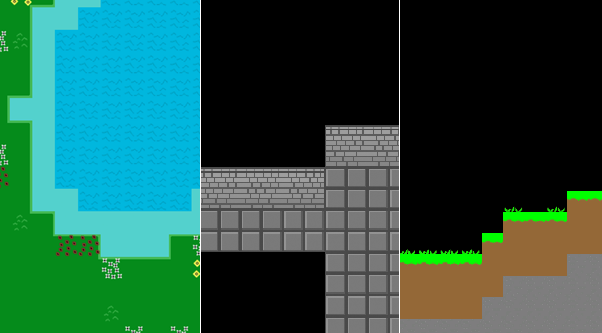
Teesside University

Computing Project

**(COM3051-N)**

Final Year Project: Report

Tilemap Generator



Aziz Arar (S6053935)

Contents

[Abstract 3](#_Toc7641012)

[Introduction 3](#_Toc7641013)

[Development and Implementation: Sprint 1 5](#_Toc7641014)

[Generation Window 5](#_Toc7641015)

[2D Top down 5](#_Toc7641016)

[Perlin Noise World Map 5](#_Toc7641017)

[Development and Implementation: Sprint 2 7](#_Toc7641018)

[Generation Window Updates 7](#_Toc7641019)

[2D Top down 7](#_Toc7641020)

[BSP Dungeon 7](#_Toc7641021)

[Perlin Noise World Map Updates 8](#_Toc7641022)

[Development and Implementation: Sprint 3 8](#_Toc7641023)

[2D Side scrolling 8](#_Toc7641024)

[Patch Notes 9](#_Toc7641025)

[Research 10](#_Toc7641026)

[World & Level Generation 10](#_Toc7641027)

[Perlin Noise 10](#_Toc7641028)

[Binary Space Partitioning 10](#_Toc7641029)

[Game Engines and Plugins 10](#_Toc7641030)

[Use of Level Generation in Games 11](#_Toc7641031)

[Terraria 11](#_Toc7641032)

[Enter the Gungeon 11](#_Toc7641033)

[Social and Ethical Considerations 11](#_Toc7641034)

[Testing and Evaluation 11](#_Toc7641035)

[Peer Feedback Received 11](#_Toc7641036)

[Conclusion 12](#_Toc7641037)

[Evaluation 12](#_Toc7641038)

[Future Plans 12](#_Toc7641039)

[Reflection of the Artefact 12](#_Toc7641040)

[References 13](#_Toc7641041)

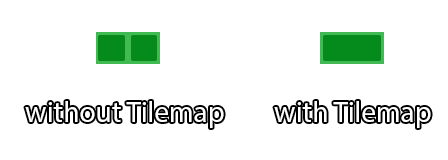
# Abstract

|  |
| --- |
|  |

# Introduction

The artefact is a Unity Editor Plugin which would randomly generate 2D maps using the Tilemap system. The reason for this was the decision to take advantage of the Unity 2017 feature as it allows tiles to be painted on a grid using a Palette of various Tiles. By including the extra features found on Unity’s Tilemap 2d extras [1] repo, it allows a further extension of how these tiles can interact. In the application, various Tile types are used in the examples, such as Terrain tiles, Rule tiles, animated tiles and Random tiles. The 2d extras is not necessary for my Tilemaps, but it creates it gives the illusion of a more natural environment.

This is an example of the advantages of using a Tilemap. The left side shows without a Tilemap, and the Right side a Tilemap with Grass Terrain Tiles.



A window will be visible from the options Menu [Aziz > Tilemap Generator] which would allow the user to select the type of level they want generated, such as a World Map and Dungeon. The user would decide the size of the grid, and the Tiles generated on it. There are extra options such as options to generate a collision layer and whether foliage should generate, and the density of it.

There are two creation modes: Simple and Advanced. Simple is the general user trying to generate a level easily. Advanced mode allows the user to remove Grid restrictions, and to change the Animation Framerate, Pixels per Unit and Cell Size of the Tilemap.

For the methodology of the project, Scrum was used as it allows me to split my tasks up into sprints, to allow me to focus and finish one thing at a time. For project planning, HacknPlan was used to plan my tasks. This allowed me to make sure certain features of the Tilemap Generator is up to scratch before ending the sprint. Whenever bugs were found during a previous sprint, it was added to a list. Any bugs on this list was returned to during the next sprint to attempt to be fixed.

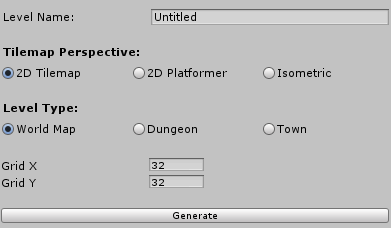
The reason I have decided to base my

Why this Artefact? The users.

# Development and Implementation: Sprint 1

## Generation Window

Firstly, to start the project off, a Basic window was created which will be used by the User so they can decide what they want generated. To do this, the Unity ‘Editor Window’ Class was used, this class included various helpful GUI Elements which enabled me to save time when putting this together.

This was the first draft of the Editor Window, the various GUI Layout Elements available that were used were, Labels, Selection Grids, a Text Field and Int Fields. At this time, while the visuals of the Generation Window was coming together it was not yet functional. The next step after this was the creation of the first type of generation to be included in the Artefact, which was the World Map using Perlin Noise.

The path of the Generation Window was located in the Unity Menu Bar > Aziz > Tilemap Generation. This way it should be immediately accessible by users.

## 2D Top down

### Perlin Noise World Map

Firstly, to start on the Perlin Noise, a decision needed to be made on how the Tilemap was going to be implemented. It was decided that 2 arrays would be used. An array of positions as a Vector3Int, and an array of Tiles as TileBases. TileBases in Unity Tilemaps is a specific Tile type, such as Grass or Water. The different tiles would be linked to a specific height when the heights get calculated.

The Tilemap ‘setTile’ function required Vector3Ints to determine the position, so was used instead of a preferred Vector2. Using Unity’s ‘Mathf’ class, assisted in the Perlin Noise generation as a function was already included. This function was used instead of creating a new version. However plans were put in place to create a new Perlin Noise function with additional noise settings.

A new variable named arrayLength was introduced which was the width and height of the grid multiplied. This was created and used to set the size of the two arrays decided previously. The start function reintroduced these variables as new arrays, to have fresh data put inside. The size of the the arrays was determined by the arrayLength variable.

A loop runs, which for every position in the array, it would run the Perlin noise function to determine what would generate there.

|  |
| --- |
| for (int index = 0; index < positions.Length; index++)  {  positions[index] = new Vector3Int(index % gridX, index / gridY, 0);  float height = Mathf.PerlinNoise((float)positions[index].x / 10, (float)positions[index].y / 10);  } |

As the application lacked any sprites, tiles were created, so there was a visible indication when testing if the Perlin noise worked correctly. There were four different tiles created in Photoshop. (See Figure 2)

https://raw.githubusercontent.com/Bodmass/AzizArarFYP/37d8d603c627acf270738ede87707ac608bc0a1f/Aziz%20Arar%20s6053935%20FYP%20-%20Tilemap%20Procedural%20Generation/Assets/Demo/Sprites/2DTopdownWorldMapSpritesheet.png?token=AGKIN57DKCYTZVMRWJ47J6S4ZRQZE

Figure

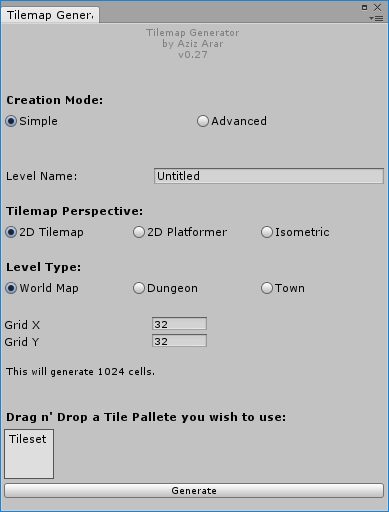
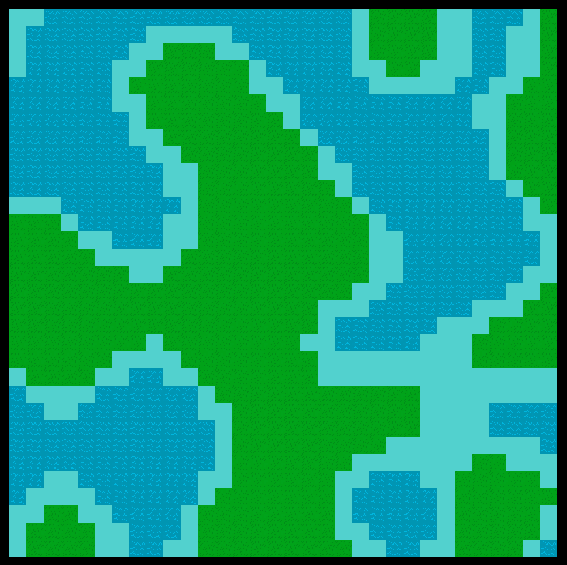
To avoid issues with copyright, new tiles were created, as tiles are planned be included and used in a Demo included in the final version of the Artefact. Once the new tiles were setup, they were usable by the Perlin Noise.

For each tile created, a TileBase variable was made in the Perlin Noise Generation Script, the types of tiles were: Shore, Grass, Water and Mountain respectively. A conclusion needed to be reached for the various the height for the water, and as it is a World Map, A high volume of water was desirable. The ratio of Land to Water was 60:40.

|  |  |
| --- | --- |
| Tile | Height |
| Water | 0f – 0.39f |
| Shore | 0.4f – 0.49f |
| Grass | 0.5f – 1.0f |
| Mountain | N/A |

By default the tiles were set to the Shore to ensure a Tile would appear in case of an error. The final heights were decided, and as the height is a float that ranges from 0 to 1. Mountains were included at the time in the Perlin Noise.

|  |
| --- |
| for (int index = 0; index < positions.Length; index++)  {  positions[index] = new Vector3Int(index % gridX, index / gridY, 0);  float height = Mathf.PerlinNoise((float)positions[index].x / 10, (float)positions[index].y / 10);      tileArray[index] = Shore;  if(height >.5f)  {  tileArray[index] = Grass;  }  else if(height >.4f)  {  tileArray[index] = Shore;  }  else  {  tileArray[index] = Water;  }  thisMap.SetTiles(positions, tileArray); |

This proved to be successful, and generated a Map on a 32x32 Grid. See Figure

After the implementation of the Perlin Noise, the Generation Window was updated and they were connected. This stopped the majority manual input of values into the script through the inspector.

The Generate button was given function, when pressed a Grid is created in the current scenes hireachy with a Tilemap attached to it. This Tilemap contains the Generation Script. By having the Perspective and Level Type filters, GUI.enabled was used to disable the Generate button if the Perspective and Level type was set to one other than 2D Topdown and World Map. A testing and feedback follow this stage so when it came to getting feedback from my peers, they were not be able to unintentionally cause errors.

An attempt implenetation of a drag and rop system for the Tiles was introduced, however was not able to get it functional in in time for the testing phase, so was left unused at this stage. At this time, it was required to maually apply the tiles in the inspector window. The Grid IntField was updated, locking from going below 0.

The initial plan with the drag and drop system was that there will be Tilesets which would contain the various different tiles that would list various different TileBases, for example: Water, Grass and Mountain. This will automatically connect to the Generator, assuming they are in the right position, and tagged correctly. The user would be able to create their own Tilesets, or use Unity’s Tile Palette.

A label was added beneath the Grid settings, which would give information. It would go between 2 errors, stating if grid was below 0, or if the grid was set over 256. If this was the case, the Generate button would be disabled. If they were not, it would display the amount of cells that would be generated.

# Testing and Evaluation (Part 1)

It was a vital to the artefact that it was tested and that feedback was recorded. The feedback received was applied to the project. During this phase, only two pieces of feedback was received by my peer, who gave feedback on the performance and feel of the generator. While it is early stages, I wanted to tackle the performance issues. My plan is to attempt to implement the algorithms for the level Generation first, then return to them later and improve upon them.

I have laid out a table which includes the feedback received, and at what version I received that at this stage in the process.

## Peer Feedback Received

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version | Feedback Received | Actions Taken |
| Callum Powley  (Games Programming) | 0.2 | “I think the generation is slower when I set the grid to a larger size, can you improve this” | I have attempted to implement a multi-threaded job for the Perlin Noise function. However at this time it requires more testing and tinkering to see if there is a performance increase. |
| “” | “” | “Can you make it so we can select the tiles as the Drag and Drop doesn’t seem to work, and I have to select them manually on the Inspector each time” | The Drag and Drop does not work at this time, but I plan to remove this and put an asset selection box instead. |

# Development and Implementation: Sprint 2

## Generation Window Updates

Firstly, as the primary interface for the User, the Generation Window needs to be easy and accessible to use for both new and advanced users. I decided to remove the Drag and Drop for the Tile Palette as it was cumbersome, and a more direct and clear Tile selection system needed to be created.

What replaced this was Objects. These, when listed, would link to a specific Asset file in Unity. By specifying the type to TileBase, this ensures that only Tiles, even those added by the 2d extras, will be the only selection visible and legal to enter this field. I created 4 Objects, these would be for the max amount of tiles that could be selected by the user. As the 2D Top down World Map has 4 different TileBases, this is how I decided the amount required. These objects are reusable and can be set to any Tile type when a certain Generation Script requires it.

Moving to Objects for each Tile from the Drag and Drop system allowed me to immediately push the Tiles to my Generation Scripts when created. This allowed for quicker testing.

## 2D Top down

### BSP Dungeon

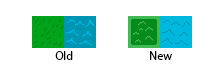
It was time to begin working on the second type of level generation in the project. I set out to implement Binary Space Partitioning (BSP) to create dungeons. The idea of BSP is to keep splitting a room, in this case a leaf, into smaller parts. Each leaf will them attempt to split itself, if the leaf can no longer be split, a room is created in its place. This creates a tree with various different branches which split into newer branches. A minimum and maximum size is determined and used check if the leaves can be split anymore. I initially began with a function to completely clear the Tilemap to a BG tile. Unlike the World Map generation, there may not always be a tile in a certain position, so I will be using this as a clear function on regeneration.

It was during this time, I realised it may be inefficient to rewrite many of the functions used in my World Map Perlin Noise generation here. I created a new script which would be the parent to all generation scripts. I moved many of the variables, which I believe is not specific to the Perlin Noise generation. These include variables which I now set as protected, such as the size of the grid size and the arrays for both the position Vector and tileArray. I also created virtual functions for Regeneration which can be overridden by both scripts that inherit from this, to make it specific to the generation type.

Firstly, returning to the BSP script, I followed steps similar to Timothy Hely, and Romain Beaudon in their approach to implementing BSP Trees into their project. I created a Leaf class, which would contain a variable of two leaves inside of them, which split to the left and right. I then created a BSP function, which will be used to split up the leaves into smaller leaves. At this time, a Rect is used to determine the size of the room. It would first check that these leaves do not already have leaves connected to them, and if the current size of the room is bigger, it would run a split function.

This split function would check whether it should be split horizontally or vertically. The split function will not run if the current leaf has already been split into two smaller leaves. I then made a Draw Function would check if the leaves under it. If it doesn’t, Tiles are placed between the minimum and maximum size of the Rect. This is a temporary solution, but I plan to remove all Rects and apply them to the tileArray instead. This worked successfully and I decided to make tiles as I was just using Grass and Water from my original Tile array. I went into Adobe Photoshop CC and created 2 new tiles which I used to be the floor and background. See Figure.

### Perlin Noise World Map Updates

Firstly, in my set of updates to the Perlin Noise World Map, I wanted to update the sprites and tiles included in my Demo to show how the tiles included 2d extras from Unity Technologies. The various different tiles I wanted to include was a Terrain tile for the grass, this will improve the overall look of how the grass connects together. The terrain tile allows you to create a sprite for each different position for the tile, which when you draw it automatically connects any nearby terrain tiles.

I also updated the tiles for Water by using an animated tile, which was a modification of the current water, and using the Flag effect in Adobe Premier Pro CC, then exporting 60 frames as PNGs.

Secondly, after receiving feedback about the level generator thus far, I wanted to improve the performance of the World Map generation. This is especially noticeable when you generate at bigger sizes, it takes longer for the Perlin Noise function significantly longer to run. I set out of a way to improve this and I came across the Unity Jobs class. An interface included in this called ‘IJobParallelFor’ allows for multiple threads to work in parallel to each other. Theoretically if I had the Perlin Noise work on multiple threads at the same time, it should significantly improve performance and loading time. I implemented a new job called Perlin Job. I managed to make the Job work, however using Deep Profiling, I did not notice a performance increase. I believe this is down to not fully understanding how to take advantage of the Jobs, but I decided to leave it in as it did not hinder performance.

# Testing and Evaluation (Part 2)

///Something first before engaging with the immersion of the generation.

I have laid out a table which includes the feedback received, and at what version I received that at this stage in the process.

## Peer Feedback Received

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version | Feedback Received | Actions Taken |
| Samantha Kelly  (Games Art) | 0.3 | “I found it easy to generate a level from the tiles you provided, but for the World Map generation, As an artist, I feel it lacked immersion, adding the option to generate flowers and trees would add this to a generated world. “ | The feedback is greatly appreciated, in an upcoming version of the plugin I plan to add foliage to generate. The user will be able to decide if they want Foliage to generate, and at what |

# Development and Implementation: Sprint 3

## 2D Side scrolling

# Patch Notes

As the project only reached Version 0.7, I will include the patch notes in a table below of at what stages I achieved certain tasks. A similar version of the Patch Notes is included in the Project Readme file.

|  |  |  |
| --- | --- | --- |
| Version | Type | Patch Notes |
| 0.7 | | |
| Additions | |  |
| Added Reset Button |
| Added Stone Generation with Height Option |
| Changes | |  |
| Updated 2D Platformer Generation Tiles |
| Moved Foliage Density to Simple Options |
| Tilemap Perspective Label "2D Tilemap" changed to "2D Topdown" |
| Tilemap Perspective Label "2D Platformer" changed to "2D Sidescroller" |
| Removals | |  |
| Isometric Selection (To be reintroduced, Project requires Unity Version Upgrade) |
| Town Selection (To be reintroduced) |
| 0.6 | | |
| Additions | |  |
| Added 2D Platformer Integration to Generation Window |
| Added Foliage option to 2D Platformer |
| Added Foliage Density Option in Advanced Options |
| Changes | |  |
| Grid Size on 2D Topdown now ensures X and Y are the same due to bug |
| 0.5 | | |
| Additions | |  |
| Added Foliage to 2D Topdown : World Map (Demo Includes Random Tile) |
| Added Walls Generation to 2D Topdown : Dungeon |
| Added Grid Cap removal in Advanced Settings (\*) |
| Changes | |  |
| All Tilemaps Generated now default at 60fps |
| Updated 2D Platformer Generation, also now places down Grass and Dirt |
| 0.4 | | |
| Additions | |  |
| Added Collision Generation (Water and Dungeon BG) |
| Added Early Version of 2D Platformer, using 1D Perlin Noise, NYI into the Generation Window |
| 0.3 | | |
| Additions | |  |
| Added Tilemap Generator Base Class, all Generation Scripts will Inherit from this. |
| Added 2D Topdown Dungeon Generation using BSP |
| Added Objects on the Generation Window for the Tiles |
| Removals | |  |
|  | | Removed Drag n Drop which did not work. |
| 0.2 | | |
| Additions | |  |
| Added 2D Topdown World Map functionality to Generation Window |
| Added Advanced Options with Cell Size and Pixel Per Unit options |
| Added Box for a Not Yet Implemented TileBase Drag n Drop |
| Added 2D Topdown World Map functionality to Generation Window |
| 0.1 | | |
| Additions | |  |
| Added Basic Generation Window |
| Added 2D Topdown World Map generation using 2D Perlin Noise |

# Research

## World & Level Generation

There are different types of ways games creation randomness, from having Worlds and Levels generate, and having pre-created sections randomly fit together. It is vital for the players experience, that these games generate in a natural and seamless way. There are many different methods of generating levels.

### Perlin Noise

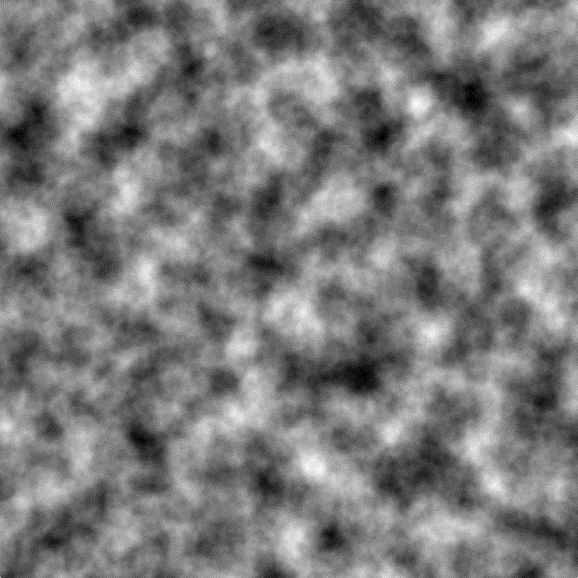
Firstly, Perlin Noise, developed in 1983 by Ken Perlin, is a type of gradient noise. The algorithm can be applied in different dimensions. 1D Perlin Noise generates what seems to be a line on a graph (See Figure). Which can be used for things like animation and, generating 2D hilly terrain. In 2 Dimensions, when generated, its appearance is a texture like static, with different heights/depths, depicted with the intensity of black and white. (See Figure 1). An example of it in use is generating top down outdoor levels using the lowest height for water, and the biggest height for mountains. Another example could be applying the noise to a texture or shader to achieve a dissolve or fire effect.

Figure ?

Perlin Noise can also be generated in the 3rd Dimension, however as we are working with Tilemaps which are exclusively 2D, we will not be using this dimension. I plan to use Perlin Noise for my 2D Topdown and Isometric World Map generation as I can run a Perlin Noise function for each position to determine a height, then save the height information for the Drawing Phase. I also plan to use Perlin Noise in 1 Dimension, this will allow me randomly generate heights to give a hilly effect on the generated level.

### Binary Space Partitioning

## Game Engines and Plugins

In this project I decided to use Unity 2018.2.15f1. This majority of the reason my Artefact was in Unity was to add more support for the Tilemap system which was added in Unity 2017. However, there are other game engines with support for a 2D Tile system, which I could have used instead. Unity has been continually supporting the creation of 2D games, with this system it makes creating 2D games easier as you can paint the tiles on, with different types of tiles available.

Unreal Engine 4 features Paper 2D, this is how Epic Games caters to the creators of 2D games. It allows the creation of Tile Sets and Tilemaps. It is possible to script random generation for the tile system; however, this system did not have an extension with different types of tiles unlike Unity’s Tilemap system.

(<https://docs.unrealengine.com/en-US/Engine/Paper2D>)

## Use of Level Generation in Games

### Terraria

World Generation

### Enter the Gungeon

Pre-created rooms linked together

# Social and Ethical Considerations

It is important to take into consideration the Social, Ethical and Political issues that could come with a project. As I am creating a Unity Plugin, I need to understand what goes into publishing an asset like this online. The Social and Ethical issues I have taken into consideration is the privacy which comes with receiving feedback from my peers. Each person I have received feedback I have outlined that their feedback will be used to improve my project, and their feedback will be stored and possibly used, and asking permission to reference them in a feedback report. For those I could ask I have received permission to take their feedback and apply it.

I need to ensure I comply with the Data Protection Act –

To ensure I do not come into contact with issues regarding Copyright, I decided to make my own tiles. The tiles will be available in the demo and will be released under CC-BY, allowing others to use these tiles if they wish, however must reference me as the creator of them.

# Conclusion

## Evaluation

## Future Plans

## Reflection of the Artefact

# References

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.

Compton, K. and Mateas, M., 2006, June. Procedural Level Design for Platform Games. In AIIDE (pp. 109-111). Prachyabrued, M., Roden, T.E. and Benton, R.G., 2007, June. Procedural generation of stylized 2d maps.

In Proceedings of the international conference on Advances in computer entertainment technology (pp. 147-150). ACM.

Unity Technologies. 2018. Unity - Scripting API: Tilemap. [ONLINE] Available at: https://docs.unity3d.com/ScriptReference/Tilemaps.Tilemap.html. [Accessed 21 January 2019]

TO REFERENCE

<https://flafla2.github.io/2014/08/09/perlinnoise.html>

<https://docs.unity3d.com/2018.3/Documentation/ScriptReference/Unity.Jobs.IJobParallelFor.html>