**System Development**

4.1. Introduction

In this chapter, a detailed analysis of the system development will be given, but first the original prototype will be discussed. There are many aspects of the system development that will be discussed such as the procedural generation, the UI, the inspection system, the quiz feature, the models and animation, the audio, and the AI.

**4.2 System Development**

Overall, this project has around 47 different custom scripts written in C#. Since there’s a large quantity of scripts, not all of them will be described in detail. The important scripts will be discussed that are relied on by the important features of the system and the low importance scripts will have brief descriptions of what role they play in the game.

**\*\*talk about the different scenes and then mention them in the sections as well\***

**4.2. World Generation**

**Procedural generation has a large role in the creation of the environment of this game. The terrain that the player walks on, the mountains in the background, the grass that’s being created and the forest that is created are all generated through code using procedural generation and a technique called Perlin noise as discussed in previous sections. Each of these aspects will be discussed in detail as they make up a large and important part of the system. Overall, There are about 11 different scripts involved in the procedural generation, below are the aspects that are most important.**

**Terrain Generation**

**First, the terrain that is generated in which the player uses to manoeuvre throughout the environment will be discussed.**

**The first step in creating this was to create an object in Unity in which the script for generating the terrain could be attached to, this object was renamed to Terrain. Next, a C# script was created and attached to the object.**

**This script is extremely important because it is connected to both the forest generation and the grass generation. The way in which the terrain is generated is by using a pseudo random pattern called Perlin noise. Variables like the scale, the heights, and the size of the terrain were created and used to generate it. By using this technique, it allows the script to generate a random terrain at different heights. Meaning once the game starts the terrain will be generated randomly through this code at different heights and positions than before.**

**This Scripts is quite long so only some snippets will be shown. Below are all the variables that were set that were used to generate the meshes.**

**Text

Description automatically generated**

**Below are some of the functions used to generate the terrain and pass in the variables to generate the different heights using Perlin Noise.**

**Text

Description automatically generated**

**Generate() Function generating Vertices, triangles, and vectors:**

**Text

Description automatically generated**

Once the Script was done the terrain size, scale and height multiplier were able to be changed in the inspector as shown below. Also, a reference for the forest and grass generator scripts were set.

A screenshot of a computer

Description automatically generated with medium confidence

Below is an image of the terrain being generated:

Chart, surface chart

Description automatically generated

**Forest generation**

**Next, the procedurally generated forest was created. this script uses the Generated terrain script to allow it to be placed on the terrain at the correct heights and be the corrected size. It does this by using the 2D heightmap array set in the terrain scripts.**

**First the forest game object was created and the forest script was attached to it. This script generates the trees by using arrays to pass in tree objects and randomly placing them throughout the terrain once the game starts. Every time the user plays the game the forest is generate and different each time. Multiple vectors were created with random range values so that the forest would look as realistic and less procedural as possible.**

**Below is a snippet of the forest generater class and the Generate function in it:**

**Text

Description automatically generated**

**In the inspector, the tree objects were able to be added so they could be generated and the forest size and spacing was set.**

**Graphical user interface, application

Description automatically generated**

**Below is the forest that was generated on the terrain:**

**Map

Description automatically generated with medium confidence**

**Grass Generation**

Next Up was the Grass Generation. This was also linked to the terrain script. This feature uses its own script just like the forest script. It is quite similar to the forest generation script with a few minor changes so it won’t be discussed in too much detail. As you can see in previous images the grass script is referenced in the terrain object. Below is an image of the grass object inspector where the object is added and the size and spacing are set:

Graphical user interface, application

Description automatically generated

Overall, there are Four scripts that are used to generate the grass in different areas of the game. But they all follow a similar pattern so there is no need to show all of them. Below is an image of the grass generated on the terrain in the forest:

A grassy area with trees in the background

Description automatically generated with medium confidence

**Mountains Generation**

**Finally, the Procedurally generated mountains were created. These were also created using Perlin noise but at a different approach from the terrain generation. This was because I wanted them to look more like mountains and to be a little bit more customizable and random. A game object was created and a script to generate the mountains was created. this was quite a long script so only small parts of it will be shown. One of the big differences to how the terrain was generated is this script uses Octaves, a height curve, lacunarity and a seed to generate a mesh more similar to mountains. The seed is basically a number that is used to generate unique set of mountains each time.**

**Bare just two of the functions in this script, one for the seed function and the other for using the octaves and Perlin noise to generate. There are many more functions in this class but these are quite important.**

**Text

Description automatically generated**

**Below is the an image of the inspector where the attributes of the Mountains could be changed such as the height curve, size, scale, seed and other attributes.**

**A screenshot of a computer

Description automatically generated with medium confidence**

**The images below show an example of the mountains being generated. A few different examples were taken form the game to show that each time the user plays the game a unique set of mountains would be generated.**

**Chart

Description automatically generated**

**As you can see the mountains are different each time.**

**Animal Generation**

**Another area of this project where procedural generation was used was when creating and spawning in the animals throughout the game.**

**Four different scripts were used to specifiy the generation behaviour for each of the animals. This is because some animals have different traits and behaviours so the scripts allowed random amounts and types of animasl to be generated throughout the map. For example some for the foxes, the scripts allowed a random number of them to be generated each tim ethe game is played each with different behaviours, these behaviours will be discussed in more detail in the AI section.**

**Objects were created to act as spawners/genrators for the animals. Each spawner had the script for the animal in wehich it was generating along with the animal itself.**

**Below is an example of one of the animal generator scripots to generate the foxes.**

**Text

Description automatically generated**

**Below is an example of the object and inspector of the fox spawner:**

**Graphical user interface

Description automatically generated**

**As said before each of these spawners were quite similar in the way they were created and written so only one of them have been discussed. Shown below is an image of the Foxes being Generated at random locations and with different behaviours:**

**\*\*Image\*\***

**4.4. Animal AI System**

**One of the main parts of this project was the animal AI system. In real life, animals have different behaviours and react differently to their environment and people. To simulate this, an animal AI system was created. This allowed the animals that were created to manoeuvre throughout the terrain and react to the environment and player in a certain way with a unique set of behaviours for the animals. The method that was used to achieve this is a type of AI pathfinding called NavMesh. This feature in Unity allows the Developer to write scripts and behaviours for AI and to build areas in which these behaviours will work.**

**In total there seven different C# scripts involved in making the AI system. The first step of the process was to create scripts that set the behaviours of the animals. In total there were four different sets of behaviours an animal could possibly have.**

**The first one was a scripts that if assigned to animal would allow that animal to attack or follow the player depending on how close it was. If the player came within a certain distance this animal would start moving towards and following the player, and if the player would run away and escape that distance the animal would stop. For example, the badger animal in the game is known to be aggressive, so the badger is more likely to be assigned these traits. So, when the world is generated a random number of badgers will be spawned in an there is a likely chance that they will have this aggressive behaviour script assigned.**

**\*\*insert images of scripts and animals\*\***

**The next type of behaviour scripts was set to be the most commonly assigned behaviour script. The behaviours included the animal roaming or wandering around the environment and if the player approach or came to close to that animal, the animal would run away from that player. A series of positions in which the animal would wander to were set and then a function was created to make the animal run away from the player if they grew near. For example, if a squirrel was generated with this behaviour script it would wander around the terrain and if the player approached it would run away from the player.**

**\*image scripts\***

**The next two behaviour scripts were ones that worked together. One allowe the animal to run around the terrain and the other allowed an other animal to chase that animal that was running. These allowed some aniumals when generated to run around the terrain and ither animals would chase them to simulate an animal chasing another one that might happen in real life. For example if a two foxes were generated with one of these scripts, ine of the foxes would chase other around the map.**

**\*insert images scripts\***

**The next step was very important in creating the AI as if it wasn’t done these scripts wouldn’t work. For these scripts to work the area in which the animals were being generated needed to be built so that the animals could interact with the environment. To do this scripts were created to set the areas as NavMesh Surfaces so that the AI could use the scripts they were assigbned which us Unitys AI pathfinding technoilogy called NavMesh.**

**To do this each animal that is generated is assigned a script called NavMeshSurface. This script represents the walkable area for the anials to walk on and where the navmesh should be built. This is quite a long script so only snippets will be discussed.**

**\*scriopt image\*\***

**\*don’t forget inspector images aswell and navmesh images surface.\*\***

**To make this AI system a bit more complex two more scripots were used in the system which allowed some of the animals to jumpo to another navmesh surface which create da realistic jump using code. The second script allowed the surfaces of the nav mesh to be built once the game was ran in runtime so that when the animals were generated they could find an area where their AI scripts would work just incase they were generated somewhere that they weren’t supposed to be.**

**\*images\*\***

**Once these scripts were written they were attached to various animals. As said in the previous section, once the game is started the animals will be generated and each animal would be assigned a set of behaviours at random according to the AI scripts written. For example…**

**\*still have to describe the code.\*\***

**4.5. Inspection System**

**The next feature that played an important part of this project is the inspection system which uses a technology in C# called Raycasting. This system allows the player to look at wildlife and view information about the wildlife they are inspecting and then access different features like the quiz feature from this system. To get an idea of what this inspection system looks like, images are shown below:**

**\*inoection image\***

**In total, there were four C# scripts created in the making of this feature. The first one was a class called InpectRaycast this script uses the Rayasting technique which was mentioned before. This technique allows the player to project an invisible laser (raycast) to be able to detect any object that’s in the player line of sight. So when the player looks at a piece of wildlife they will be able to click to view information about the wildlife they are inpecting.**

**Below is a snippet of the raycasting technique being programmed in C#:**

**\*inspectraycast snippet\*\***

**Once this class was finished a script was created called inspectController so the UI could be managed for the inspection system to be linked with the inspectraycast class. This allowed the UI for the information, text, and images to be displayed when using the raycast script. This was done by creating different functions to display a prompt when looking at a piece of wildlife and then once the user clicks the prompt, more information is displayed on screen for the user to view. This script also allows for which UI needs to be displayed by adding it to the inspector by having references for them as shown below:**

**\*\*inspector inspect controller\*\***

**Below is a quick snippet of some of the code for this script for showing the information and prompts:**

**\*\*inspectController\*\***

**All UI was created in unity by creating buttons, images, text and backgrounds.**

**Finally a script called object controller which is the script that allows the previous two scripts to work and be attached to wildlife objects with references to the different variables and UI elements such as what information is displayed and the name shown when the user views the prompt. Each wildlife object has this scipt attached to it. For example every frog that is generated has the object controller scipt attached which references the inspectController scriptd which in turn references the inspectraycast script which then allows the inspection system to work.**

**\*\*objectcontroller class\*\***

**Below is an image of a frog object that was generated with the inspection system scripts attached with information about it.**

**\*\*example of frog inspector\*\***

**The last part involved in the inspection system was the managerNPCUI class which allowed a symbol to be displayed over certain animals bodies to notify the player that it is near by and so its easier to find. This script was attached on to various animals that were generated. An example of this is shown below for the frog:**

**\*exclamation of frog image\*\***

**When the user approaches the animal the exclamation symbol gets smaller and when you move further away it gets bigger.**

**\*\*managerNPCUI class image\*\***

**To recap how this system works, the player wanders throughout the terrain, it finds an animal with a symbol hovering over it, the player then approaches the animal, then they look at it, the raycasting allows the prompt for the animal name and information and quiz options to be displayed, the user can left click to view more information or right click to access the quiz menu feature.**

**\*\*put down step by step image process of inspection system below:\*\*\***

**4.6. Quiz System**

**The next feature that will be discussed is the quiz feature. This feature is a main part of the game and required lots of different scripts and classes in C#. only the main parts of the development of this feature will be described in detail as there are many parts to it. In total there were about 18 different scripts involved ion this feature but only a few will be discussed in detail.**

**In total there were nine different quizzes in the game, one for each type of animal and tree. The quiz systems two main classes which are the mnost important and the longest of the scripts used to create the quiz are called GameManager and UIManager. These scripts use Lists, arrays, animations, Coroutines, structs and numerous functions to create the quizzes.**

**The first step in building the quiz was to create the UI. A new scene was created and background images, buttons and text were added to design the layout of the quiz.**

**\*\* image of layout\*\***

**An object was created for the GameManager and the UIManager scripts to be attached to.**

**Image of manager object with scripts attached:**

**\*\*inspector of manager object\*\***

**In short the GameManager script basically managed the quiz which loaded the questions, updated the answers, getting random questions, checking if the answers are coirrect or incorrect, starting the timer of the quiz, comparing answers, restarting the quiz, setting the highscroe, update the score, and just overall managing the quiz. This scripts was extremely long amounting to almost 400 lines of code so only a snippet with all the variables that were set are shown below. this shows the arrays for the questions, the lists for the answers, the game events, animators and coroutines.**

**\*\*variables for gamemanager script.\*\***

**The UImanager scripts which managed the UI of the quiz to decide what ui was top be displayed when using the quiz uses a struct which** is **a composite data type (or record) declaration that defines a physically grouped list of variables under one name in a block of memory**, allowing the different variables to be accessed via a single pointer.

**It also uses a series of fuinctgions to do things like update the question UI, diplsay resolution screens with the score counter, switch the timer, calculate the score, create and erase answers, and update the score UI.**

**This scripts was also very long so below is a snippet of the struct followed by a snippet of some of the method used like updating the questions, displaying the resolutions screens and creating answers and calulating the score.**

**\*\*screenshots of UI manager class\*\***

**Etc\*\***

**Once these scripts were written they were attached to the managers game object and all the reference to the UI and other scripts used were assigned.**

**Seen as though there are lots of scripts involved in this quiz system, the other class and scripts will be described briefly and not in too much detail.**

**Question\_editor Script**

**The question editor scripts basically allows you too create and edit questions within unity in the inpecoitr instead of hardcoding them in. it allows you to set the question, say whether or not you want to use a timer for that question, set the answer type to single or multiple answers, the amount of score that question gives you, the amount of answers there are and what the answer is.**

**\*\*screenshot of a questions inspector\***

**Graphical user interface, application

Description automatically generated**

**Snippert of Funbction from question\_editor script:**

**\*\*image\*\***

**AnswerData**

**This scripts is basically just used to manage the answer data so it can update answer data, reset, switch states and update the GUI of the answer selection. This scriopts is referenced in the UIManagers Script.**

**\*image if answer data script and inspector\*\***

**Answer\_Drawer**

**This scripts just managed the layout of the UI for the answers so that they look correctly laid out an positioned even if they are all different for each quiz.**

**\*\*image of class\*\***

**\*\*image of answer UI\*\***

**Question**

**The question script basically allowed the user top create new question objects within the Unity editor so that they didn’t have to be hardcode and you could add different questions for each quiz depending on what animal or tree it was being quizzesd on. It also allowed the quiz to get and return the correct answers.**

**\*\*image of creating new question with right click\*\***

**\*\*image of scripts\*\***

**GameEvents**

**This just managed all the different callback for the quiz and scripts which basically msnaged all the game events.**

**\*\*screenshit of script\*\***

**AudioManager object and script**

**Finally the audio manager object was created which had a script attached to it called AudioManager. This allowed all the different audio sources to be played at the right times such as the correct noise, the incorrect noise , the countdown and the game Music. It also allowed the developer to change it iin the inspector as shown below:**

**\*\*image of audio manager inspector\*\***

**This script had multiple classes and functions to fetch, play, stop and set the sounds and audio of the quiz.**

**Snippet of Play, stop and Get Functions:**

**\*\*image of functions as described above\*\***

**Once all this was done, multipole quiz scenes for each animal and tree were created along with the questions to match each one. The UI was also change to match the wildlife the user is being tested on.**

**Below is an example of the Fox quiz:**

**\*\*image step by step of fox quiz\*\***

**4.7. Models**

**Procedural generation**

* ***Terrain Generation***
* ***Forest Generation***
* ***Grass Generation***
* ***mountains***

**AI**

* **AI Pathfinding called NavMesh**
* **Four behaviour scripts, AIB, AID, AIF, AIS**
* **NavMESH Surface Script**
* **AnimalLinkMover**
* **NavigationBaker**

**Menu Creation**

* ***Main Menu***
* ***Pause Menu***

**Inspection System**

* ***Inspection controller***
* ***Inspection ray cast***
* ***Manager NPC UI***
* ***Object controller***

**Quiz**

* ***Question Editor***
* ***Answer drawer***
* ***Answer data***
* ***Audio manager***
* ***Game manager***
* ***UI manager***
* ***Game events***
* ***Question***
* ***Game Utility***

**Models**

* ***Trees***
* ***Rocks***

**Animations**

**Audio**

**Images**

**UI**