal layouts, and random layouts.

These uniform generations in **Fig 17** show that the generated WFC ruleset understands from the canvas in **Fig 16** the location of where biomes should be situated, what biomes can neighbour each other, and contains a level of randomness between transitions of biomes.

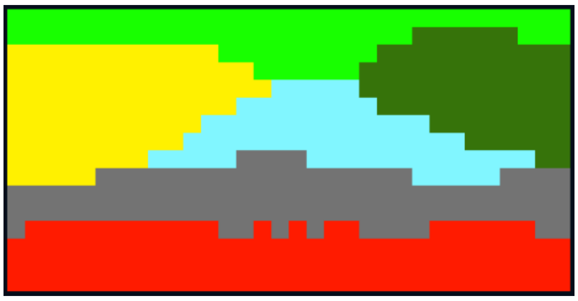


**Fig 16:** Canvas input of uniformly placed biomes.

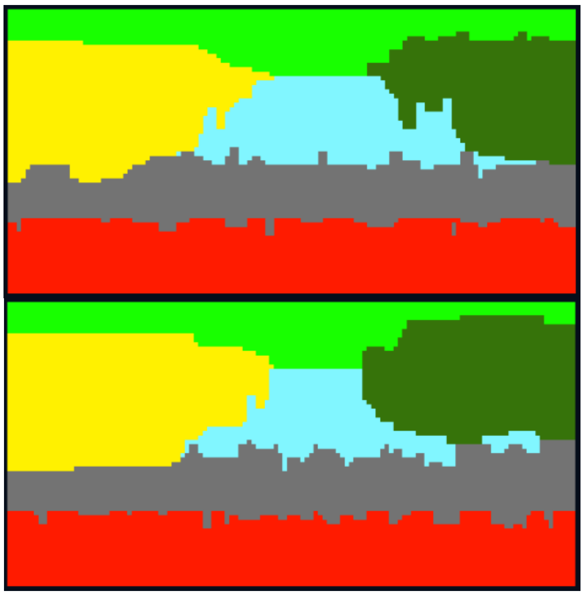


**Fig 17:** WFC results of uniform input.

The irregular generations in **Fig 19** start showing some flaws in the system. Any biomes that occupy a small number of pixels are rarely incorporated due to the weighting of that biome being drastically smaller than a nearby more popular biome. When considering the overall generation, it is still accurately separating the requested biomes into adequate positions and sizes like the layout from the canvas in **Fig 18**.

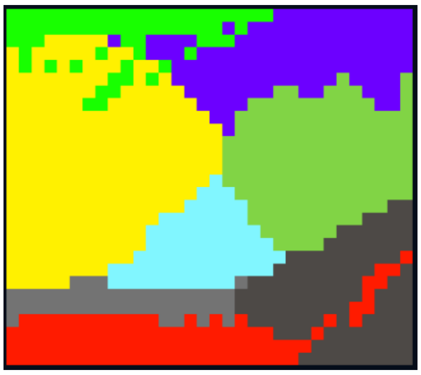


**Fig 18:** Canvas input of irregularly placed biomes.

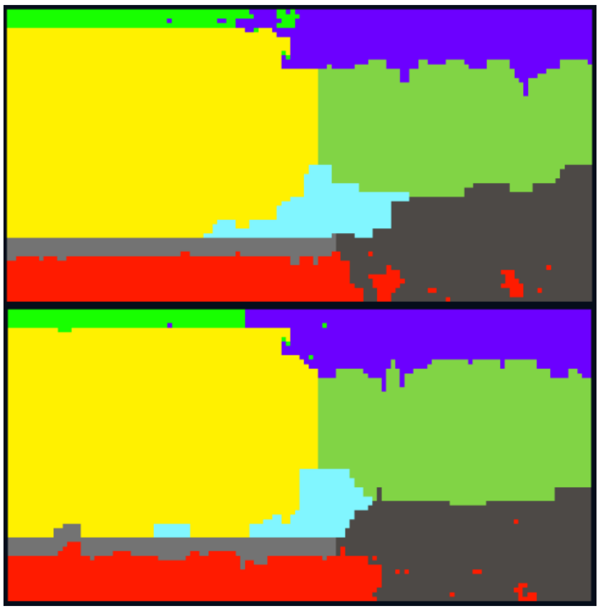


**Fig 19:** WFC results of irregular input.

Previously there was an immense flaw within the ruleset, which was the ability to understand diagonal patterns. Due to the ruleset dictating biome spreads through X and Y coordinates, it cannot distinguish between multiple weightings in width and height. For example, if Biome A had an input spread of (0,0) and (8,8) on a 2D grid, the ruleset would understand X position 0 and 8, and Y position 0 and 8 to be populated areas, this can inaccurately decide that positions (0,8) and (8,0) should be high in the biome weightings because it fulfils those position requirements, where in reality they should never generate there.

This results in a generation like **Fig 20** in which biomes are positioned in incorrect locations and transitions to other biomes when it shouldn't.

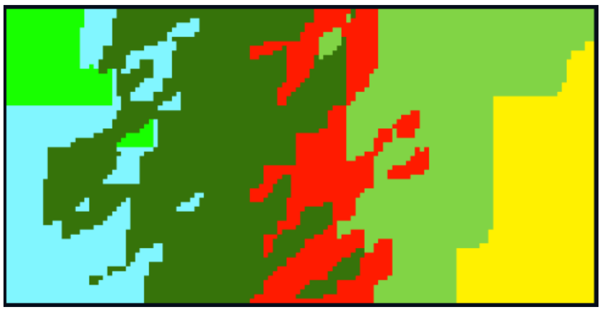


**Fig 20:** Previous diagonal generations

To fix this, the ruleset was granted the two additional values of diagonal weightings, one going from top left down to bottom right, the other going from bottom left to top right, thus creating multiple intersections that reduce the likelihood of inaccurate positioning. Now showing that diagonal patterns in **Fig 21** get detected by the ruleset and is correctly generated in **Fig 22**.

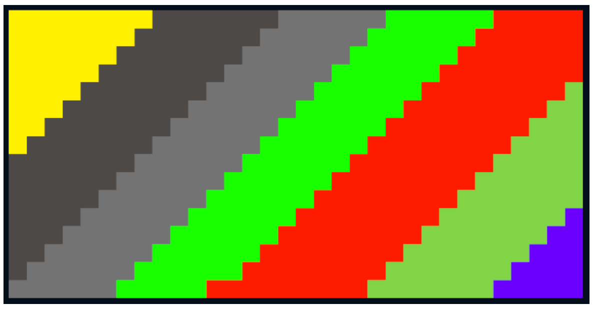


**Fig 21:** Canvas input of diagonally placed biomes.

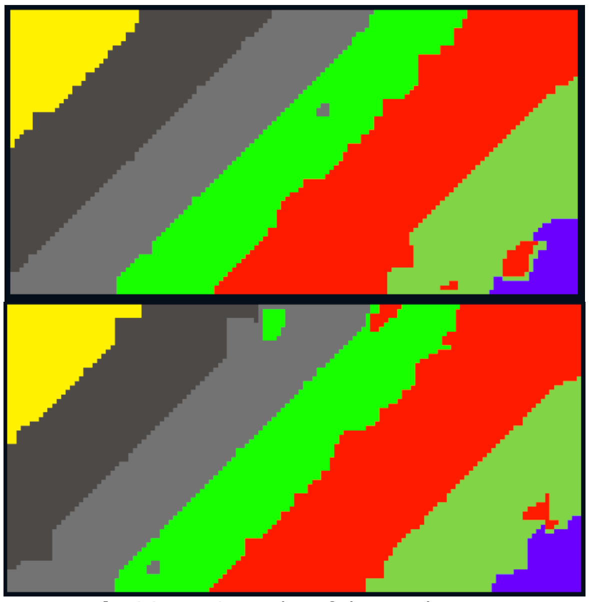


**Fig 22:** WFC results of diagonal input.

Another layout demonstrates a failsafe situation in which a completely unusable input in **Fig 23** can still be converted into an appropriate biome map shown in **Fig 24** The map generation will produce a low-quality map, but the WFC still provides a functional output.

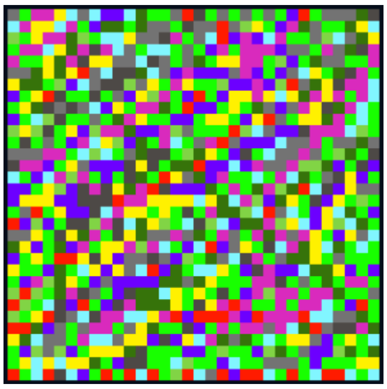


**Fig 23:** Canvas input of randomly placed biomes.

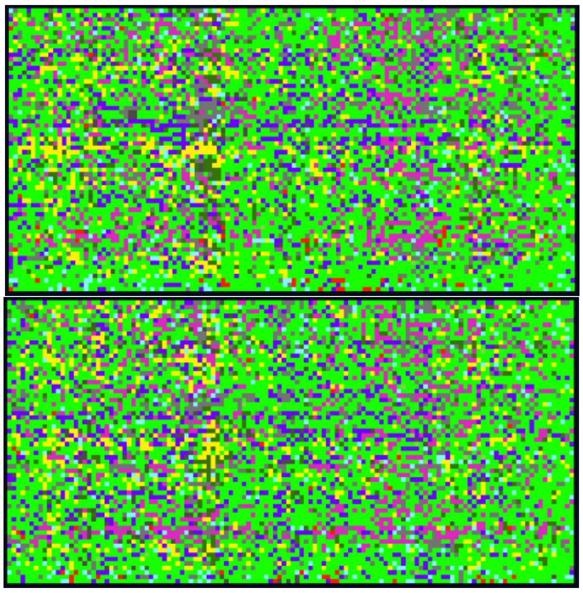


**Fig 24:** WFC results of random input.

## 6.2 Generated Results of Map

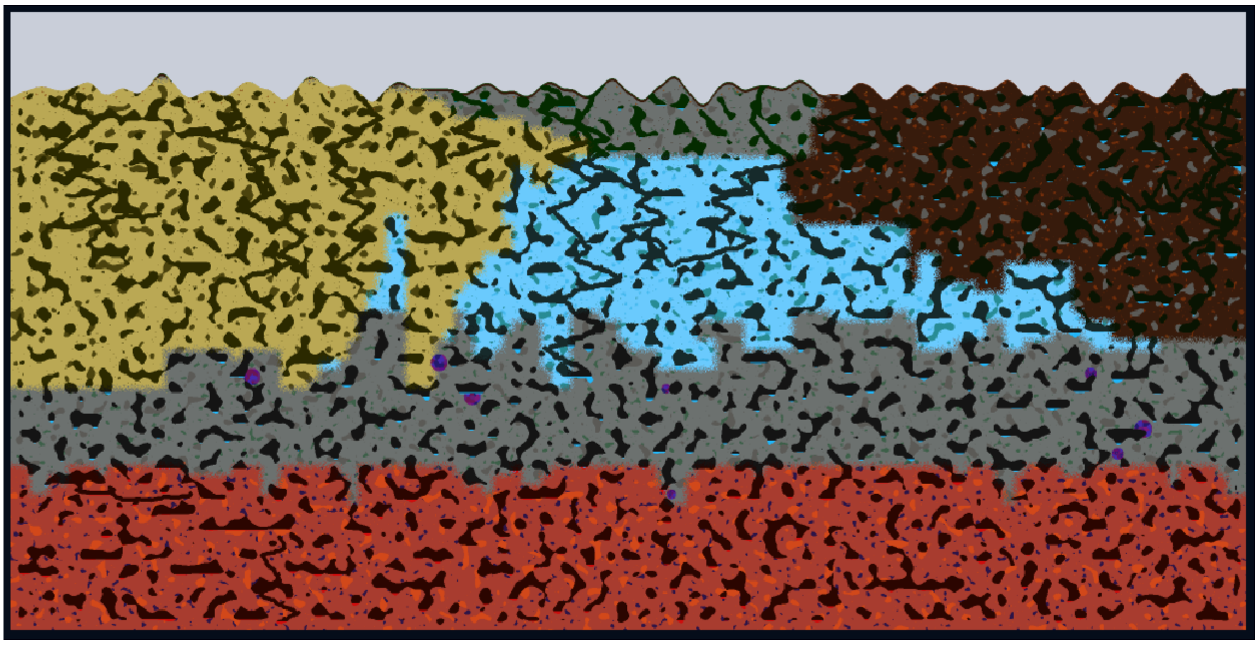


**Fig 25:** Map generation of uniform biome map.

The map generation is more difficult to analyse due to every important variable being accessible to the user, providing countless generation types. For the sake of this analysis, the map generation will be using the default values given to the system.

Creating a map from the uniform layout in **Fig 25** generates a 2048 by 1024 pixel sized map in Fig 00. From this and similar map generations it can be deduced whether the system has distinct biome generation, unique traversable caves, correctly settled water, and more.

The water generation can be seen in **Fig 26** through two viewing instances, one just as the noise generation, the other being the process of settling and culling the tiles. These show accurate simulations to how water should generate in a 2D environment by abiding to gravity and filling in gaps within caves and tunnels.

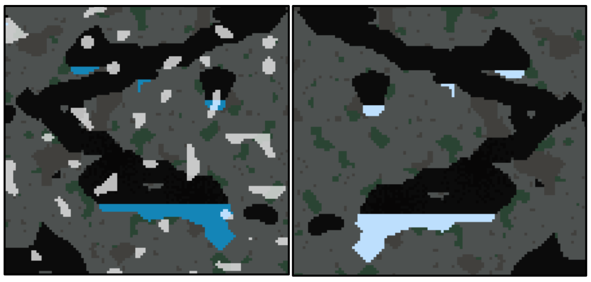


**Fig 26:** Water layers before (left) and after (right) settling and culling.

Seeing tunnels generate in a culmination of directions and scenarios shows its utility and applicable nature. **Fig 27** shows contains horizontal, vertical, and flat tunnel generations that all indicate a randomly pathed tunnel that changes its direction and length after each turn.

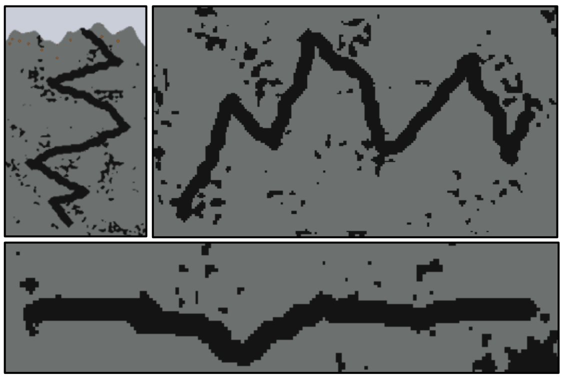


**Fig 27:** Tunnel generations of varying lengths and directions.

Surfaces successfully generate surface treasure that have a block of sand indicating a spawn below it, that being a treasure tile surrounded by sand in **Fig 28**.

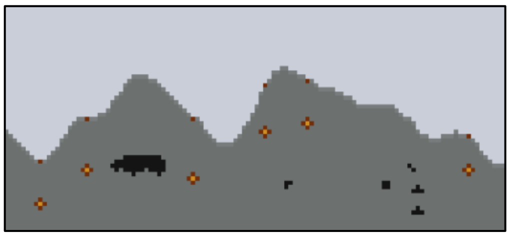


**Fig 28:** Surface Treasure of varying depths.

Alongside surface treasure for structure generation is crystal clusters that use Voronoi and Worley noise for their generative behaviour. These evenly spread the different types of crystal tiles and randomly generate adequately in sharp rock biomes **(Fig 29)**.

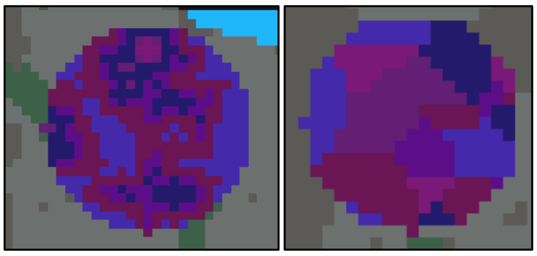


**Fig 29:** Crystal Clusters of both Worley (left) and Voronoi (right).

The overall generation depends on the individual layers used. The layers shown in **Fig 30** correctly form together to create the full Tilemap in **Fig 31**.

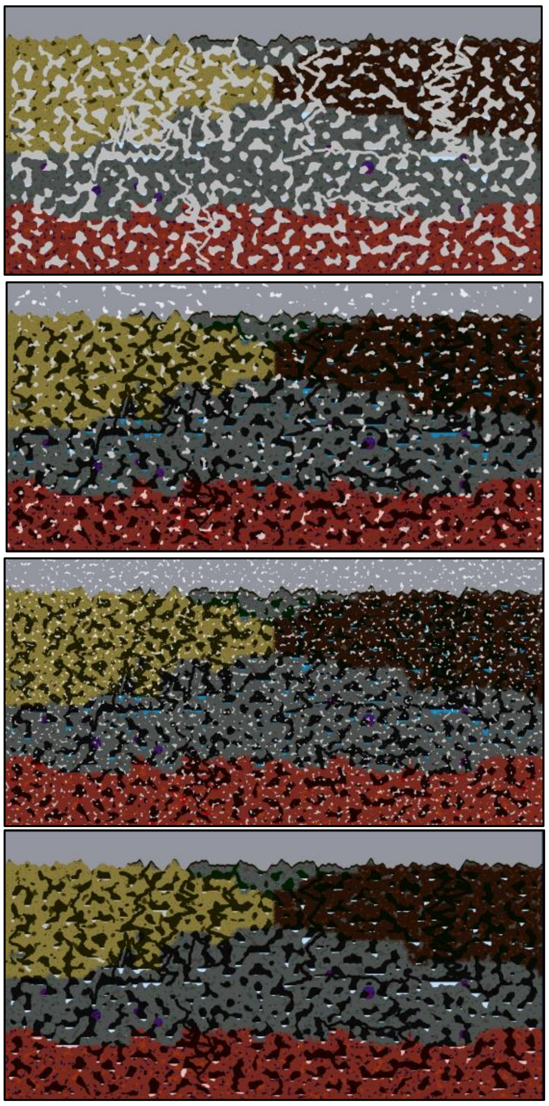


**Fig 30:** Procedural layers of a map generation.

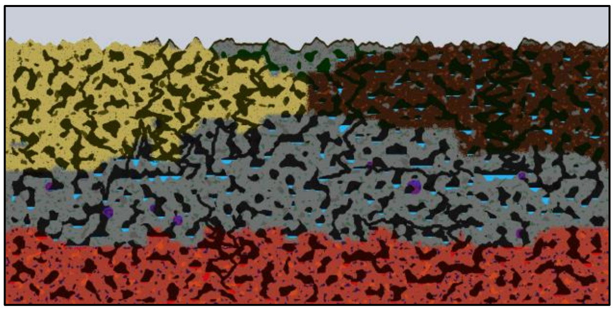
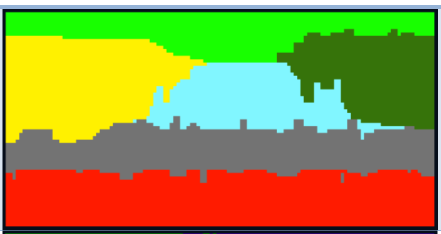
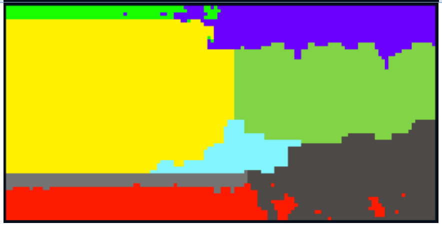
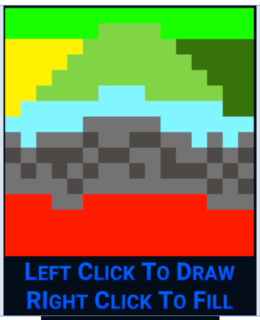
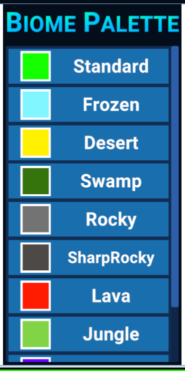
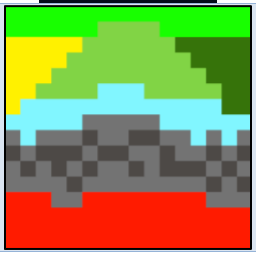


**Fig 31:** Full Tilemap of layers combined.

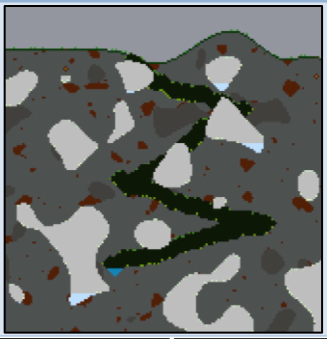
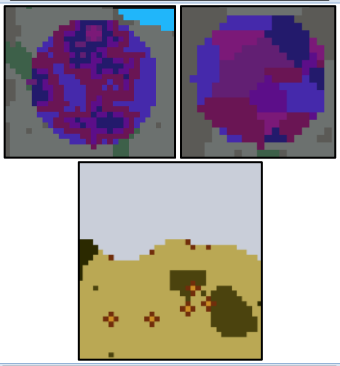
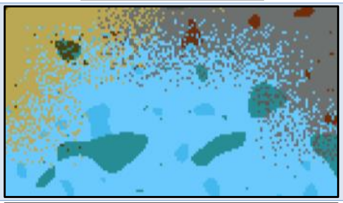
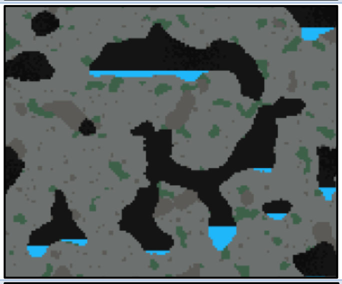
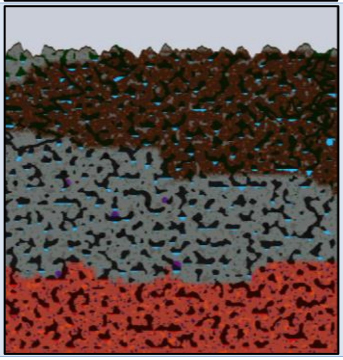
## 6.3 Evaluation of WFC and Map Generation

To evaluate the success of both systems, multiple components within then will be broken down and ranked based on how well it operates compared to the intended functionality in an unbiased manner. Within the rating section, anything rated as a 0 is noted as a failure, 1 is a success, 5 is what it is intended to be, anything in between is an interpolation between 1 and 5.

**For WFC Systems:**

| System | Rating | Notes | Proof |
|---|---|---|---|
| **Ruleset Generating and Utility for Simple Inputs** | 4/5 | The input canvas is understood by the system and converted into an appropriate ruleset that. When used for a map, outputs an accurate result. Rarely it does have faults in generating anomalous biome spreads. |  |
| **Ruleset Generating and Utility for Complex Inputs** | 3/5 | The input canvas is understood by the system and converted into an appropriate ruleset. However, can provide incorrect map generations with biomes being missed out due to the chances for them to generate when in reduce group sizes being small. |  |
| **Canvas Can be Interacted Upon by the User** | 5/5 | Canvas can be interacted with by left clicking and right clicking to draw or fill with the selected biome of choice. |  |
| **Canvas is Dynamic to Developer Dictated Variables** | 5/5 | All palettes and canvases adapt to the dictated list of biome types and colours that the developers have set. The only scenario in which a conflict can occur is if a developer introduces new biomes or removes existing biomes and does not have a version system to convert saved instances. |   BIOME PALETTE<br>Standard<br>Frozen<br>Desert<br>Swamp<br>Rocky<br>SharpRocky<br>Lava<br>Jungle |
| **WFC Visualisation** | 5/5 | The canvas shows the biomes layout accurately by pairing a colour to a biome type, which can be edited. WFC system can also be toggled to show frame-by-frame generation for the developer to see the process gradually, checking for faults or generative patterns. |  |

**For Map Generation:**

| System | Rating | Notes | Proof |
|---|---|---|---|
| **Layer Generation** | 5/5 | All of the procedural layers produce the intended result with variables that the user controls, allowing variety in scale, output size, persistance, lacunarity, octaves, and seed. |  |
| **Structure Generation** | 4/5 | The surface treasure and crystal clusters both generate as requested with reference to player controlled values, allowing for more or less common generations. Although, more structures could have been implemented to show higher range of utility. |  |
| **Biome Transitions** | 5/5 | Biomes transition with varrying percentages with no errors and provide random variation. |  |
| **Water Generation** | 4/5 | Water is generated with realistic application of settling through gravity, and is culled by the cave layer. The only scenario that does not suit the intended environment is if water is generating on the border of a desert biome. Desert biomes have no water generation so it will convert it to blank tiles, creating floating water. |  |
| **Map Visualisation** | 5/5 | Map is visualised quickly and adequately by having tiles represent a colour similar to what they portray. The input of moving around and zooming in and out provide good utility to view the map. |  |

**Heuristic Evaluation:**

Developers analyse a product's interface using heuristics to find any issues. A heuristic evaluation is a form of problem-solving strategy that guide usability assessment and uncovers usability problems (Nielsen, 1992). Good design within a tool or project results in effective communication for the system. It would be more optimal to have more than one person help to assess a heuristic review to reduce bias, however, was not achieved during this project. When reviewing it as one person, understanding user perspectives helps mitigate bias and false perspectives.

| Heuristic Number | Heuristic Name | Heuristic Application |
|---|---|---|
| **1** | Visibility of system status. | The map generation has a progress bar that shows the percentage of completion accomplished at that point with extra information about what processes had just occurred and what processes will next occur. |
| **2** | Match between the system and the real world. | Terminology from PCG and game development are utilised for users to understand what tools are and what they are referred to in the overall context for external research. |
| **3** | User control and freedom. | Users can draw and fill for the canvas operation, and can save and load to maintain their creation between sessions. However they do not have a history log of changes during canvas sections, so they cannot undo or redo operations. |
| **4** | Consistency and adherence to standers. | UI is consistently shown to the user with systems of similar operations to be located in nearby locations and grouped up based on their combined utility. |
| **5** | Error prevention, specifically prevention usability-related errors. | UI fields such as sliders are validated to stay within predetermined bounds dictated by the developer. When operating through systems, multiple condition checks are flagged to prevent erroneous opperations or indicate to the developer when errors would occur. |
| **6** | Recongnition rather than recall. | Value editors all have the same layout and accessibility to help the user understand the overall system. Slider names are stated in the README and Wiki but not shown as a hovered-over tooltip. |
| **7** | Flexibility and efficiency of use. | The UI is consistent with it's layout and has no flexibility. However, each value editor is minimised to improve the rate of traversal to get to certain values. |
| **8** | Aesthetic and minimalism in design. | There is a level of cluttered UI that can be improved upon but is easy to understand and focusses on the key tools within the system. The value editors are less overwhelming by having a minimise toggle, but can be further minimised with each section being grouped. |
| **9** | Recognition, diagnosis, and recovery from errors. | When issues arrise, the system does log them to the console for developers but do not have any feedback in case a player encounters it. A popup window can be implemented to show any logs. |
| **10** | Help and documentation. | The README.md and wiki both contains information about the system, values, and processes, offering understanding of how individual components can be tweaked. |

## 7. Conclusion and recommendations

By combining the features of Wave Function Collapse and general Procedural Content Generation techniques, a unique system in which the player can have complete and quick control of a full world map generation process for a 2D side-view video game has successfully been achieved. This system has advanced the level of creativity that a player can be given with a procedural environment while having a set of developer dictated limitations.

With the provided adaptability of the system, this can be further expanded upon for more biome types and generations, extra tiles, additional layers, and more to grow upon the generative process. To further increase the accuracy of the system, especially within the WFC process, additional ruleset calculations can be developed upon to work alongside the existing tools. This report, as a concept, can even be developed into a 3-dimensional counterpart, in which the WFC functionality considers a Z-axis.

More PCG tools would also assist in the diverse generative environment by varying how elements are produced, especially if more structures were developed, such as mazes or dungeons which would utilise differing processes to WFC. An important factor that limits its uses in commercial environments is the level of interaction design that was not focused upon due to scope, if resources were applied to this area, then the tool would be much more applicable in a public setting, being easier to use and understand.

In conclusion, this project has become another step in the direction of more immersive inclusion between players and videogames, allowing systems that build on top of this to reach more adventurous generative processes and revolutionise the procedural video game market.

## 8. References

Axosoft, LLC. (2014) GitKraken [computer program]. Available from: https://www.gitkraken.com/ [Accessed 20 Apr. 2024].

Biagioli, A. (2014). Understanding Perlin Noise. [online] adrianb.io. Available at: https://adrianb.io/2014/08/09/perlinnoise.html [Accessed 25 Oct. 2023].

Cheng, D., Han, H. and Fei, G. (2020). Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm. Lecture Notes in Computer Science, pp.37–50. doi: https://doi.org/10.1007/978-3-030-65736-9_3 [Accessed 21 Apr. 2024]

GitHub, Inc. (2008) GitHub [computer program]. Available from: https://github.com/ [Accessed 20 Apr 2024].

Kirillov, A. (2018). Non-periodic Tiling of Procedural Noise Functions. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 1(2), pp.1–15. doi: https://doi.org/10.1145/3233306 [Accessed 20 Apr. 2024].

Minecraft (2009). PC / Xbox / Play Station / Mobile / Switch [Game]. Mojang, Stockholm.

Nielsen, J. (1992). Finding usability problems through heuristic evaluation. Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92. doi: https://doi.org/10.1145/142750.142834 [Accessed 21 Apr. 2024]

Nielsen, J. (2020). 10 Heuristics for User Interface Design. [online] Nielsen Norman Group. Available at: https://www.nngroup.com/articles/ten-usability-heuristics [Accessed 20 Apr. 2024].

Noita (2020). PC [Game]. Nolla Games, Helsinki.

Perlin, K. (1985). An image synthesizer. ACM SIGGRAPH Computer Graphics, 19(3), pp.287–296. doi: https://doi.org/10.1145/325165.325247 [Accessed 23 Oct. 2023]

Stangl, R. (2017). Procedural content generation: techniques and applications. [online] Available at: https://umm-csci.github.io/senior-seminar/seminars/spring2017/stangl.pdf [Accessed 23 Oct. 2023].

Summerville, A., Snodgrass, S., Guzdial, M., Holmgard, C., Hoover, A.K., Isaksen, A., Nealen, A. and Togelius, J. (2018). Procedural Content Generation via Machine Learning (PCGML). IEEE Transactions on Games, 10(3), pp.257–270. doi: https://doi.org/10.1109/tg.2018.2846639 [Accessed 21 Apr. 2024]

Terraria (2011). PC / Xbox / Play Station / Mobile [Game]. Re-Logic, Indiana.

Togelius, J., Kastbjerg, E., Schedl, D. and Yannakakis, G.N. (2011). What is procedural content generation? Proceedings of the 2nd International Workshop on Procedural Content Generation in Games - PCGames '11. [online] doi: https://doi.org/10.1145/2000919.2000922. [Accessed 20 Apr. 2024].

Togelius, J., Shaker, N. and Nelson, M.J. (2016). Introduction. Procedural Content Generation in Games, pp.1–15. doi: https://doi.org/10.1007/978-3-319-42716-4_1 [Accessed 21 Apr. 2024]

Unity Technologies (2024). Mathf. Unity Documentation. Available at: https://docs.unity3d.com/ScriptReference/Mathf.html [Accessed 20 Apr. 2024].

Unity Technology (2024) Unity. [Computer Program]. Available at: https://unity.com/ [Accessed 20 Oct. 2023].

van der Linden, R., Lopes, R. and Bidarra, R. (2014). Procedural Generation of Dungeons. IEEE Transactions on Computational Intelligence and AI in Games, 6(1), pp.78–89. doi: https://doi.org/10.1109/tciaig.2013.2290371 [Accessed 23 Oct. 2023].

Worley, S. (1996) 'A cellular texture basis function', in International Conference on Computer Graphics and Interactive Techniques: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. [Online]. 1996 ACM. pp. 291–294. [Accessed 23 Oct. 2023].

## 9. Bibliography

Anderson, J. et al. (2010) Effective UI. First edition. Beijing ; O'Reilly.
[Accessed 20 Apr. 2024]

McKay, E. N. (2013) UI is communication : how to design intuitive, user-centered interfaces by focusing on effective communication. 1st edition. Amsterdam, Netherlands: Elsevier.
[Accessed 20 Apr. 2024]

www.redblobgames.com. (n.d.). Red Blob Games. [online] Available at: https://www.redblobgames.com/
[Accessed 23 Oct. 2023].

The Book of Shaders. (n.d.). The Book of Shaders. [online] Available at: https://thebookofshaders.com/12/
[Accessed 23 Oct. 2023].

Togelius, J., Whitehead, J. and Bidarra, R. (2011). Guest Editorial: Procedural Content Generation in Games. IEEE Transactions on Computational Intelligence and AI in Games, 3(3), pp.169–171. doi: https://doi.org/10.1109/tciaig.2011.2166554
[Accessed 21 Apr. 2024]

Tondello, G.F., Kappen, D.L., Mekler, E.D., Ganaba, M. and Nacke, L.E. (2016). Heuristic Evaluation for Gameful Design. Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts. doi: https://doi.org/10.1145/2968120.2987729
[Accessed 21 Apr. 2024]

www.youtube.com. (n.d.). Coding Perlin Noise in Game Maker. [online] Available at: https://www.youtube.com/watch?v=eevjZsMYx6M
[Accessed 23 Oct. 2023].

www.youtube.com. (n.d.). What Actually Happens when Terraria creates Worlds! [online] Available at: https://www.youtube.com/watch?v=bF-_AeRRbmU
[Accessed 23 Oct. 2023].

https://www1.uwe.ac.uk/students/studysupport/studyskills/referencing/uwebristolharvard.aspx

**Appendix A: Project Log.**

| Month | Task(s) | Duration (Days) |
|---|---|---|
| **October** | Create Project Proposal | 3 |
| | Finalise and Submit Proposal **[26th]** | 1 |
| | Research Procedural Functions | 3 |
| | Basic Procedural Function Scripts | 2 |
| | | **9** |
| **November** | Create WFC Conversion System | 7 |
| | Create Interactive Canvas | 2 |
| | Combine Canvas with WFC System | 1 |
| | Implement Image Importing | 2 |
| | Implement Image Exporting | 2 |
| | | **14** |
| **December** | Image to Tile map Conversion | 1 |
| | Multiple Biome Generations | 7 |
| | Implement Procedural Functions | 3 |
| | Implement Biome Transitioning | 3 |
| | Contingency Time | 3 |
| | | **17** |
| **January** | Refine and Add More Biome Generation | 5 |
| | Create Functional Unity Build | 2 |
| | Create Poster | 2 |
| | Poster and Progress Update **[18th]** | 0 |
| | Poster Presentation **[22nd]** | 1 |
| | Contingency Time | 5 |
| | | **15** |
| **February** | Structure Generation | 6 |
| | Tunnel and River Generation | 4 |
| | Create Variable Sliders for Generation | 3 |
| | Contingency Time | 2 |
| | | **15** |
| **March** | Finalize Biome Generation | 6 |
| | Finalize Biome Blending | 3 |
| | Finalize Miscellaneous Generation | 3 |
| | Contingency Time | 3 |
| | | **15** |
| **April** | Generate Mass Examples of Generation | 1 |
| | Create Structure of Report | 2 |
| | Create Full Report Draft | 6 |
| | Finalise Report | 4 |
| | Create Project Video | 1 |
| | Report Submission **[25th]** | 0 |
| | Contingency Time | 2 |
| | | **16** |
| **May** | Vivas **[13th]** | 0 |

**PREVIOUS LOG CHANGES:**

| Month | Task(s) | Duration (Days) |
|---|---|---|
| **January** | More Biome Generations | 4 |
| | Create Functional Unity Build | 4 |
| | Create Poster | 4 |
| | Poster and Progress Update **[18th]** | 0 |
| | Poster Presentation **[22nd]** | 1 |
| | Contingency Time | 2 |
| | | **15** |
| **February** | Structure Generation | 6 |
| | Tunnel and River Generation | 4 |
| | Contingency Time | 5 |
| | | **15** |
| **March** | Create Variable Sliders for Generation | 3 |
| | Finalize Biome Generation | 4 |
| | Finalize Biome Blending | 3 |
| | Finalize Miscellaneous Generation | 3 |
| | Contingency Time | 2 |
| | | **15** |

**Appendix B: Project Timeline.**

| Date | Activities / Notes | Codewords |
|---|---|---|
| **21 Apr** | Final build made and uploaded to OneDrive. | OneDrive |
| **21 Apr** | Final push with small value tweaks. | Git Push |
| **21 Apr** | Video made that showcases system. | Video |
| **20-25 Apr** | Finalising report. | Report |
| **19 Apr** | **Online Meeting with Supervisor. [Notes Below]** | **Meeting** |
| **17 Apr** | Implementing diagonal weighting for WFC which fixed false generations, and fixed small value tweaks. | Implementation, Fix, Git Push |
| **3-19Apr** | Finalising first report draft. | Report |
| **8 Apr** | Fix transition bug and removed 2 biomes for time constraints. | Fix, Git Push |
| **7 Apr** | More sliders for generation variables. | Git Push |
| **4 Apr** | Layer Viewer implemented and more refactoring/fixing. | Implementation, Git Push, Fix |
| **29 Mar** | Project reformatting and fixing. | Git Push, Fix |
| **29 Mar** | **Online Meeting with Supervisor. [Notes Below]** | **Meeting** |
| **20 Mar** | Structure generation implemented with surface treasure and crystal clusters underground. | Implementation, Git Push |
| **17 Mar** | Starting report draft. | Report |
| **16 Mar** | Refactoring, more UI sliders, and more progress information on progress bar. | Implementation, Git Push |
| **15 Mar** | Tunnel generation integrated with UI fields and discovered bug with feathering that was fixed. | Implementation, Git Push, Fix |
| **8 Mar** | Better biome generation and start of water generation. | Implementation, Git Push |
| **29 Feb** | Tunnel generation implemented. | Implementation, Git Push |
| **28 Feb** | Project log creation. | Documentation |
| **21 Feb** | **Online Meeting with Supervisor. [Notes Below]** | **Meeting** |
| **19 Feb** | Fix for seed randomisation when generating Map Gens. | Fix, Git Push |
| **18 Feb** | Input fields for UI such as noise tweaking and threshold | Implementation, Git |

| | | |
|---|---|---|
| | editing implemented for all three processes. | Push |
| **16 Feb** | Tweak field implemented for UI value editing. | Implementation, Git Push |
| **16 Feb** | Complete layout overhaul for all systems, including new progress bar. | Implementation, Git Push |
| **16 Feb** | Discovered cause of time-consuming process being the noise generation. | Discovery |
| **22 Jan** | **Poster Presentation. [Notes Below]** | **Deadline** |
| **17 Jan** | **Online Meeting with Supervisor. [Notes Below]** | **Meeting** |
| **15 Jan** | Map generation is applied to a texture instead of tile map to drastically reduce time to display. | Fix, Git Push |
| **15 Jan** | Additional biome array stored with map output to show walls. | Implementation, Git Push |
| **30 Dec** | More biome and tile types implemented with better map generation. | Implementation, Git Push |
| **26 Dec** | Map generation foundation created with new biome transitioning. | Implementation, Git Push |
| **21 Dec** | More efficient entropy calculation implemented, project reformatting, and small fixes. | Improvements, Fix, Git Push |
| **20 Dec** | Discovered cause of time-consuming process being entropy calculation. | Discovery |
| **19 Dec** | Texture to biome conversion implemented. | Implementation, Git Push |
| **13 Dec** | **Online Meeting with Supervisor. [Notes Below]** | **Meeting** |
| **11 Dec** | Importing and exporting for all systems, optimising, refactoring, and info logging. | Implementation, Fix, Git Push |
| **22 Nov** | **Online Meeting with Supervisor. [Notes Below]** | **Meeting** |
| **22 Nov** | Implemented automatic rule generation for WFC based on canvas image. | Implementation, Git Push |
| **11 Nov** | Additional WFC rules implemented | Implementation, Git Push |
| **10 Nov** | WFC implemented with rules. | Implementation, Git Push |
| **5 Nov** | Uploading Unity project to git repository. | Git Push |
| **5 Nov** | Creating git repository. | Git Push |
| **Oct & Nov** | Research into WFC, Perlin noise, map generation in video games, Terraria & Noita reference material. | Research |
| **26 Oct** | **Project proposal.** | **Deadline** |
| **20 Oct** | **Initial In Person Meeting with Supervisor. [Notes Below]** | **Meeting** |

**Expanded Notes**

| Date | Event | Notes |
|---|---|---|
| **19 Apr** | Online Meeting | All Feedback for Report:<br>Add citations and references to everything.<br>Complete map generation literature review.<br>Complete map generation outcome evaluation.<br>Complete WFC outcome evaluation.<br>Do heuristic table for outcomes.<br>Rewrite areas in order of chronology e.g. diagonal section.<br>Refer to research questions in outcome evaluation.<br>Mention interactive design and UI.<br>Move user testing into conclusion.<br>All claims must be sourced.<br>Rewrite any areas that feel 'fluffy'.<br>100% doc size is print size so adjust to that. |
| **29 Mar** | Online Meeting | For Report:<br>- Delve more into the ethical research.<br>- Indicate the Terraria aspects of generation and how it can be transferred to other games.<br>- Investigate target audience / target users.<br>Give user guidance and accessibility for the unity project.<br>Video must:<br>- Be factual and informant.<br>- Show application and topic.<br>- Not miss key aspects.<br>- Not be too long or short.<br>- Not have bad audio. |
| **21 Feb** | Online Meeting | Reminder to start project log ASAP.<br>Idea to showcase map generation as rendered tiles with textures instead of colour per tile.<br>Look into making draft report soon for feedback before Easter. |
| **22 Jan** | Poster Presentation | Showcase to both supervisor and second marker. [Joshua and Louca]<br>Idea to sperate the 'permission layers' of tools between the developer and players. In which the developer can use this project as a tool and dictate the limits that the player can interact with the system.<br>MoSCoW objectives for report to indicate success of project.<br>Time of completion to create map generation is almost constant based on map dimensions (O(n) notation with 0.00000097 seconds/tile) indicating good optimisation. |
| **17 Jan** | Online Meeting | Showcase current poster before poster session.<br>Poster feedback. |
| **13 Dec** | Online Meeting | Investigate a save-as window for importing and exporting.<br>Share project documents with supervisor.<br>GANTT chart for progress update.<br>Poster tips and answers for questions. |
| **22 Nov** | Online Meeting | Determining the methodology of result conclusions.<br>Show that I am doing collection of unedited results for conclusion analysis.<br>Check through specification for conclusion grading.<br>Contingency time added to project plan.<br>Change project plan and mention it in the poster session.<br>Look into more PCG functions. |
| **20 Oct** | In Person Meeting | Presenting project idea to supervisor. |

**Appendix C: Assets used in the Project.**
Help With 2D Noise Function: https://www.youtube.com/watch?v=XpG3YqUkCTY.
And https://www.youtube.com/watch?v=wbpMiKiSKm8&list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3.
Additional Unity Packages Used: TextMeshPro.
No other visual, scripting, audio, or other assets we used.

**Appendix D: Mass examples of generation**