Teesside University Computing Project **(COM3051-N)**

Final Year Project: Report

Tilemap Generator

Aziz Arar (S6053935)

Contents

Abstract

A plugin for Unity which allows users to generate levels onto Unity's 2D Tilemap system in different perspectives. The user can select the size of the grid, various settings for Collision and Level specific generation, as well as the tiles which are used. The development cycle follows the addition of new features which improved user immersion such as the generation of foliage, as well as the sacrifice of others.

Introduction

The artefact proposed was a Unity Editor Plugin which would randomly generate 2D maps using the Tilemap system. The reason for using the Tilemap system was to take advantage of the Unity 2017 feature, allowing 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 allowed 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. *(See Figure 1)*

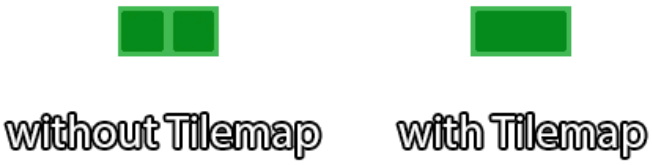

Figure 1 - A grass tile with and without a Tilemap Terrain tile.

A window is 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 was able to 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 implemented: Simple and Advanced.

Simple is designed for 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 allowed 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.

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. *(See Figure 2)*

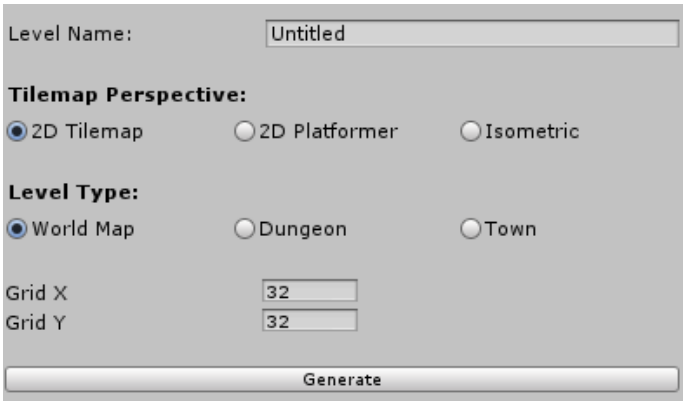

Figure 2 - Early Stage of Generation Window

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 added in the Unity Menu Bar > Aziz > Tilemap Generation. This way it should be immediately accessible by users. *(See Figure 3)*

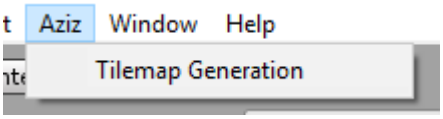

Figure 3 - Generation Path

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 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 4)*

Figure 4

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.

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. *(See Figure 5)*

Mountains were included 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

Figure 5 – Perlin Noise Heights

```
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 6)*

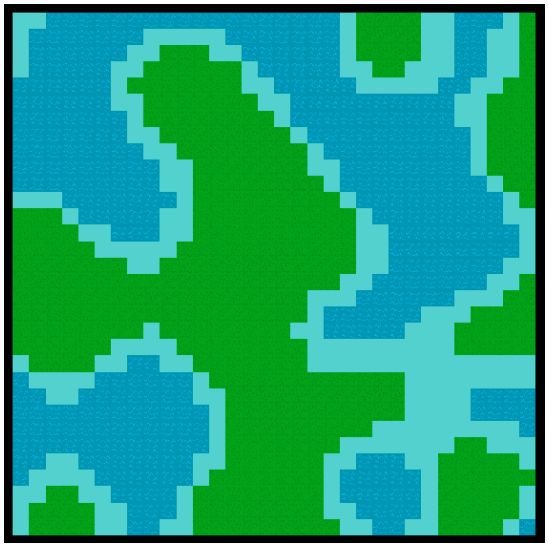

Figure 6

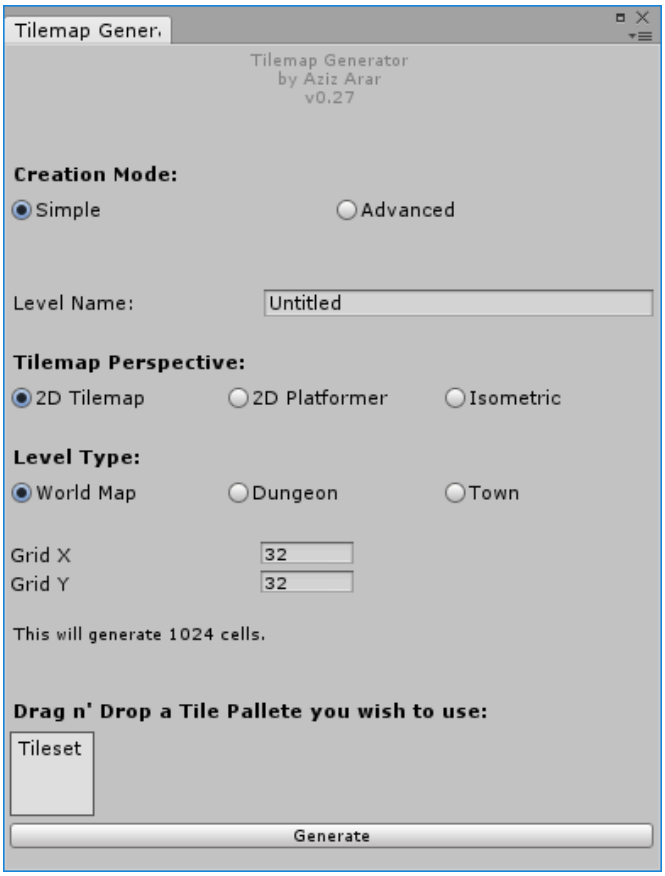

Figure 7 – The Generation Window during Version 0.27

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 attempted implementation of a drag and drop system for the Tiles was introduced, however was not able to get it functional 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. (See Figure 7)

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 was still at early stages, tackling performance issues would improve the generator when more features are added, and when levels become more complicated. The plan during this stage was to implement the algorithms for the level generation, then to return to them in a later sprint to improve it.

Peer Feedback Received

A table was created to chart all feedback at this time, which includes the feedback received, and which at version.

Action taken based on the feedback is included in the table.

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 does not 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 needed to be easy and accessible to use for both new and advanced users. The Drag and Drop for the Tile Palette was removed 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 ensured that only Tiles, even those added by the 2d extras, will be the only selection visible and legal to enter this field.

Four objects were created, these would be for the max number of tiles that could be selected by the user. As the 2D Top down World Map had four different TileBases, this is how the decision for the amount required was reached. These objects were reusable and could be set to any Tile type when a certain Generation Script required it.

Moving to Objects for each Tile from the Drag and Drop system permitted me to immediately push the Tiles to my Generation Scripts when created. This allowed for quicker testing. *(See Figure 8)*

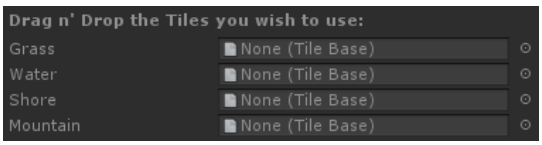

Figure 8 – Objects Update (in Dark Theme)

Tilemap Generator

During this stage, progress was made to make the Tilemap generation more effective. It would be inefficient to rewrite many of the functions used in my World Map Perlin Noise generation, so a new script was created which would be the parent to all generation scripts.

Many of the of the variables was moved into this especially those which were not specific to the Perlin Noise generation. These included variables which are set as protected, such as the size of the grid size and the arrays for both the position Vector and tileArray. Public virtual functions were created for Regeneration was overridden by both scripts which inherit from this, this made it regenerate specifically by the generation type.

```
public class TilemapGenerator : MonoBehaviour
{

    [SerializeField] protected int gridX = 32;
    [SerializeField] protected int gridY = 32;
    protected TileBase[] tileArray;
    protected Vector3Int[] positions;
    protected Tilemap thisMap;
    protected int arrayLength;
    public virtual void Regenerate()
    {

    }
}
```

```
public class TopdownWorldMapGenerator : TilemapGenerator
```

2D Top down

BSP Dungeon

It was time to begin working on the second type of level generation in the project. The purpose of implementing Binary Space Partitioning (BSP) was to create dungeons. Initially, a function to completely clear the Tilemap to a Background tile was created. Unlike the World Map generation, there was a chance a tile would not spawn in the same position and overlay on previously generated rooms. I used this as a clear function on regeneration.

I followed steps like Timothy Hely, and Romain Beaudon in their approach to implementing BSP Trees into their project, which greatly assisted in the creation of the BSP Tree generation. A Leaf class was created, 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 when used splits up the leaves into smaller leaves. A Rectangle 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 the split function.

This split function would check whether it should be split horizontally or vertically. The split function would not run if the current leaf has already been split into two smaller leaves. I then made a Draw Function. This would check if the current leaf has leaves under it. If it did not, Tiles are placed between the minimum and maximum size of the Rectangle.

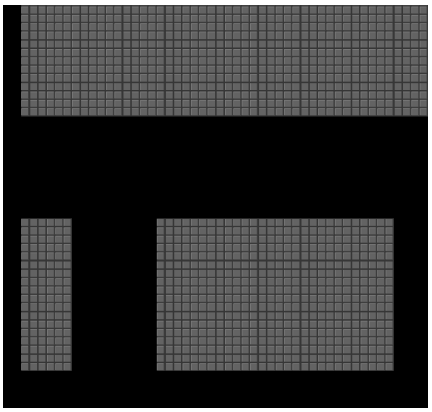

Figure 9 – BSP Rooms Generating

This was temporary solution, but a planned overhaul is in place to remove all Rectangles 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 9)*

The generation would not be complete without corridors. With assistance, the implementation was successful. The corridor is created between two leaves. A random point in each room is picked. From the furthest left point.

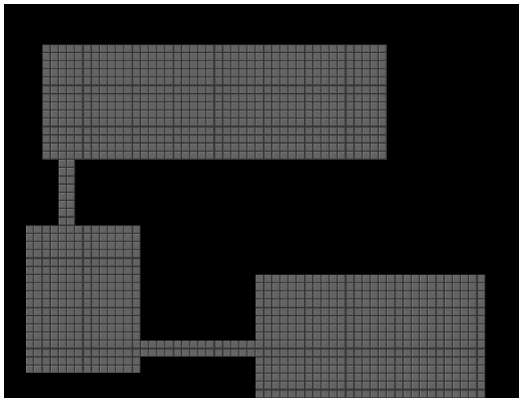

Figure 10 – Corridors connecting leaves

The code checks whether the points are aligned horizontally. If they are, a Rectangle is added going up or down depending on the result of the two points subtracted, indicating a direction. Otherwise, a Rectangle is placed to the right of the current point, and more Rectangles are created from there. The Rectangles are stored within a List. A tile is placed between the minimum and maximum points in each Rectangle to draw the Corridor. *(See Figure 10)*

World Map Updates: Part 1

Firstly, in the set of updates to the Perlin Noise World Map, an update the sprites and tiles was made for the Demo which is included in the Artefact to show how the tiles included 2d extras from Unity Technologies improved immersion in the Tilemap Generator. The various tiles used in the updated Demo include a Terrain tile for the grass, this improved the overall look of the demo by the grass tiles automatically connecting.

The tiles for Water was replaced with an animated tile, this was a modification of previous water tile, after being taken into Adobe Premier Pro CC and with a Flag effect applied. It simulates water moving. Exporting this at 60 frames as PNGs, allows it to be directly put on an animated tile and to run.

Figure 11 – Tile Update

(See Figure 11)

Secondly, some action was taken from the previously obtained feedback about the level generator at this phase. An attempt to increase performance was taken to improve load times of the World Map Perlin Noise generation. This was especially noticeable when you generate at bigger sizes, as it took the Perlin Noise function significantly longer to run.

After some research, I found the Unity Jobs class which contained an interface called 'IJobParallelFor', this allowed for multiple threads to work in parallel to each other. Theoretically if the Perlin Noise was working on multiple threads at the same time, it would significantly improve performance and loading time, a new job under this interface called Perlin Job was created.

```
public struct PerlinJob : IJobParallelFor
{
    public float size;
    public int gridX;
    public int gridY;
    public int offsetX;
    public int offsetY;


    public NativeArray<Vector3Int> positions;
    public NativeArray<float> heights;
    public void Execute(int i)
    {
        Vector3Int pos = positions[i];
        pos.x = i % gridX;
        pos.y = i / gridY;
        float height = Mathf.PerlinNoise(((float)pos.x / size) + offsetX,
((float)pos.y / size) + offsetY);

        positions[i] = pos;
        heights[i] = height;
    }
}
```

Testing and Evaluation (Part 2)

After the implementation of the Perlin Job, and some tests were made using Deep Profiling. The tests taken did not seem clear if the Perlin Noise yielded performance increase. However, as this was my first implementation of this Job, I put down to not fully understanding how to take advantage of the Jobs.

After taking off Profiling, it seemed the levels generated with the Perlin Job was smoother, to confirm this another test was taken. This test was made to record the time it took from pressing Regenerate to the updated level appearing. This test yielded promising results. The Perlin Job did seem to give a performance increase. The tests were not done on smaller grid sizes as they would generate instantly. *(See Figure 12)*

Without Perlin Job					**With Perlin Job**				
Grid Size	**Time 1**	**Time 2**	**Time 3**	**Mean**	**Grid Size**	**Time 1**	**Time 2**	**Time 3**	**Mean**
128x128	1.20	1.04	1.02	1.08	128x128	0.69	0.71	0.72	0.71
256x256	4.00	4.32	3.78	4.03	256x256	2.31	2.41	2.45	2.39
512x512	15.24	13.54	10.72	13.16	512x512	9.95	8.96	8.70	9.20

Figure 12

*The tests were run on the following system: Windows 10 Pro 64bit. AMD FX 8320 (@4.0GHz). 8GB DDR3 RAM (@1600MH) and NVIDIA GTX 750Ti (@2GB)

Peer Feedback Received

During this stage, feedback was recorded again, specifically about the immersion of the World Generation. The level generation only generates levels with Grass, Water Again, I have laid out a table which includes the feedback received, and which at version.

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 benefit the generated worlds. “	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 and 4

2D Top down

World Map Updates: Part 1

In this update of the Artefact, measures were taken to improve immersion based on previous feedback. Two variables were created, a float named foliageDensity and a Boolean named GenerateFoliageLayer, defaulted to false and A foliage generation function was added to the World Map Generation script.

This would run after the Perlin Noise function, if GenerateFoliageLayer was set to true. If true, for each tile a Random.Range function would run, obtaining a value between 0 and 1. This allowed for a density of the foliage to be decided. If set to 0.6, the chance of foliage generating on that tile would be 60%. For testing a foliage grass tile was created.

As this worked, creating a Random Tile for the foliage with various new sprites for the flowers achieves a more natural looking world generation. *(See Figure 13)*

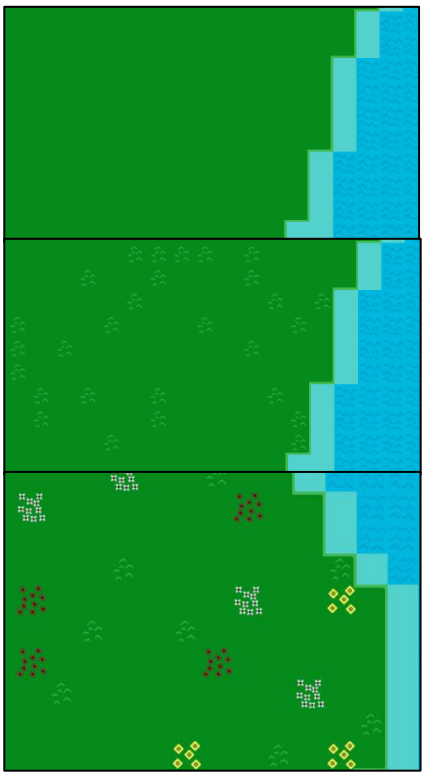

Figure 13 – Various stages of Foliage Development

Dungeon Update

A system like the foliage was added in the Dungeon Generation script. The purpose of this was to place Wall tiles on a BG Tile which is a floor tile beneath it. In the demo, this tile was set as a Rule tile to create a seamless wall. This can be enabled and disabled inside the generation window.

2D Side scrolling

In this stage of the Artefacts development, we were in a different perspective. From 2D Top down to 2D Side scrolling. With previous research, the implementation was already planned to use 1D Perlin Noise to achieve a World Generation like Terraria. For reference, a backup of Hugo Elias' paper on Perlin Noise was used. Arend Peter uploaded a copy to his website as the original is no longer accessible.

Firstly, to implement this, a new generation script with a single TileBase was created. This was a Dirt Tile. A Perlin noise function was created, which for every tile in the X axis,

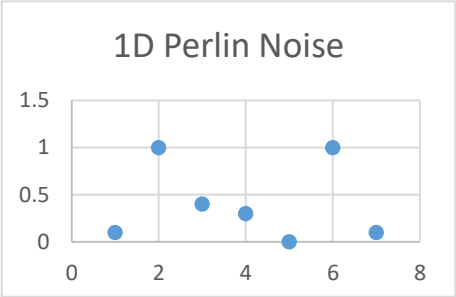

Figure 14 – 1D Perlin points randomized

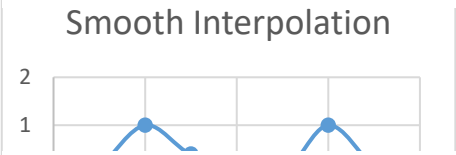

Figure 15 – Interpolation between 1D Perlin Points

would calculate noise. Firstly, a detail per chunk variable was implemented to decide the distance between the samples of the noise.

The lower this variable a hillier the terrain would generate, as there was not as many tiles between the two samples. The higher this variable, the more tiles between each sample, causing a smoother transition between samples. *(See Figure 14 and 15)*

The index was obtained by the division of our x value by the detail per chunk variable. We checked where we are in the chunk by getting the remainder of "[X % Detail Per Chunk] and dividing it by Detail Per Chunk, cast as a float.

Each sample is a random value calculated by using the seed, if the index and seed was the same between generations, it returns the same value every time. The importance of as having a seed which the user can reuse to get to a level placement was desirable and a feature enjoyed amongst users of level generators.

Figure 16 – 1D Perlin Noise implementation

Two samples were made for the current chunk, which calculated a random set of values by the seed. and the following chunk. Depending on where we are, a left or right sample is used, and possibly a level in between, which is more likely the higher the Detail Per Pixel value is.

Once the noise was calculated, another loop gets executed, for every value in the Y axis up to the noise. For each loop, tiles are placed down on the Tilemap at these locations. See Figure.

Figure 17 – Grass Layer Added

An improvement was made to replace the top layer of tiles to give the illusion of grass. While there was not a grass texture at this time, an undocumented and unused Tile I created was used instead.

This was further improved an added variable 'StoneDepth'. It can only generate on Dirt, and after a certain number of tiles deep, Stone will generate.

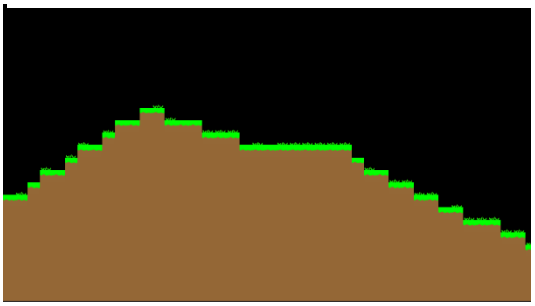

Figure 18 – Updated Grass

The final touches made to this generation at this stage was the inclusion of foliage. This was like the previous implementation of Foliage found in the Perlin Noise World Map Generation. Instead being generated on grass tile, the foliage is instead generated on the tile enough, if it is empty. Additional tiles were created for foliage, grass and stone. Levels generated have a more complete feel.

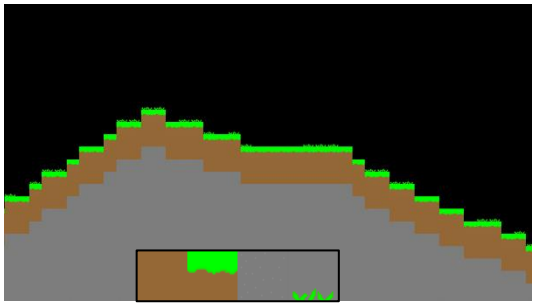

Figure 19 – Stone and Updated Tiles

Minor Updates

In this final version of the sprint, due time constraints, the planned Isometric version and Hexagonal tiles. The generation window was updated to reflect this, as well as the minor cleaning up and renaming of Labels which were previously incorrect.

A reset button was added to set all the values back to default. During this time, unused code was removed, code was refactored, and comments were made more clearly. *(See Figure 20)*

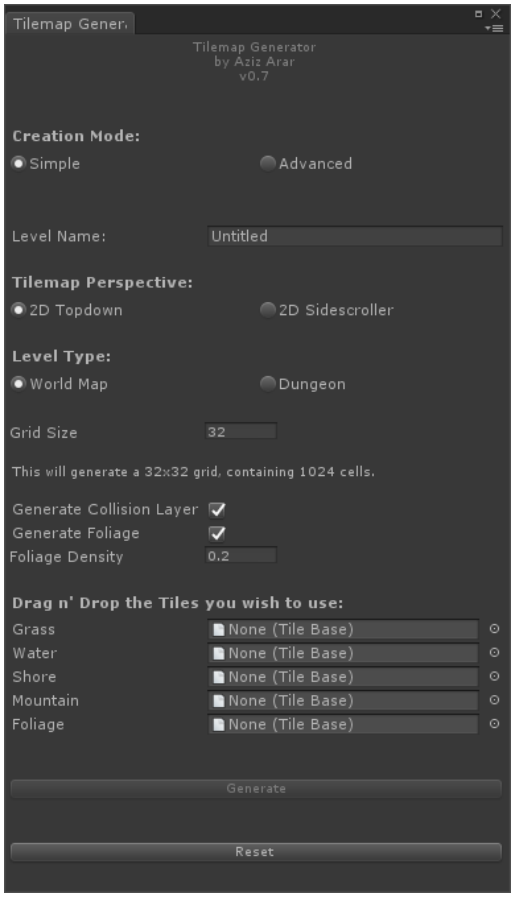

Figure 20 – Version 0.7 (Latest) Generation Window

Extended 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 yet simplified 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
Changes		
		Perlin Noise updated with Perlin Job, yielding a positive performance increase.
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

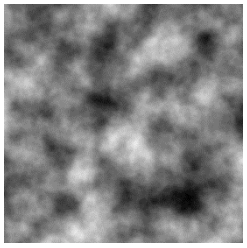

Figure 21

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 15)* 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 21*). 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.

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

The idea of BSP was to keep splitting a room, in this case a leaf, into smaller parts. Each leaf attempted to split itself, if the leaf can no longer be split, a room was created in its place. This created a tree with various branches which split into newer branches. A minimum and maximum size is determined and used check if the leaves can be split anymore.

Recursively, the space is partitioned, either horizontally or vertically. *(See Figure 22)*

Figure 22, A BSP Example from the 1st to 4th Iteration

Game Engines and Plugins

Unity Engine

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.

Since the introduction, it has been continuously supported. Adding features such as an Isometric Tilemap (Introduced in 2018.3), and Hexagonal tiles (introduced in 2018.2). On the 18th March 2019, an article released by Alice Hilton-Jones, a Technical Evangelist for Unity Technologies, was released. This was during the development of my Artefact, which helps further supports Unity's interest in the continued development of the Tilemap system. With the release of Unity 2019, it will not be long before more features are added.

2D extras

Unity Technologies additionally supports their Tilemap system introduced 2017 with their GitHub repository '2d extras', this includes Tiles and Brushes for the Tilemap system. These all benefit the Tilemap system as it makes creators of 2D content have an easier time drawing levels out.

As we are only placing down tiles, the Brushes included will not be used, however the various amounts of tiles will be. Different types of Tiles I plan to use are Animated, Terrain, Random, Rule, Hexagonal and Isometric.

These tiles are quite self-explanatory, but the Terrain tile makes us decide the sprite to be used in each position. And the rule tile allows a certain sprite to be placed when the tile is connected in a certain way.

Unreal Engine

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. While due to not being as light weight as Unity and lacks support from Epic Games.

Existing Applications

The only existing application which I could find which that rival the Tilemap Generator I proposed was Strata by Matt Mirrorfish. It follows the same concept of procedurally generating levels, with being native to Unity's 2D Tilemap system. It was added to the Asset Store, with its latest

update in December. Strata is strongly classic roguelikes and features Top down and Side scrolling generations.

Rivalling this, I would need to attempt to implement features which this does not have but learn from the competition about how they went about their generation.

The Target Audience and Rationale

The target audience for this project is those interested in the procedural generation of 2D content in games. The proposed plan for this Artefact is a level generation plugin for Unity, which will allow for quicker creation of levels using the Tilemap system. It possibly could work and in and with levels previously created in the Tilemap.

The reason I planned my project around this system, was because I primarily focus on 2D games, and personally work on multiple 2D projects. Trying to create levels from hand can be time consuming and trying to keep it feeling random and unique is important. Having levels generated based on certain filters set will save time and give some creative assistance with the randomness of the levels.

Project Constraints

The constraints regarding the process is my limited knowledge with Level Generation as this is my first time learning about these algorithms and having to learn and adapt them to the Tilemap system may prove to be difficult. However, I should not be constrained significantly if adequate time is taken in understanding different methods of Level Generation.

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 2018, which controls how the personal information obtained in feedback is used. Those providing feedback will be informed with how their data is being used, and only keeping the data recorded for as long as it is necessary. People who have given feedback will be allowed to access to the personal data I have recorded of them and request the removal of their personal data.

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

The Artefact is a working level generation tool inside Unity, the plugin is available as a Source file and as a Unity Package. The Unity Package can be imported into a Unity Project and use. To access the demo immediately, using a version of Unity 2018.2 is recommended. Using a different version of Unity requires the 2d-extras files to be removed and replaced, with the branch following the version.

The current state of the Artefact allows levels to be generated in 2D, in both top down and side scrolling perspectives. The size of the grid and tiles can be selected by the user, additionally, the option to generate Collision Maps, Foliage and Walls (for the Dungeon). Advanced options are available to change the size of cells, pixels per unit and Tilemap framerate, which defaults at 60fps.

A demo included in the Artefact contains tiles for Animated Water, Rule Wall tiles, Terrain grass and Random foliage by taking advantage of the 2d extras features, however is not required for the application to function.

The Artefacts reached Version 0.7 in its development cycle. This was at the sacrifice of Isometric and Hexagonal Tiles, and other planned additional features.

Future of the Artefact

The Tilemap still has ways to go before I would begin considering it finished. There are various features and changes which I am planning to introduce following the submission of the Artefact.

These would include:

- Fixing currently known issues
- Updating Unity Version to 2018.3 for the Demo
- Isometric Tilemap
- Hexagonal Tiles for 2D Top down World Map
- Introducing Unity Jobs for the other generation scripts, currently only implemented on 2D Top down World Map.
- Tree generation for Perlin Noise generated maps.
- Research into what drives exploration in worlds and adapt the generation to cater to this.
- Implementing procedurally generated terrain when in a certain range.

Reflection of the Artefact

Now the Artefact processes is over, I feel that I have done an adequate job on it. The beginning of the process was easier, as a fresh project, I was more motivated to try to get as much work done as possible. Starting the initial steps earlier allowed me to do early testing phases and get

feedback, However I failed in communicating with my peers, so I only got feedback two different people throughout the processes.

In the early phases, I found the initial implementation of the Generation Window and 2D Perlin Noise functions straight forward. The circumstance that Unity had its own 2D Perlin Noise function meant I did not have to fully understand in detail what would happen in the function, but understanding its results was important. Once I implemented this, progress on the Artefact slowed down.

During the process, it is visible that I struggled with managing my time, when working between the Artefact, and another University project, and with motivation. When I struggled on a certain part of the process, I found it easier to neglect the Artefact. My first major struggles were attempting to introduce Binary Space Partitioning into the Level Generator, and I felt that the initial research lacked significantly, so I had to go back and attempt to refresh my knowledge my reading different articles and following various tutorials.

Recording information such as the feedback and tests for functions such as the Perlin Job speed test, made me better understand the extensive testing features may need, as I did not initially see a performance increase when using the Profiler.

What I could do better was better managing my time and attempting to stick better with the sprints. I initially planned out my sprints and tasks in Hack n Plan, however I did not strictly follow them correctly, which caused me to run out of time near the end, as two features were not included.

In hindsight, using the competition, Strata as motivation to build a better and more complete Artefact would have been important to getting more work done during the slower periods. When compared to Strata, my Artefact lacks the ability to save out levels, and includes different types of generation methods I did not research such as Cellular automata.

A negative side effect of using Unity's predefined function for 2D Perlin Noise was that I struggled again with the implementation of 1D Perlin Noise, as I was not sure how I would achieve certain results. I was inspired by the level generation in Terraria and spending a good amount of time seeing various blog posts and video tutorials on Perlin Noise, it became much clearer on how I would achieve the desired effect.

The Isometric Tilemap and Hexagonal tiles were things I wanted to cover in the Artefact as they were not being used in any Tilemap generation project I could find. The lack of these generation methods makes the Artefact feel incomplete, as the original plan was to have these included.

As Unity 2018.3, where the introduction of Isometric tiles in the Unity Tilemap system released at the end of 2018, I did not have enough knowledge about the implementation of it. Being unprepared caused me to use the incorrect Unity version throughout the Artefact as I would not have been able to implement Isometric tiles in the version I had used, which was Unity 2018.2.15f1.

Leaving a Unity version upgrade to the end of the project did not seem to be a wise idea, as there may have been unforeseen issues with a version change. As little time was left before the end of the Artefacts process, I decided not to implement it.

While going through these various tests, I realised other issues such as a rotation bug with Water Tiles that would occasionally happen. I also encountered a regeneration bug when having Foliage enabled, that caused the level to not regenerate, but I could not reproduce this.

To conclude, I am generally pleased with the results of the 2D Top down World Map and the 2D Side scrolling Level, as they feel like an actual world is being generated. Especially with the tiles I have included which take advantage of the Tilemap, it creates a more immersive and pleasing generated level. Due to my lack of knowledge and an incredibly rough implementation based on previous implementations, I feel that I did not achieve an immersive dungeon generation experience. More research into Roguelike Dungeons and BSP is needed to possibly re-implement this, with a greater focus on Roguelike.

If I was to do a similar project again, more time would have been taken in the earlier phase for research and planning of the project. I would stick more closely with Hack n Plan and come up with a well thought Version system. Spending more time earlier researching would have also ensured when coming to implement certain ideas to a system, it would be easier to code and move onto later stages sooner.

By going through this process has given me appreciation of what it takes to go through an applications process beginning to end, and how important each stage is. Having enough research to see the project through is important, as it gives a clear understanding of how various systems work. Planning sprints and going through various Design and Implementation stages allows reflection on how each sprint went, and more controlled testing phases in between.

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.

MIRRORFISH, M. (2018). STRATA EASY 2D LEVEL GENERATOR - ASSET STORE. [ONLINE] ASSETSTORE.UNITY.COM. AVAILABLE AT: HTTPS://ASSETSTORE.UNITY.COM/PACKAGES/TEMPLATES/SYSTEMS/STRATA-EASY-2D-LEVEL-GENERATOR-126103 [ACCESSED 18 APR. 2019].

BIAGIOLI, A. (2014). UNDERSTANDING PERLIN NOISE. [ONLINE] AVAILABLE AT: HTTPS://FLAFLA2.GITHUB.IO/2014/08/09/PERLINNOISE.HTML [ACCESSED 23 APR. 2019].

H, ELIAS (N.D). PERLIN NOISE. [ONLINE] AVAILABLE AT: HTTP://WWW.ARENDPETER.COM/PERLIN_NOISE.HTML [ACCESSED 25 APR. 2019].

GOV.UK. (2018). DATA PROTECTION. [ONLINE] AVAILABLE AT: HTTPS://WWW.GOV.UK/DATA-PROTECTION [ACCESSED 29 APR. 2019].

HELY, T. (2013). HOW TO USE BSP TREES TO GENERATE GAME MAPS. [ONLINE] GAME DEVELOPMENT ENVATO TUTS+. AVAILABLE AT: HTTPS://GAMEDEVELOPMENT.TUTSPLUS.COM/TUTORIALS/HOW-TO-USE-BSP-TREES-TO-GENERATE-GAME-MAPS--GAMEDEV-12268 [ACCESSED 30 APR. 2019].

BEAUDON, R. (2018). RANDOM 2D DUNGEON GENERATION IN UNITY USING BSP (BINARY SPACE PARTITIONING) TREES - ROMAIN BEAUDON. [ONLINE] ROMBDN.COM. AVAILABLE AT: HTTP://WWW.ROMBDN.COM/BLOG/2018/01/12/RANDOM-DUNGEON-BSP-UNITY/ [ACCESSED 30 APR. 2019].

UNITY TECHNOLOGIES. (2019). UNITY - SCRIPTING API: TILEMAP. [ONLINE] DOCS.UNITY3D.COM. AVAILABLE AT: HTTPS://DOCS.UNITY3D.COM/SCRIPTREFERENCE/TILEMAPS.TILEMAP.HTML [ACCESSED 1 MAY 2019].

UNITY TECHNOLOGIES. (2019). UNITY - SCRIPTING API: IJOBPARALLELFOR. [ONLINE] DOCS.UNITY3D.COM. AVAILABLE AT: HTTPS://DOCS.UNITY3D.COM/2018.3/DOCUMENTATION/SCRIPTREFERENCE/UNITY.JOBS.IJOBPARALLELFOR.HTML [ACCESSED 1 MAY 2019].

GITHUB. (2017). UNITY-TECHNOLOGIES/2D-EXTRAS. [ONLINE] AVAILABLE AT: HTTPS://GITHUB.COM/UNITY-TECHNOLOGIES/2D-EXTRAS [ACCESSED 1 MAY 2019].

ESQUERDA, L. (2013). DUNGEON GENERATION USING BSP TREES | ESKERDA.COM. [ONLINE] ESKERDA. AVAILABLE AT: HTTPS://ESKERDA.COM/BSP-DUNGEON-GENERATION/ [ACCESSED 2 MAY 2019].

EPIC GAMES INC. (N.D.). PAPER 2D. [ONLINE] AVAILABLE AT: HTTPS://DOCS.UNREALENGINE.COM/EN-US/ENGINE/PAPER2D [ACCESSED 3 MAY 2019].

HINTON-JONES, A. (2019). ISOMETRIC 2D ENVIRONMENTS WITH TILEMAP – UNITY BLOG. [ONLINE] UNITY TECHNOLOGIES BLOG. AVAILABLE AT: HTTPS://BLOGS.UNITY3D.COM/2019/03/18/ISOMETRIC-2D-ENVIRONMENTS-WITH-TILEMAP/ [ACCESSED 2 MAY 2019].