Procedural Content Generation

Design Documentation

Christopher Ross

ROS16626280

CGP3017M

Contents

Christopher Ross
ROS16626280

1 Design

Pirates Cove, the name chosen for the developed game, was designed to be as simple as possible, yet very effective in the ways that it generates different objects each time.

1.1 Base Design

The project began by first designing the theme of the game to ensure that it meets the assignment specifications. This involved discovering where the game should take place, if it is land or water based, whether the game should be interactive, and what the main game objects are.

As seen in figure 1, many ideas were drawn on paper and each them were heavily scrutinised in terms of what advantages they may bring and how much time would be required for their implementation.

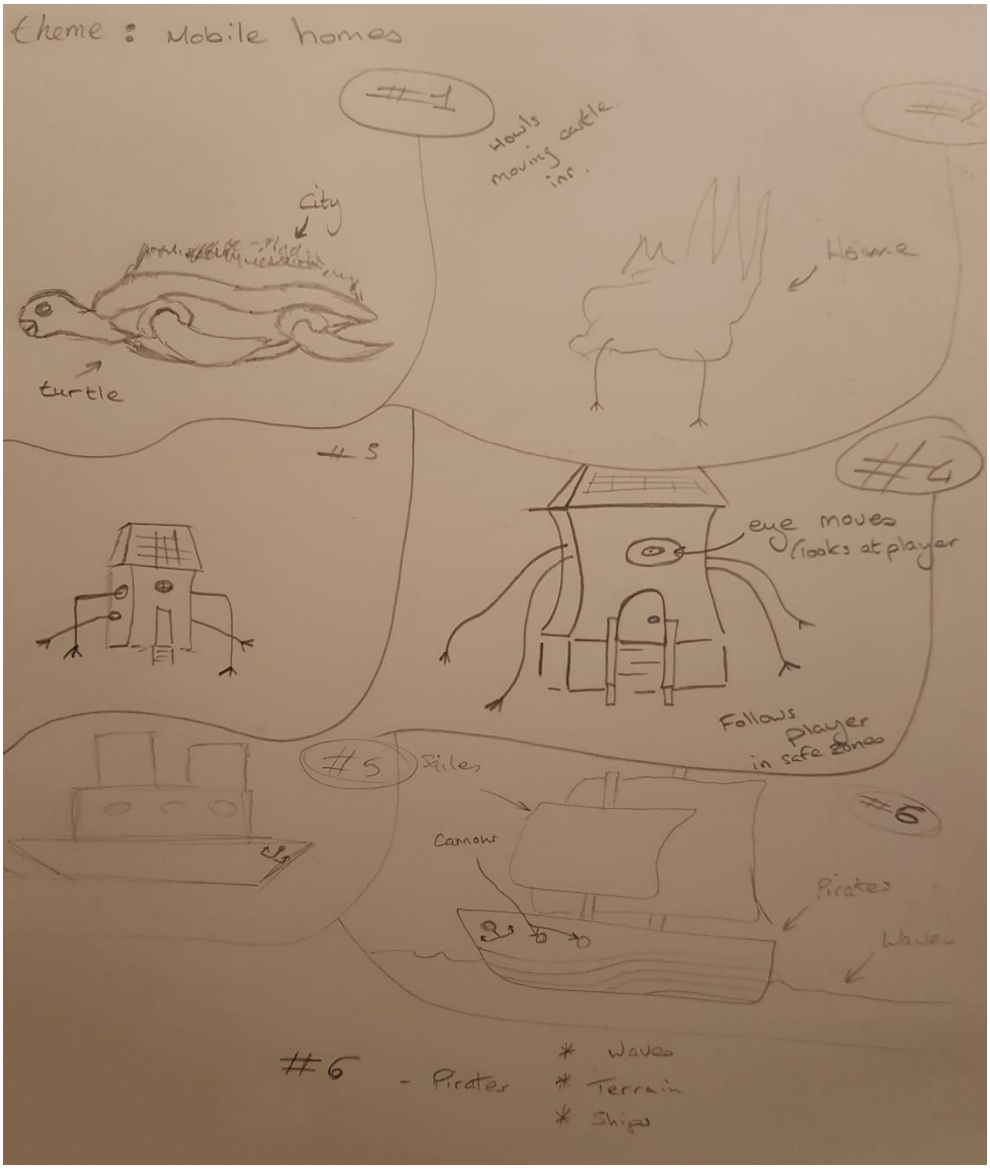

Figure 1: Theme Design

Christopher Ross
ROS16626280

The first idea (#1) was greatly inspired by World of Warcrafts (Blizzard, 2004) Shen-zin Su, an island-sized turtle that held the native villages and towns of the Pandaren race. This idea would have required a great amount of time to model, design and then implement other features as the turtle, being the size that it is, would require a high poly model meaning the rest of the models would also need to be high poly to maintain consistency.

The second idea (#2) was inspired by Studio Ghibli's Howls Moving Castle, however like the first idea, this would require high poly models which drastically increases the time frame.

The third (#3) and fourth (#4) ideas were discarded due to an insufficient number of ways to add procedurally generated content and the object would be only one piece rather than multiple pieces. Overall this idea would not have brought much to fulfil assignment specifications.

The fifth (#5) idea was going to use modern battleships however this then evolved into the sixth (#6) and final idea which was greatly inspired by World of Warcraft - Battle for Azeroth's Kul Tiras / Proudmoore ships (Braddock, 2018). As shown in figure 2, these ships have the potential to procedurally generate countless objects on the deck, sides, interior, and surrounding them making these objects capable of doing numerous activities including having battles between one another or other game objects.

Figure 2: World of Warcraft - Battle for Azeroth - Tidal Ship (Braddock, 2018)

As the sixth idea was chosen, a number of additional elements were also required and can be seen at the bottom of figure 1. Waves, water, and terrain were chosen to be implemented as it was decided that the game would be water based, terrain was added to potentially allow ships the ability to traverse from land to land. Finally, a possible pirate theme meaning that the aim was to have fleets, potential battles, and sinking ships.

1.2 Early Objects Design

Once the initial base design of the game had been finalised, the design and implementation of various game elements began. Initially, there were four elements that were to be developed and each would be implemented in a specific order based on requirements. For example, the terrain was developed first, and the sea was developed after as without land there would be no water and therefore the sea requires land. This was done to better visualise the setting of the scene, placements, and the boundaries of elements within the game.

The initial objects were the terrain, water, rocks, and trees. The design of these elements was visualised through sketches as seen in figure 3.

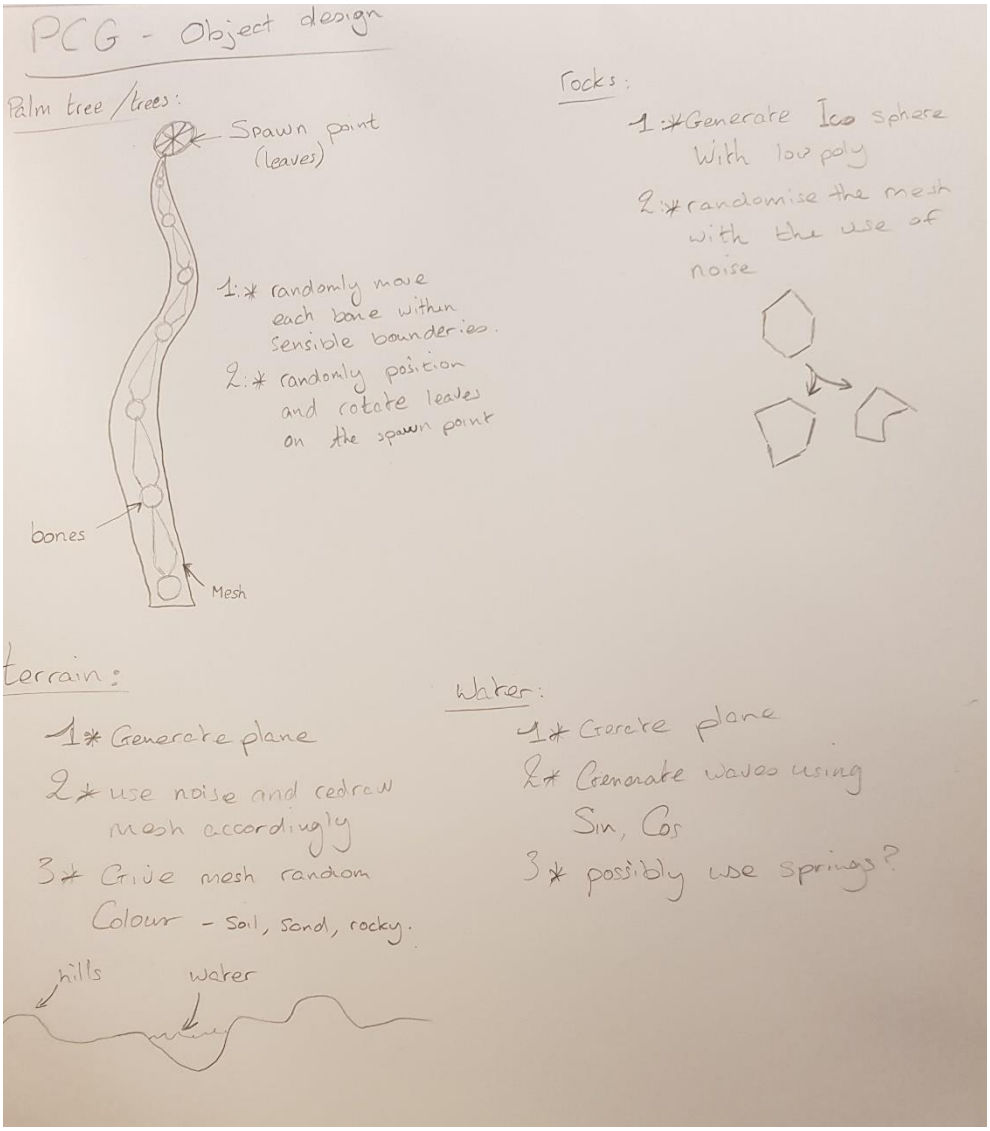

Figure 3: Early Objects Design Sheet

Christopher Ross
ROS16626280

To efficiently assist with the generation of each object a custom script was created to allow content generation outside of unity's play mode and inside the editor. An example of the generate and destroy buttons can be found within the game objects inspector tab as shown in figure 4. Furthermore, the custom editor was heavily utilised throughout the project to ensure variables were clearly understood and do not exceed set ranges set within scripts. This was done so vector2's (Figure 4: Size) have ranges as well as modified names that are similar to a normal float range (Figure 4: Side Pulse Speed).

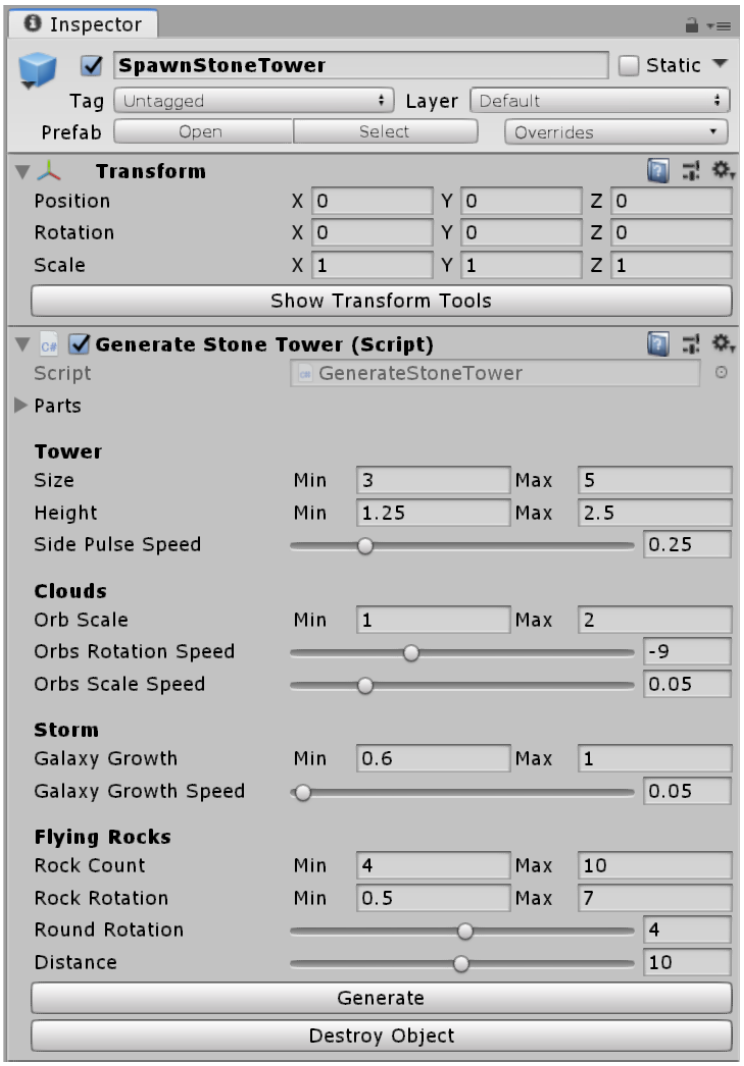

Figure 4: The Inspector

Christopher Ross
ROS16626280

1.3 Later Objects Design

As the project developed further, the complexity of game objects also increased. Bigger water and land items were needed to give the scene life and as this project was largely inspired by World of Warcraft's newest expansion, these objects were ships, homes, birds, fish and magical towers with some other elements that were not implemented. The design of these elements largely took place on paper and is presented in figure 5.

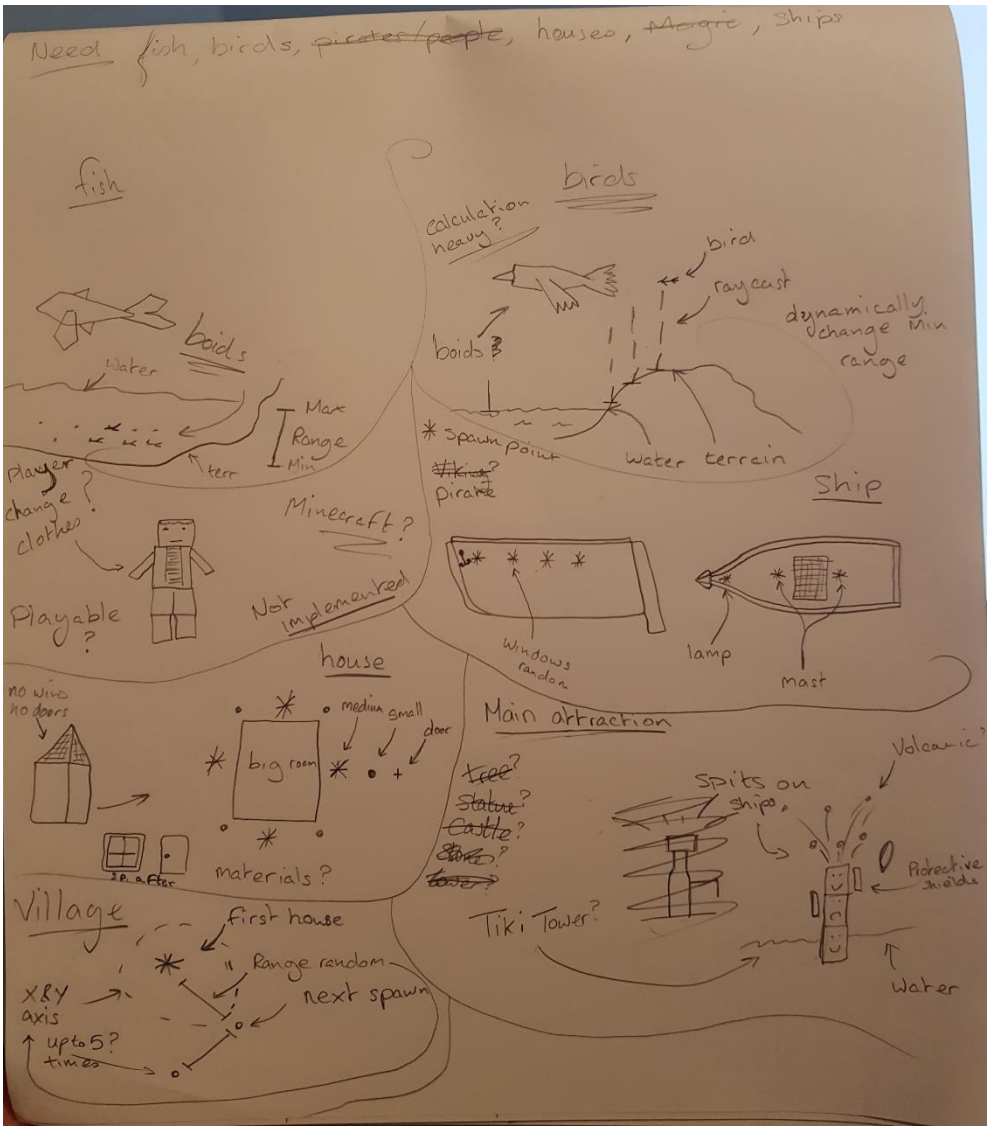

Figure 5: Later Elements

Christopher Ross
ROS16626280

2 Development

2.1 Terrain & Sea

As the desired theme was to be similar to that of World of Warcraft's Battle for Azeroth in a Caribbean setting, the terrain was covered in a sandy material and Perlin noise was generated to distort the terrain's mesh and create randomly scaled hills as shown in figure 6.

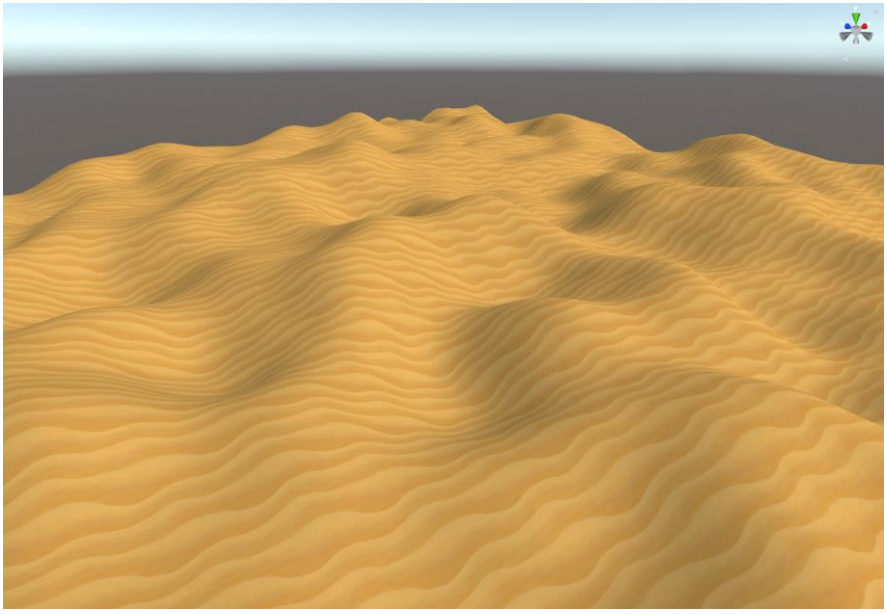

Figure 6: Terrain

With the terrain implemented it was decided that a cartoon shader similar to the one used in The Legend of Zelda: Breath of the Wild (Nintendo, 2017), should be created to give the land a painted art style thus adding to the theme and giving the scene a more vibrant aesthetic.

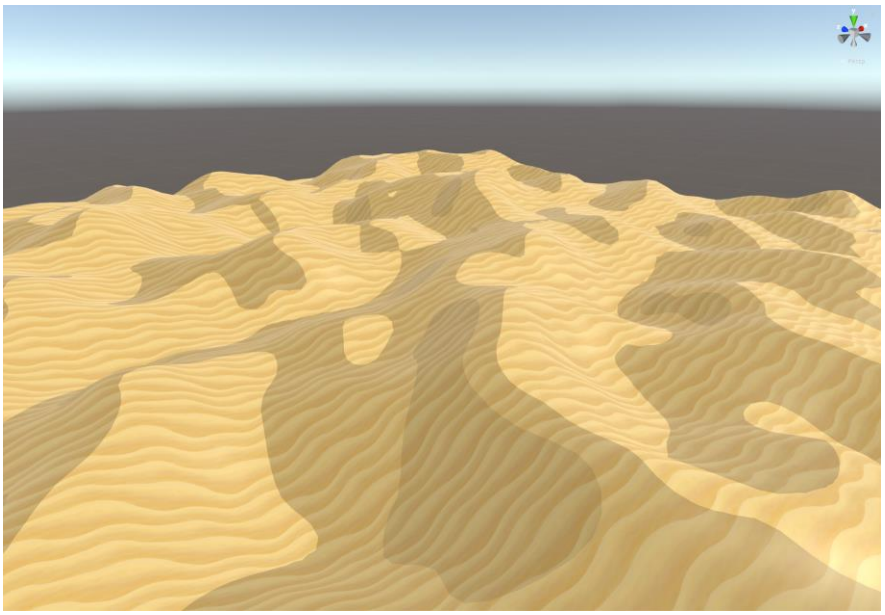

Figure 7: Cartoon Shader

Implementing the sea and waves was the next step in the development of the project. This was accomplished by first creating a plane and then modifying both its mesh and shader to create waves and foam. The mesh was modified by moving its vertices using the Mathf.Sin function to generate waves in a Sin fashion.

The shader was similar to the terrains cartoon shader, with the exception of a noise function which generates random quantities of white, which is then slowly pushed across the whole mesh to give the illusion of foam which can be seen in figure 8.

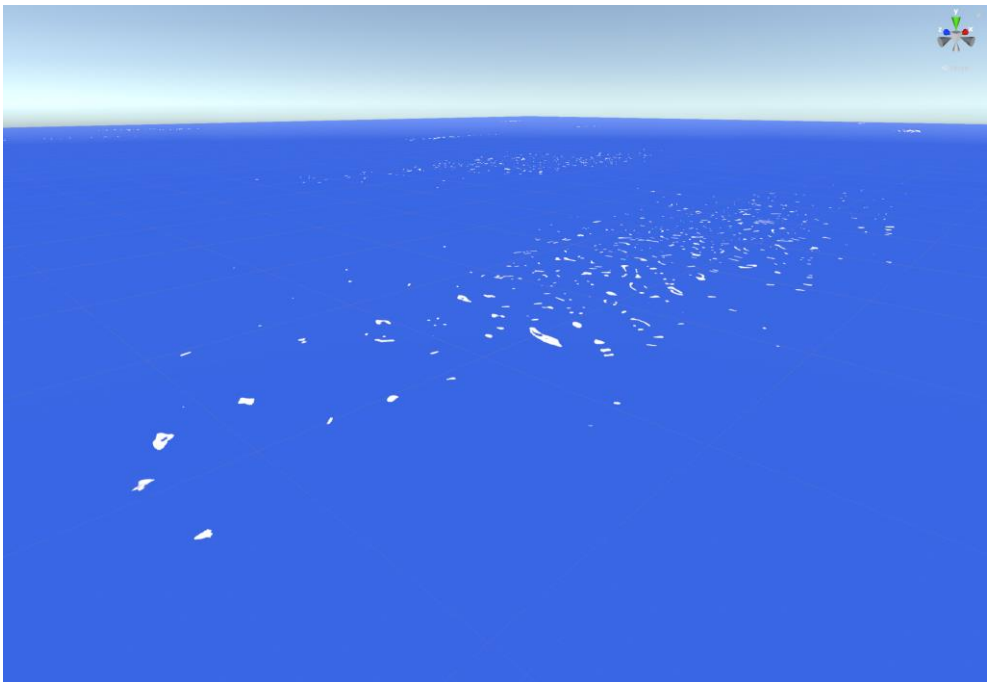

Figure 8: The Sea

2.2 Rocks

One element that can be found along all sea beds in the real world is rocks. Rocks within the game were created in three separate processes.

The first process was to generate an ico-sphere, this was accomplished through using a script created by kaiware007 which was then modified to suit the project and obtain the desired functions (Kaiware007, 2016).

An ico-sphere script was used to allow further possibilities in terms of how different rocks can be from one another. If this had been created through the use of an external program such as blender, only one version of the ico-sphere could be used at any given point meaning that it would always have a set number of vertices making all rocks generate similar to one another.

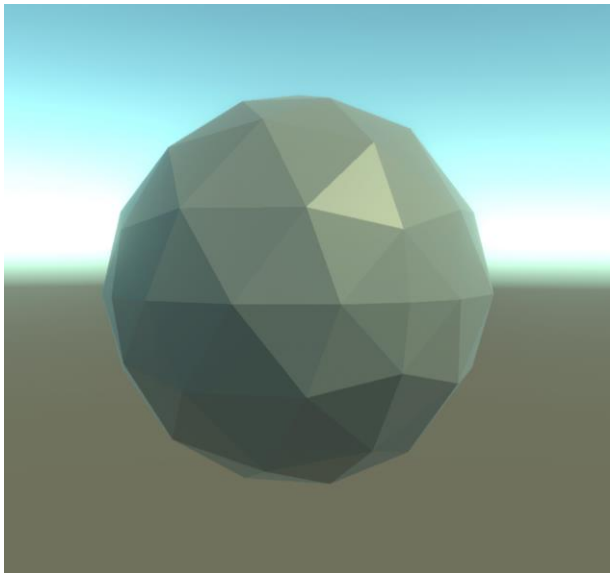

Figure 9: An Ico-Sphere

Once the ico-sphere was successfully appearing within the scene as seen in figure 9, the second process was to modify the mesh using Billow Noise (Bevins, 2003). This randomises the meshes vertices according to the noise generated though variables that are input into unity's inspector tab. Depending on what the variable is, it can contain a minimum and maximum range which is then used to select a random value between the two ranges.

Finally, the third and final process involves randomly resizing the rocks within sensible ranges, so that they generate different with each attempt.

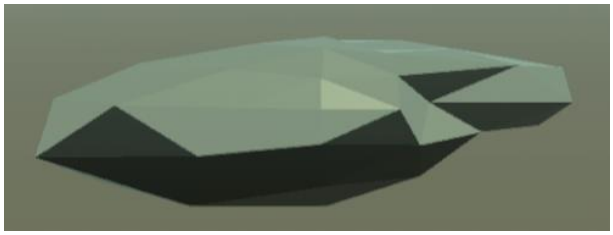

Figure 10: Rock

2.3 Trees

The first tree created within the game was a palm tree. This was done by closely following the tree drawing found in figure 3 of section 1.2.

The drawing explains that when the tree is created within blender, numerous bones are placed inside it and once the object is imported into the unity game engine, the bones can be moved and manipulated into various positions making the trunk appear more realistic. This was done within the "generate palm tree" script where each bone was rotated and positioned according to a set of predefined rules that calculates random number between sensible minimum and maximum ranges.

When the bones of the tree are moved into their final resting position, the leaves can then be added by placing them into the predefined spawn points along the tree. The trunk displayed in figure 11, has three possible locations where leaves can spawn, this can be seen in figure 12. As seen in the after picture of this figure, leaves have spawned in two of these locations. The top of the tree will always spawn leaves, whereas the other two locations are completely at random.

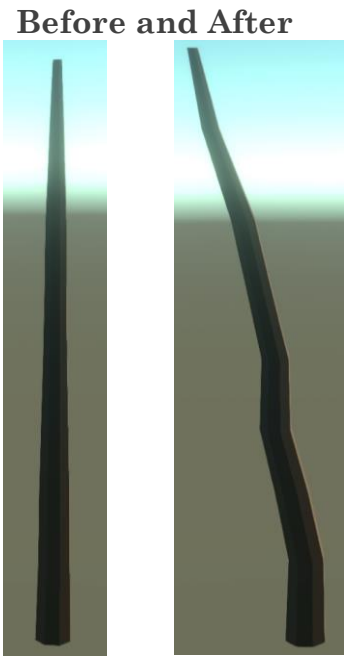

Figure 11: Tree Trunk - Before and After

Figure 12: Tree Leaves - Before and After

As seen in the after picture of figure 12, the palm tree's leaves are spawned on the tree and then, like the bones, they are then rotated to different angles and directions which as a result gives a realistic effect. Additionally, once the tree has been fully established, there is an option to repeat the process on top of the existing tree which can then act like a branch as shown in figure 13.

Figure 13: Final Palm Tree Result

The second tree created within the game was a normal bush-type tree. This tree uses a very similar process to the palm tree except for the leaves. As shown in figure 14, rather than spawning palm tree leaves, this tree uses the rock generator and spawns a rock in their place and recolours it to green giving the effect of a low-poly bush.

Figure 14: Bush Tree

As seen in the figure above the trunk is also shaped differently, this is due to numerous trunks with minor alterations being spawned on top of each other to give a jagged bark appearance that real trees have. As these trunks spawn and vary, they may also create branches similar to the palm tree as presented in figure 15.

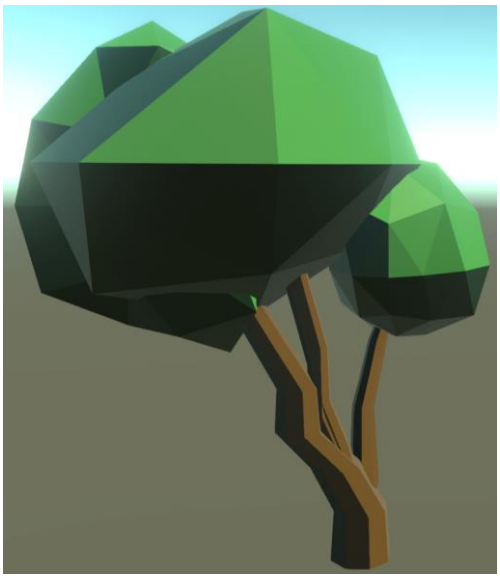

Figure 15: Bush Tree Branches

2.4 Stone Tower

Once the scene's environment had been successfully implemented, work began on a main environmental attraction. This attraction went through many iterations including a tiki statue as seen in figure 3 of section 1.2. This element was finally developed as a magical stone tower which resonates power from within its cracks and summons a storm. As a result, this tower was to be placed directly into the middle of the scene as the main attraction.

Before and After

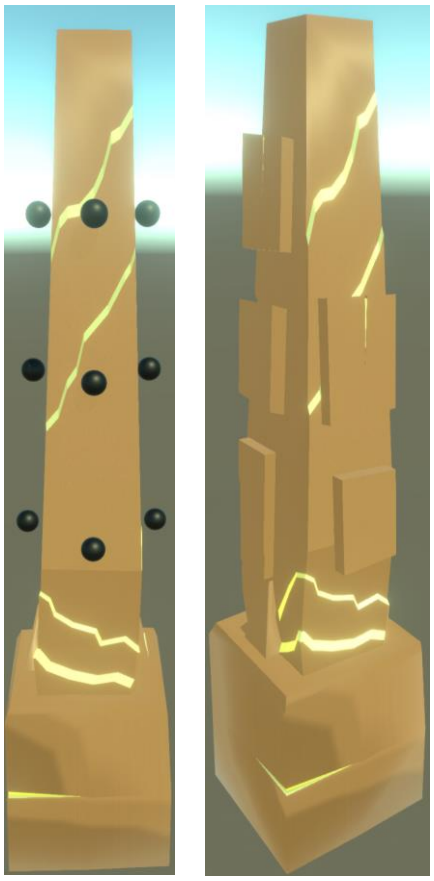

Figure 16: Tower Stone Shields

This game object was implemented using the same node type scripting as the trees which spawns other prefabricated elements into specific places. The tower currently has nine shield holders which can been seen in the before picture of figure 16. These shields were initially intended to protect the tower against the ships around who would gradually become angrier and more aggressive against as the storm grew larger. However due to time constraints this feature was not implemented.

Around the tower, there are several floating rocks that can spawn and will rotate around the tower at different speed. The number of rocks that spawn depend on the minimum and maximum ranges input into the inspector very similar to objects previously made. These floating rocks were created with the aim of providing the stone tower with some ammunition to instantly destroy some of the ships that were attacking it. However, like the feature mentioned above, this was not implemented due to time constraints.

The final feature implemented on stone tower was the storm itself. The storm consists of two elements, the first being the eye of the storm and second being the clouds.

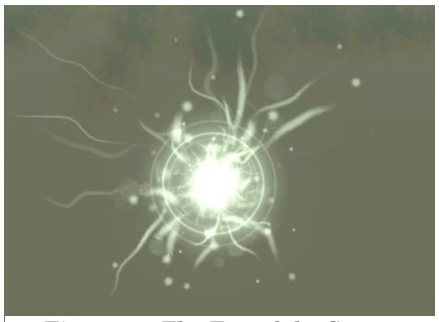

Figure 18: The Eye of the Storm

The eye of the storm uses four particle systems and was designed to be at the centre and always located at the top of the tower. Depending on how much health the tower has left, the colour of this object would slowly become darker.

The storms clouds consist of a galaxy of particle systems developed to resemble dark clouds. In figure 18, this can be clearly seen when comparing the before and after pictures of the storm where the orbs represent the positions of the particle systems.

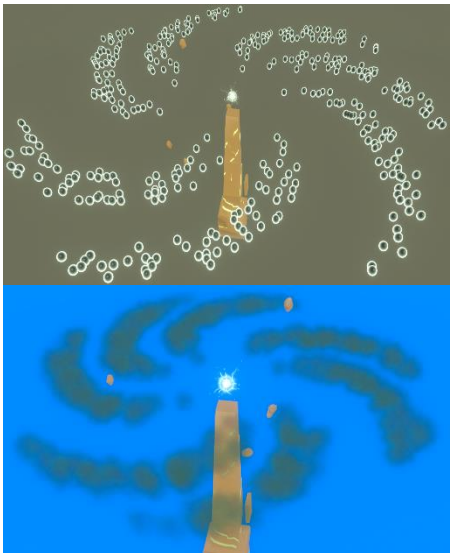

Figure 17: Storm Clouds - Before and After

Christopher Ross
ROS16626280

2.5 Villages

Houses within the game were developed almost exactly as presented in figure 5 of section 1.3. As the house spawns, an initial large room is first to instantiate. This object contains eight spawn points where smaller pieces to the house may spawn at random. If a section does not spawn in a location, it opens the area up to spawn a window instead which can similarly happen at random.

Each of the smaller pieces also have spawn points where windows can spawn at random. Depending on the size of the section, two possible windows may appear.

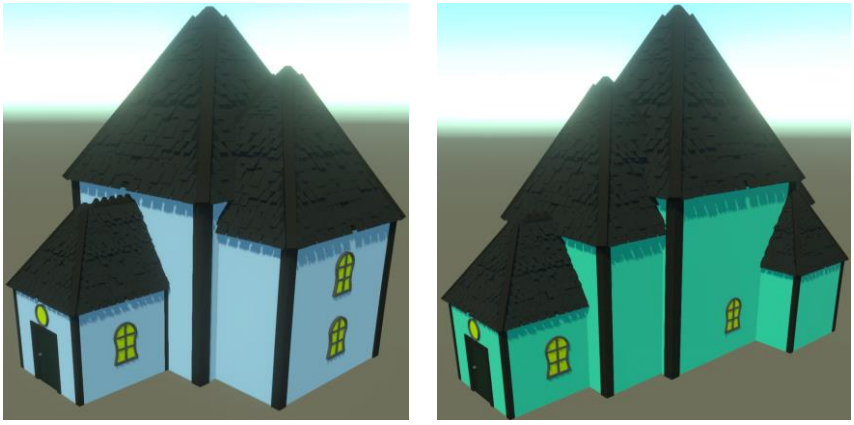

Figure 19: Generating a House

Within the game, villages are created through chains where each house spawns at a random distance from the previous one. The distance is a random number between a set minimum and maximum value. The chain can continue up to five times, however, if a house is spawned in water it is then removed from the scene.

Initially villages were supposed to spawn around a Japanese type tower which can be seen in figure 20, however this idea was discarded as there was an insufficient amount of space on most islands and as result the two game objects were then separated. The Japanese tower is built in almost the same way as the houses where its pieces are instead generated on top of one another rather than all around.

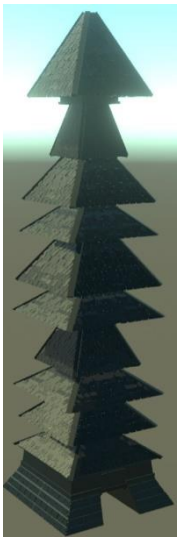

Figure 20: The Japanese Tower

2.6 Boids

To give mobility to birds, fish, and ships, a boid algorithm was used to move all neighbouring objects of the same group in certain ways. Birds and fish as seen in figure 21, are all the same with no added content to make them different from one another, whereas ships on the other hand can be generated in three ways, a boat, a small ship, and a large ship.

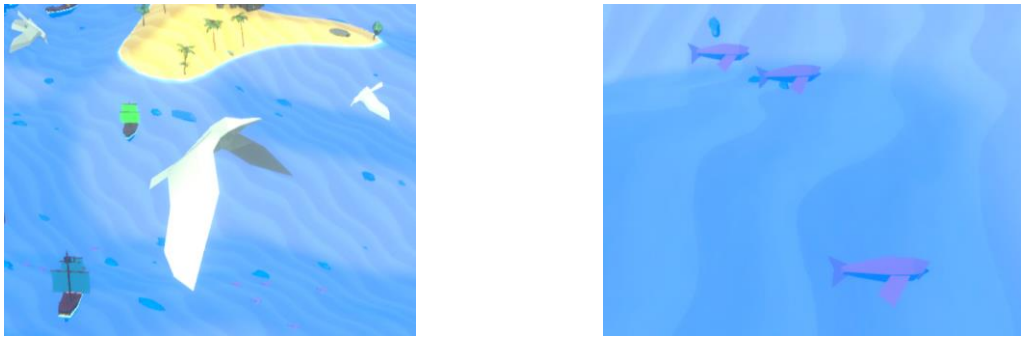

Figure 21: Birds and Fish

Like the rest of the procedurally generated objects, the ships use a spawn point system where different elements can spawn in various places at random. This allows the ship to be generated differently each time with different shades of colour.

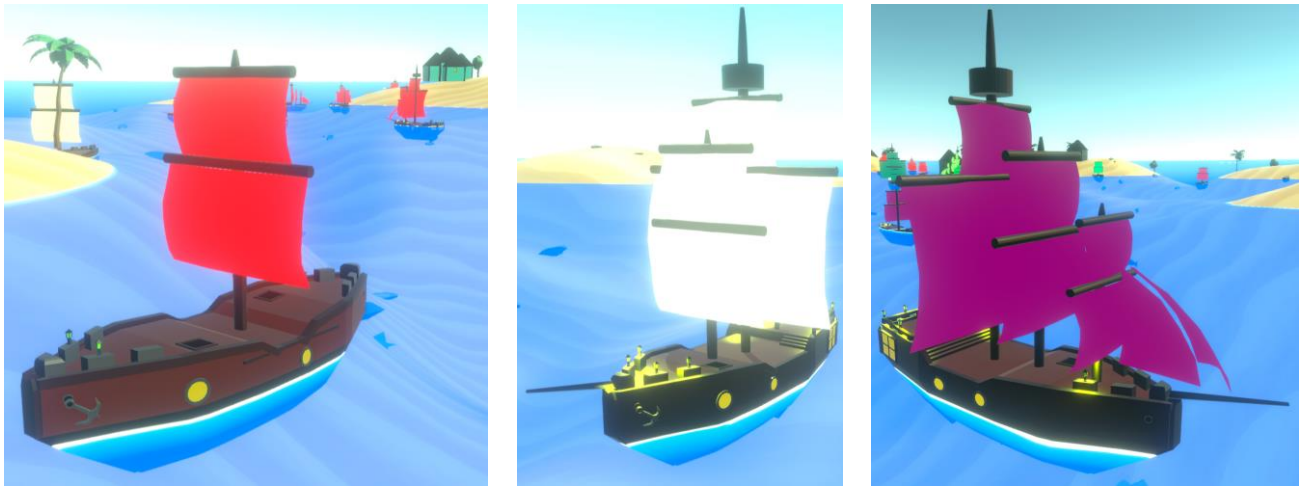

Figure 22: Boats & Ships

To control a group of boids, elements called nests were created. A nest is an invisible game object that can be placed anywhere in the scene and will attract the linked group of boids to its location. Within the scene, nests are used to ensure the ships do not wander out of the cameras view, this was done by turning the nests on and off after various amounts of time.

By using the boid generator, fleets were able to be generated with varying colours as seen in figure 23.

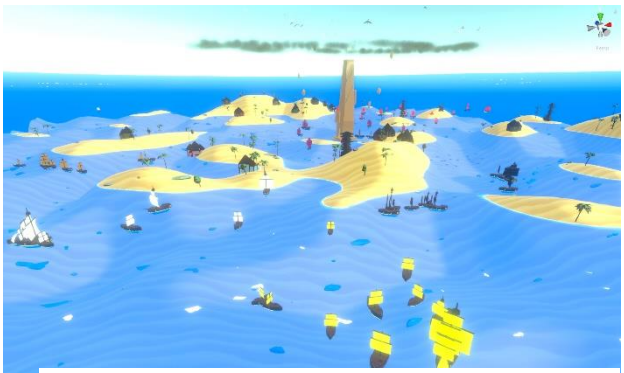

Figure 23: Fleets

3 Reflection

Even though many of my ideas were not implemented due to time constraints, I am very happy with the result. As seen in figure 24, I feel that I have accomplished what I initially set out to achieve by creating a Caribbean style, World of Warcraft - Battle for Azeroth inspired game.

Figure 24: The End Result

If I attempted to do this project again, I feel that there are several things that I would do differently. First, I would begin working on the ships and magical stone tower before anything else in the scene as these were the two main elements of the game. Additionally, I would put a lot more effort into implementing the battles between the two, as many of my plans that had gone into this portion of the game had to be discarded due to timing.

Another aspect that I would do differently is to create other terrains and environment styles to give the scene even more variations in its appearances each time it generates. This, combined with the previous mentioned change, would have provided more opportunities for gameplay concepts.

One aspect that I feel especially proud of, is the way the ships were implemented as there are many variations where they appear to be almost different each time they are generated and once paired with the boid script, they truly feel as though they belong in the scene.

Although the fish and birds were not implemented as designed in figure 5 of section 1.3, I tried very hard to make them act in similar ways to their real-life counterparts. However, as a result, I do feel as though I may have spent far too long tweaking and adding to the boid script rather than doing something else more productive within the game.

Overall, this has been fantastic experience and I feel that I will use a lot more procedurally generated content in my future assignments and work.

4 References

Bevins, J., 2003. LibNoise – Billow. [online] LibNoise. Available at: http://libnoise.sourceforge.net/docs/classnoise_1_1module_1_1Billow.html [Accessed 26 April 2019]

Braddock, E., Dinh, J., 2018. Kul Tiras/Proudmoore Ships - World of Warcraft: Battle for Azeroth. [online] Art Station. Available at: https://www.artstation.com/artwork/1dLoe [Accessed 25 April 2019]

Kaiware007, 2016. Icosphere. [online] GitHub. Available at: https://github.com/kaiware007/IcoSphereCreator/blob/master/Assets/IcoSphereCreator/IcoSphereCreator.cs [Accessed 26 April 2019]

Valecillos, D., 2019. Unity3d Procedural Generation of Buildings with a custom Unity script and custom inspector. [online] YouTube. Available at: https://www.youtube.com/watch?v=qi3qBzsddcI [Accessed 25 April 2019]

YouTube Video

https://youtu.be/rcGfEd7WlaM

Christopher Ross
ROS16626280