Aziz Arar

S6053935

Tilemap Procedural Generation

Contents

[Abstract 3](#_Toc7285818)

[Introduction 3](#_Toc7285819)

[Development and Implementation: Sprint 1 4](#_Toc7285820)

[Generation Window 4](#_Toc7285821)

[2D Topdown 4](#_Toc7285822)

[Perlin Noise World Map 4](#_Toc7285823)

[Development and Implementation: Sprint 2 6](#_Toc7285824)

[Generation Window Updates 6](#_Toc7285825)

[2D Topdown 6](#_Toc7285826)

[BSP Dungeon 6](#_Toc7285827)

[Perlin Noise World Map Updates 7](#_Toc7285828)

[Development and Implementation: Sprint 3 7](#_Toc7285829)

[2D Sidescrolling 7](#_Toc7285830)

[Research 7](#_Toc7285831)

[World & Level Generation 7](#_Toc7285832)

[Game Engines and Plugins 7](#_Toc7285833)

[Testing and Evaluation 7](#_Toc7285834)

[Conclusion 7](#_Toc7285835)

[Evaluation 7](#_Toc7285836)

[Reflection of the Artefact 7](#_Toc7285837)

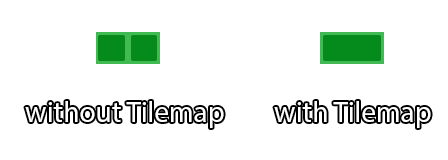
# Abstract

|  |
| --- |
|  |

# Introduction

My artefact is a Unity Editor Plugin which would randomly generate 2D maps using the Tilemap system. I have decided 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. I use various Tile types in my 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.

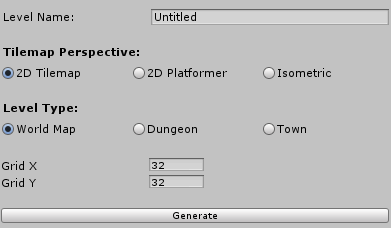
There are two creation modes: Simple and Advanced. Simple is the general user trying to generate a level easily. Advanced mode allows the user to remove Grid restrictions, and to change the Animation Framerate, Pixels per Unit and Cell Size of the Tilemap.

I have decided to use Scrum as the methodology of this Project as it allows me to split my tasks up into sprints, to allow me to focus and finish one thing at a time. I am using HacknPlan to plan my tasks. This allows me to make sure certain features of the Tilemap Generator is up to scratch before ending the sprint. If I find any bugs during another sprint, I can add them to a bug list.

Why this Artefact? The users.

# Development and Implementation: Sprint 1

## Generation Window

Firstly, to start the project off, I began with creating a Basic window which will be used so the User can decide what they want generated. To do this, I used the Unity ‘Editor Window’ Class, this class has various GUI Elements which will save time when putting this together.

This is the first draft of my editor window, using the GUI Layout Elements available to me, using Labels, Selection Grids, a Text Field and Int Fields I was able put this together. While it is not yet functional, I can begin working on the first part of the generation, which is the World Map using Perlin Noise.

## 2D Top down

### Perlin Noise World Map

To start on the Perlin Noise, I had to decide how I was going to connect it to the Tilemaps. I decided on using 2 arrays. An array of positions, and an array of Tiles. Unity’s ‘Mathf’ class has a Perlin Noise function inside of it which I decided to use.

Firstly, I set the size of the positions array by creating a new Vector3Int which was the width and height of the grid multiplied, with the size of the Tile Array copying the positions size. The Tile Array was an array of TileBase, which in Unity Tilemaps is a specific Tile type, such as Grass or Water. This way when the heights are calculated, this would be set to Grass for example, and when it sets the tile, it would know exactly what to set.

I then ran a loop, 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);  } |

I need to create tiles, so I can see if the Perlin noise is working correctly, so using Photoshop 4 different tiles. (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

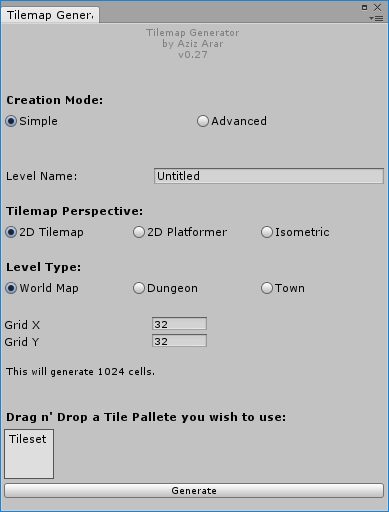
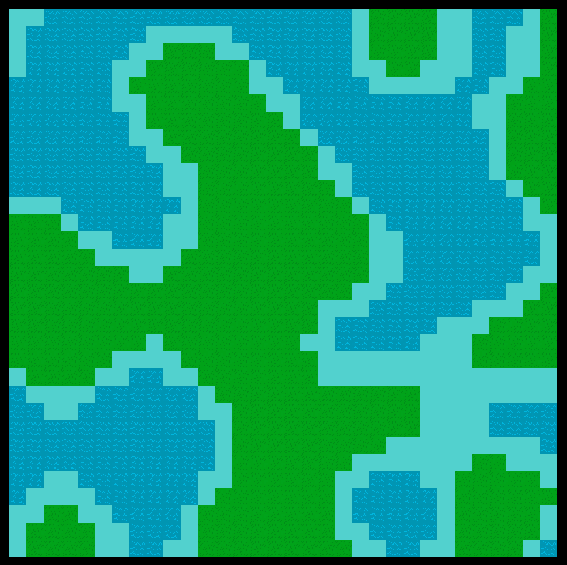
I created my own tiles to avoid any copyright issues, as the tiles will be used in a Demo included in the final version. Now I have the tiles setup, I can use them with the Perlin Noise.

I set up 4 TileBases in the Perlin Noise Generation Script, for Shore, Grass, Water and Mountain respectively. I needed to decide a height for the water, and as it’s a World Map, I wanted there to be a high volume of water. I decided to start with.

By default I set the tiles to the Shore to ensure a Tile would appear in case of an error. I decided to split it like this, as the Height is 0 to 1. I did not include Mountains at the time in the Perlin Noise.

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

|  |
| --- |
| 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, which looked like this.

At this point, I wanted to implement this into the Generation Window, as I had to manually input values and tiles for the World Map Generation to use.

I gave the Generate button function, creating a Grid on the Scene with a Tilemap in it. This Tilemap contains the Generation Script. By having the Perspective and Level Type filters I was able to disable the Generate Button if I hadn’t made it yet, so when it comes to getting feedback from my peers, they wouldn’t be able to unintentionally cause errors.

I wanted to Input a drag a drop system which would allow a set of tiles to be put in and sent together, However it does not have function at this time, so I still manually have to apply the Tiles. I also updated the Grid IntField, locking from going below 0.

I added a label beneath the Grid settings, which would give information. It would go between 2 errors, stating the grid is below 0 and when the grid is over 256. In this case, the Generate button would be disabled. If they are not, it would say the amount of cells that would be generated.

# Development and Implementation: Sprint 2

## Generation Window Updates

## 2D Top down

### BSP Dungeon

### Perlin Noise World Map Updates

# Development and Implementation: Sprint 3

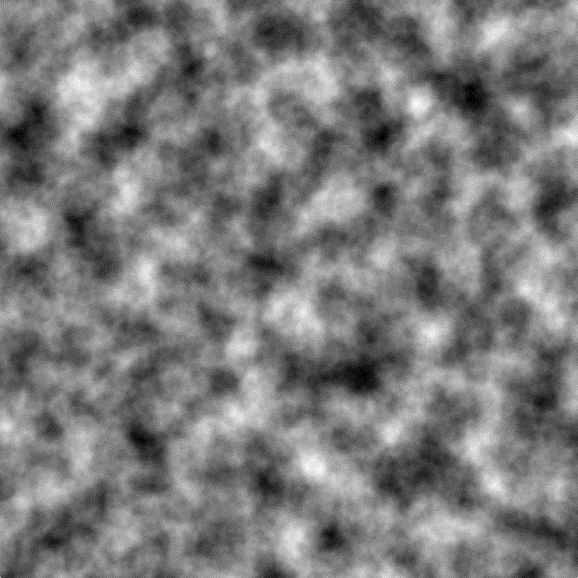
## 2D Side scrolling

# 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. When generated, its appearance is a texture like static, with different heights/depths, depicted with the intensity of black and white. (See Figure 1)

These heights allow us to easily generate a world map. If we decide what tiles will appear on certain heights by calculating the Perlin Noise at our position.

Figure 1

### 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 Pater 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

# Testing and Evaluation

# Conclusion

## Evaluation

## Reflection of the Artefact