Wild-Ireland – Educational Wildlife application that teaches through a first-person game.

TU857

BSc in Computer Science (Infrastructure)

Jake Bolger

C18395341

Tanya Thompson

School of Computer Science

Technological University Dublin

07/04/2022

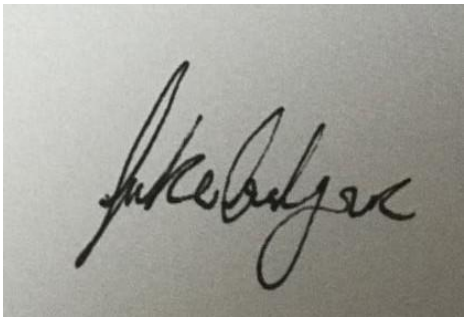
Abstract

Nowadays, it is difficult to teach students about wildlife by only using textbooks and presentations. This project attempts to create a game environment that uses procedural generation and AI to allow students to learn about Irish Wildlife and give them a new learning experience. This project uses realistic procedural environments, animal AI, and Quizzes to learn about Irish wildlife in a new and creative way. It focuses on wildlife behaviour and giving the user the right information to help them learn.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:



Jake Bolger

03/01/2022

Acknowledgements

Throughout my time at TUD, there was a lot of guidance and support received through the duration of this project. I would like to thank my parents who supported me through the year, their encouragement and support kept me on track.
I would also like to thank my supervisor Tanya for being a great mentor. She was extremely helpful and always willing to have meetings even over the holidays. Her guidance helped me complete this project.

Table of Contents

Table of Figures

1. Introduction

1.1. Project background

Teaching students in schools and colleges about certain topics can take a boring and uninstructive approach. Many people recall finding it challenging to concentrate and learn information whilst reading from textbooks and lecture slides. Although students can learn this way, it can fail to create an engaging and interactive way for the students to learn. For many students, it makes it harder to concentrate for long periods of time.

This project uses 3D modelling, AI, and educational techniques to develop and create an interactive experience that keeps the user engaged in what they are learning about. Using AI to create realistic wildlife behaviours and using quizzes to test the user on the information provided, this system gives the user another option to choose from when learning about wildlife in Ireland.

The use of procedural generation allows the user to experience an immersive environment to explore and discover various wildlife.

1.2. Project Description

The main goal of this project is to be used as an educational tool to teach students about wildlife in Ireland and improve their knowledge of the various species of plants and animals. This will be accomplished by creating a 3D game in Unity that allows people to have a more interactive and enjoyable learning experience.

The user will be able to create and start a new environment whenever they please. These environments will be randomly generated by the game. This Game will allow the user to move freely throughout the terrain which consists of real Irish wildlife such as plants, trees, and common species of animals. The player will be able to move with a first-person view of the environment and they will be able to explore the terrain as they please. Upon finding different wildlife they will be able to examine the wildlife. By inspecting the wildlife, the user will get access to information about the desired object they are looking at such as videos, images, and text.

Models and assets will be created using Blender and Unity. The animals in this game will use AI Pathfinding to manoeuvre and interact with the player and their surrounding environment, they will also be placed procedurally throughout the environment. These animals will be animated using keyframe animation

Once the player has viewed information about a certain life form which will consist of videos, pictures, and text. They will be able to use this knowledge to test themselves with quizzes. Quizzes will be available to the user who wish to test their knowledge on what they have learnt.

Sometimes learning through textbooks can be boring and hard to attract the attention of students. The idea is that the students will be able to access this game by downloading from a link or web app.

1.3. Project Aims and Objectives

The main aim of this project is to provide a game that helps students learn in an engaging way that does not just focus on learning from textbooks and PowerPoint slides.
To reach the objectives of this system, goals were set and are as follows:

- Conduct thorough research and review literature relevant to this project.
- Compared similar systems to this project.
- Choose suitable design methodologies and create prototypes.
- Create a procedural environment using procedural coding techniques.
- Design and create wildlife with realistic behaviours.
- Use educational techniques to create quizzes for a learning experience.
- Create and design interfaces that are simple to use.
- Test and evaluate the system by using suitable testing methods.
- Once the process is complete, review and reflect on the system using user feedback.

1.4. Project Scope

This project will have AI that that can interact with the user and environment. It will have a procedurally generated terrain and terrain objects. The animals will be procedurally created and animated. The user will be able to inspect objects and add them to an inventory system.

This AI in this project does not aim to use machine learning for the animals to interact with the environment. It will instead allow the animals to use AI pathfinding to navigate

through specific areas of the terrain and use the rules set in the scripts attached to the animals to react to the environment and the player in specific ways for each animal.

The game will be designed to only have one type of terrain which will be a forest, it will not have other terrain found in Ireland such as beaches, mountains, urban areas.

The animations for the animals will be completed using keyframe animation so it will not be procedural.

The inspect system will not work on every object in the game, only on specific wildlife game objects. The inventory system will not hold unlimited objects, it will hold the number of objects that are available for inspection.

1.5. Thesis Roadmap

1.5.1. Literature Review

This chapter discusses all the relevant background information and research that was completed for this project. Firstly, it looks at the similar systems to this project that have already been created. It then shows the research done into the different technologies that could be used such as games engines, programming languages, database tools, script editors and 3D modelling software. It then goes on to discuss other research such as AI, animation, audio, and assets. Lastly, it goes on to discuss final year projects and academic papers. All this research is described, compared, and decided upon.

1.5.2. Design

For this section, the design approach of the system is discussed which will be used to help develop the system. It discusses version control, software methodology and an overview of the system. It then goes on to talk about requirements, UI design techniques, prototypes, use cases, database, and the overall system design.

1.5.3. Development

This section discusses the development process that has been undertaken to create this system using the architecture presented in the design section. It then goes on to talk about issues in development.

1.5.4. Testing and Evaluation

This section outlines the testing and evaluating done on the project. It talks about the different testing methods used throughout the game such as white and black box testing and discusses in detail the feedback acquired from third- party users.

1.5.5. Conclusions and Future Work

This section will discuss and reflect on the system that was created. It will discuss the conclusions in detail, the issues and risks that came along with it and the plans for future work in this system. A GANTT chart will be used to complete this part of the project.

2. Literature Review

2.1. Introduction

This section looks at, researches, and considers all the different technologies associated with this project. It looks at alternative solutions, programming languages, script editors, render pipelines, modelling software, database tools and various applications.

It also looks at other research such as AI, whilst also focusing on other techniques to be used such as animation, audio, save systems, quizzes, UI, inventory systems and Unity Timeline for cutscenes.

Finally, it discusses existing final year projects based on this topic.

2.2. Alternative Existing Solutions to Your Problem

On a variety of platforms such as Steam, PS Plus and Xbox games – Microsoft Store, there are existing wildlife games that provide educational experiences to their users.

The following applications are games that provide these experiences.

2.2.1 Beyond Blue

Beyond Blue is an educational narrative underwater diving game that allows the user to explore the ocean and published by an American studio called E- Line Media (1). This game was researched because many of its features share similar aspects to this project, such as interactive animal AI and educational information about wildlife.

This game's features are as follows:

- Interactive ocean allowing tracking of sea creatures.
- Narrative Experience for educational purposes.
- Unique soundtrack.
- Educational mini documentaries.
- Photo mode to allow for pictures.
- Interactive AI animals.
- Animal Collection.

Beyond Blue as a game has many **advantages**, they are as follows:

- It’s a good representation of how a game can make learning very interesting for a user.
- It creates an interactive experience that supplies a lot of valuable information about wildlife.
- Allows the user to gather this information by interacting with animal AI throughout the game.
- This game is available on a variety of platforms for example, Steam, Apple Arcade, PlayStation 4 and Xbox One and comes at a price of
- €16,79.

Disadvantages:

- Although this game has many rich features it does not explore the aspect of quizzes.
- Its main purpose is not for the user to learn the information being
- presented.

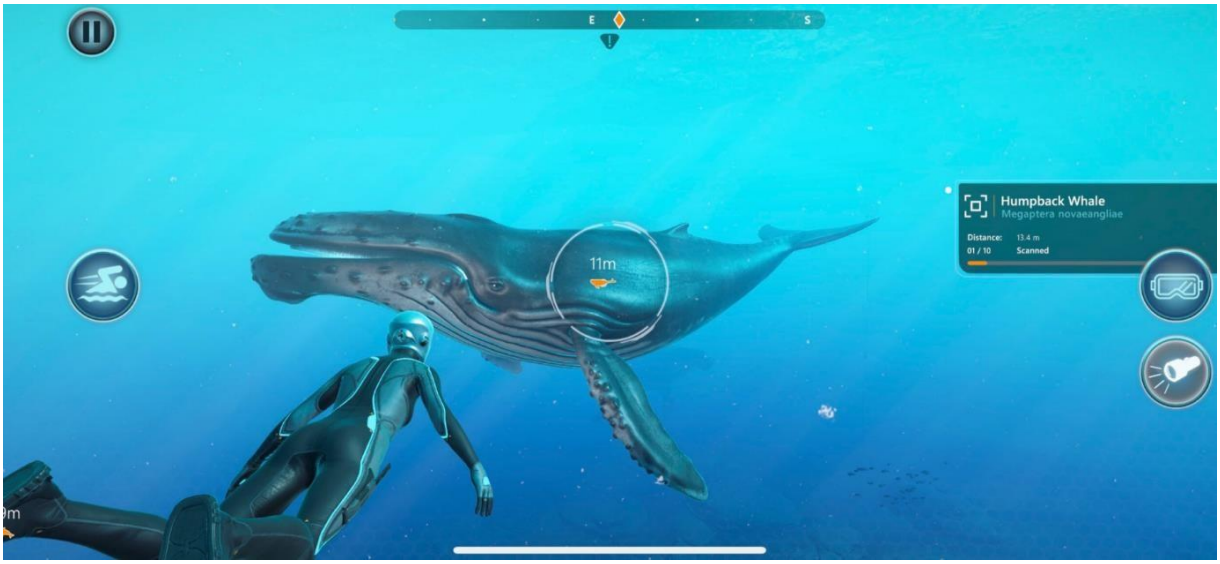


Figure 1 - Beyond Blue

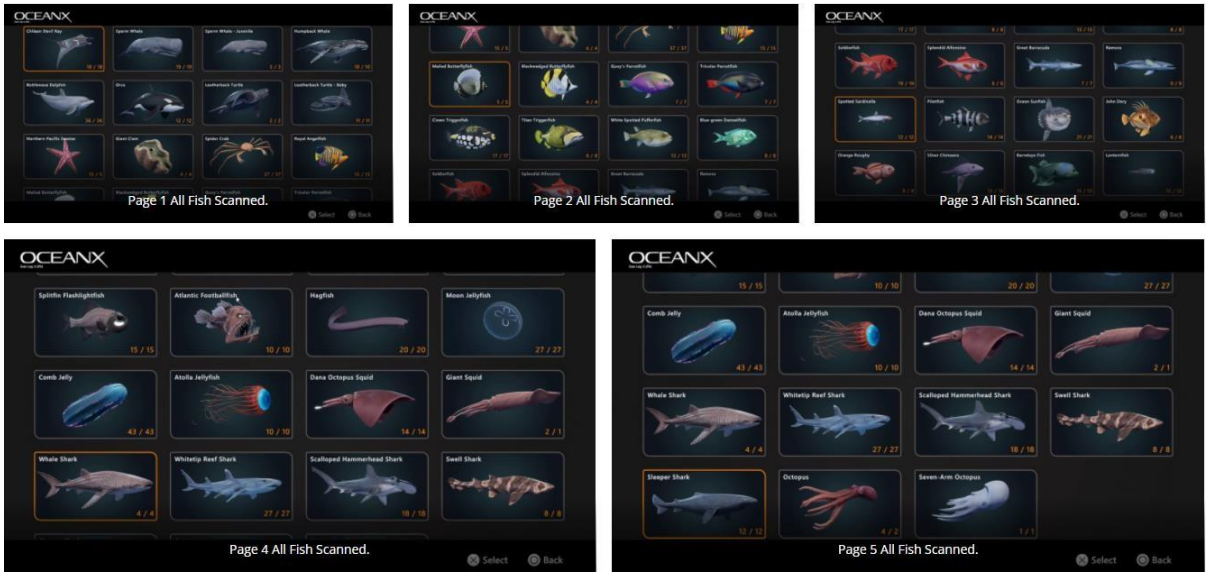


Figure 2 - Beyond Blue 2

2.2.2. The Hunter: Call of the Wild

This game is an immersive hunting game published by Expansive Worlds and Avalanche Studios (2). The main purpose of this game is hunting. Although the main purpose of this game is not educational it does teach the user how to hunt and it shares a lot of the main features and concepts of my project such as World exploration, animal AI and learning.

This game's features are as follows:

- World exploration / Open world.
- Animal AI.
- Complex animal behaviour.
- Weather events.
- Day and night cycles.
- Use of guns
- Learn how to hunt.

Call of the Wild is a high-quality game, and its **advantages** are listed below:

- Good example of how an open world game can work.
- Good example of how one could implement animals using AI into a game.
- The hunting feature demonstrates how the user can be taught information on different animals.

Disadvantages:

- However, this game isn't very educational.
- It doesn't have any methods of letting the user practice the knowledge they've learnt.

Figure 3 - The Hunter: Call of the Wild

Figure 4 - The Hunter: Call of the Wild

2.2.3. Zoo Tycoon

Zoo Tycoon is an educational game that simulates a zoo. The main purpose of this game is to build a zoo and learn about the various animals inside. This game is efficient at teaching children about different species of animals(3). This game is good in relation to the educational aspect of this project as it allows the user to collect species of animals and learn about them via a journal.

Published by Microsoft Corporation, this game's features are as follows:

- 200 animals.
- Create custom zoo.
- Collect species.
- Educational information ("zoopedia")
- Animal AI.

Advantages:

- Zoo Tycoon is a good example of an educational game for children.
- It creates an interactive gameplay environment which allows users to build and learn about animals whilst having an enjoyable experience.
- A feature that was very relevant to my project was the "zoopedia". This allowed the user to press a button and view their "zoopedia" which gave them information about the animals.

Disadvantages:

- This game had no testing feature such as quizzes so the user could test their knowledge.
- This seems to be a reoccurring theme with educational games.

Figure 5 - Zoo Tycoon

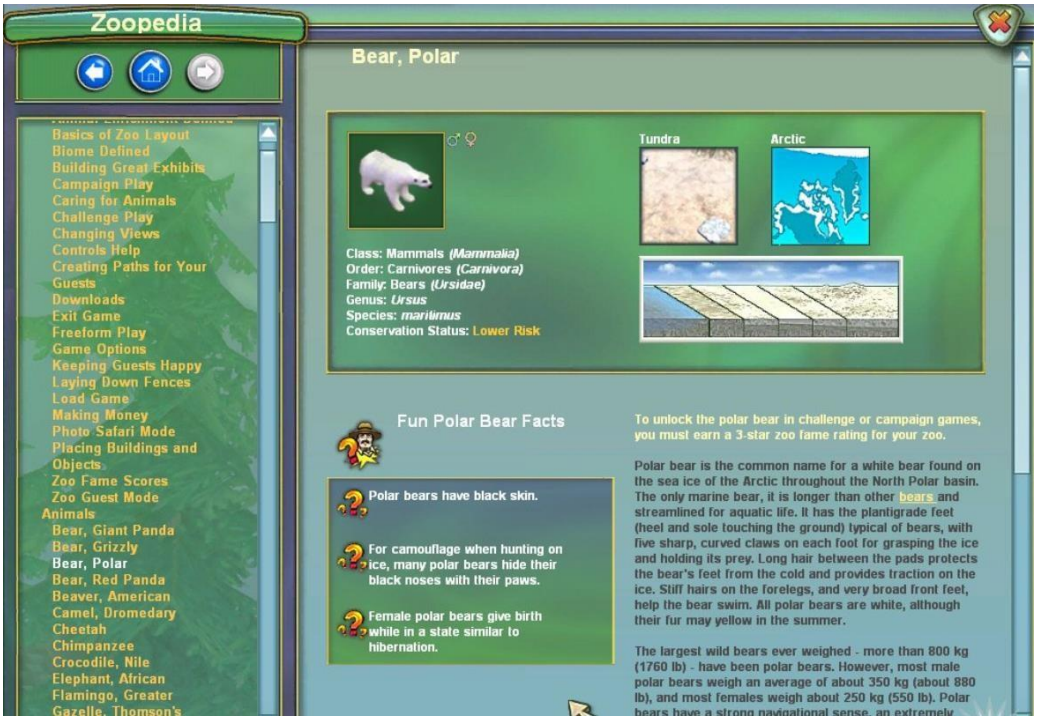


Figure 6 - Zoo Tycoon

2.3. Technologies Researched

In this section, multiple technologies that may be used in this project will be discussed. The most viable options that were considered will be discussed and the main advantages and disadvantages will be researched. Finally, an explanation of what technologies were chosen will be provided along with the technologies that weren't chosen.

2.3.1 Games Engines

A game engine is a software tool that is used to design, develop, and create video games. There are lots of different games engine available but only one will be chosen to use on this project.

Below are the games engines that were researched at and considered to use:

- Unreal Engine
- CryEngine
- Armory
- Unity

Unreal Engine:

Unreal engine is one of the most advanced and open 3D games engine tools. This engine was developed by epic games and first released in 1988. Unreal engine 4 can be downloaded free and has been used to create a variety of big games such as “Gears of War”, “borderlands” and the “Batman” games.(4)

Advantages:

- This engine provides stunning and workable graphics.
- Its user interface is constantly being updated with the latest tools
- It can enable a user to create games without writing scripts.
- It uses the programming language C++ which is generally the best option for most game developers.

Disadvantages:

- When trying to make simple games, Unreal Engine is not the best choice.
- Unreal engine is only suitable for larger teams of people to work on and projects that are long-term.
- This engine is hard to learn so for developers just starting out it us not suitable.
- The engine also requires the user to pay a 5% tax once the game is profitable.

In relation to this project, the learning curve for this engine might not be suitable for a project of its size and it is not suitable for small projects.

Figure 7 - Unreal Engine 4

CryEngine:

CryEngine is a game engine designed by a company called Crytek. This engine was used to create all the “Far Cry” games. This engine is free to use and was initially released in 2002.(5)

Advantages:

- Support for VR
- Very little programming required
- Easy to learn for someone to learn.
- Has good support and its graphics capabilities are quite good.

Disadvantages:

- CryEngine’s userbase is quite small.
- There are fewer ways to find help when using it.
- It is not suitable for small development teams or projects.

In relation to this project, although it’s easy to use, it is not suited to a small project like this.

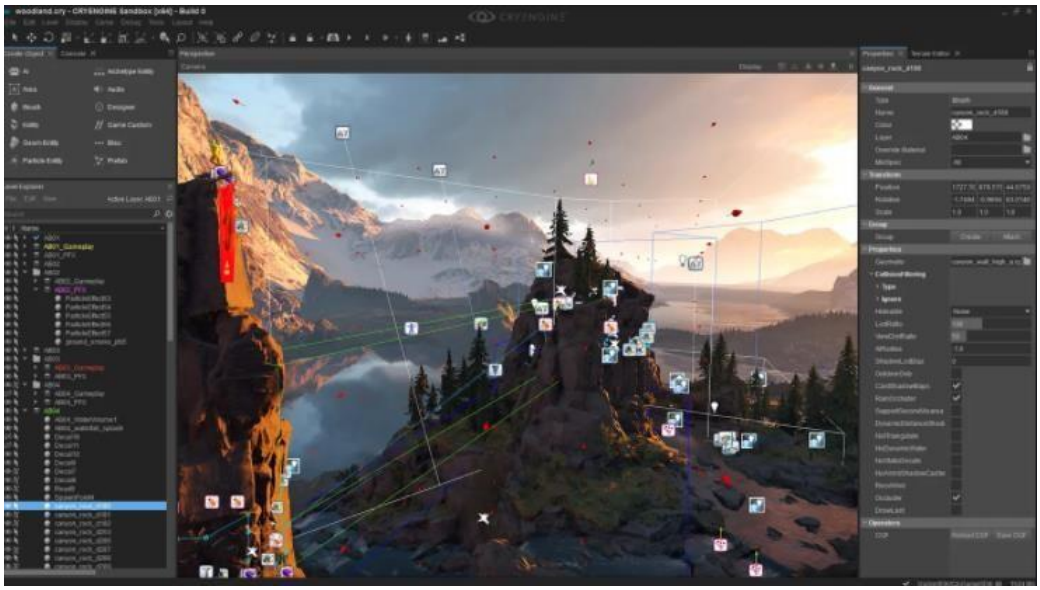


Figure 8 - CryEngine

Armory:

Armory is a 3D Engine that is open source and provides a full Blender add-on. It's written in the HAXE programming language. Games created in this engine are "Offroad Mania", "Ghost Rush" and "Xpedition". (6)

Advantages of Armory are:

- Armory is an Open-Source engine.
- This engine provides good Flexibility.
- Logic Nodes make a visual way of creating interactive behaviour.
- The coding for this engine it quite easy compared to other engines.
- This engine provides a Blender add-on.

Disadvantages are:

- This engine is not even fully developed.
- The user base for this engine is very small providing low support.

In relation to this project the blender add on would be a good advantage, but the engine is not finished.

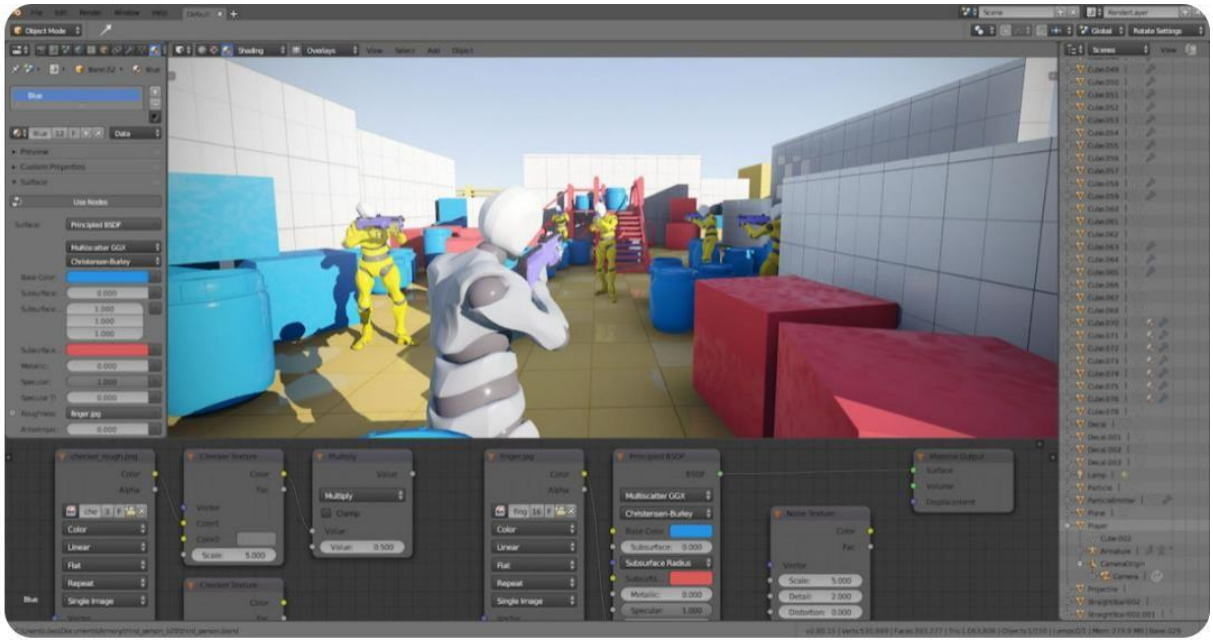


Figure 9 - Armory

Unity:
Unity is a game engine developed by Unity Technologies. It was released in 2005. It is cross-platform and is free to download and use. This engine is very popular and has lots of big games that have been created using it. “Cuphead”, “Escape from Tarkov” and “The Forest” were all created in Unity. (7)

Unity Advantages are:

- Its Free to use.
- There is a huge Community support for Unity.
- Unity is well known for being suitable for small teams.
- Interface is easy to learn.
- Uses C#. This language has tons of support and is easy to learn.
- Asset store. This provides lots of free assets to be used in games, such as models, objects, code.

Disadvantages are:

- Graphics aren’t as good as other engines
- Not suitable for AAA games.

In relation to this project its perfect for small teams like this, it’s got a lot of support so if there are issues, they can be resolved easier. Some risks that might be associated with Unity is that using too much memory or space in the project that slows the project or game down. Assets like materials and textures take up lots of memory. Optimizing the memory space in Unity is important to keep the project efficient.

Figure 10 - Unity

Decision

The games engine “Unity” was chosen. The main reason for this is that my project is a one-man game which Unity is very suitable for. Unity has great community support with lots of information online about it which will be helpful when creating this game. It also uses C# which I already have experience in.

The reason for not choosing any of the other engines listed is because they all have bigger learning curves and are not suitable for small teams such as one person. There is very little support for some of them which I will need in the future.

2.3.2. Programming Languages

In Unity, to use many of this engine’s features, scripts need to be written using a programming language and used to give it instructions to create games. there are a few different languages that I have researched and considered. (8)

The languages are as follows: C#, JavaScript / UnityScript, Boo, IronPython, MoonSharp (Lua), C/C++ and Rust.

C#:

C# is a general-purpose programming language that supports a variety of programming disciplines. Commonly known as the best option for Unity.

Advantages:

- C# is the right language for anyone starting or of little experience in Unity.
- All Unity’s libraries are built in using C#.
- C# is easy to learn and far more accessible than other languages for Unity providing greater support.

Disadvantages:

- In terms of disadvantages, C# doesn’t have any when it comes to coding in Unity.

UnityScript:

This language is modelled after JavaScript and is designed specifically for Unity.

Advantages:

- This language is good for developers who come from a JavaScript background.
- Similar to JavaScript.

Disadvantages:

- Even though UnityScript is like JavaScript there are also a lot of differences such as classes and multiple variable declaration.
- A huge disadvantage is that Unity announced that they will withdraw support for this language. The use of this language would be unwise.
- Unity withdrew support for it.

IronPython:

Iron python is a variation of the python programming language. IronPython isn't a good language to develop games and to use this language you need to download the IronPython libraries form GitHub. Overall, this language isn't very good as Unity relies heavily on C#.

Lua or MoonSharp:

MoonSharp is more commonly used to act as a bridge to C#.

Advantages:

- it would be useful for users to create game mods.
- MoonSharp is worth considering if you want to interface with your C# code.
- It is also free to download from the Unity asset store.

Disadvantages:

- Low amount of support.
- Not very popular.

C++ :

This language would be mainly used for plugin creation. However, if you can use C++, you might as well just use C# instead.

Advantages:

- Good for plugin creation.

- Similar to C#.

Disadvantages:

- Very low popularity for Unity.
- Doesn't serve too much of a purpose.

Rust:
This is a relatively new language. it was created by Mozilla.

Advantages:

- This language is better to use than C++.
- Its function is to develop high-performance software in a shorter time limit.

Disadvantages:

- Not much point in using it for this project.
- Doesn't serve a purpose for this project.

Decision

The programming language C# was chosen because it is far superior compared to the other languages in Unity. There is a huge amount of support for it. It's extremely popular and all Unity's libraries are in C#. It's clear that C# works well with Unity and the statistics from the research I have done previous show that the decision to use C# is the correct choice.

Figure 11 - C#

2.3.3 Unity Script Editors

When using scripts and writing code in unity, a script editing tool is needed so that the developer can input the code and use it in Unity.

Below are the tools I have considered:

- Visual Studio Code
- MonoDevelop

Visual Studio Code

This tool is owned by Microsoft and has been integrated for Unity. It runs on all operating systems and is a very popular editor for Unity.

Advantages:

- Is good for editing and debugging.
- The C# features in Unity are well supported in visual studio.
- It's the most popular editor for unity. Giving it a lot of support.

Disadvantages:

- Visual studio takes longer to start up than other editors.

Monodevelop

Monodevelop is an integrated development tool that works for Unity that runs on all operating systems.

Advantages:

- Good for small Projects.
- It is an open-source environment.

Disadvantages:

- when using Monodevelop with unity it is less stable than Visual Studio.
- Even though it is good when dealing with small projects, Visual studio is better.

Decision

The decision was made to use Visual studio. It is far more popular than Monodevelop and is more stable with better autocomplete features. It works just as well if not better than Monodevelop with projects of any size which is perfect in relation to this project.

Figure 12 - Visual Studio

2.3.4 Unity 3D RenderPipeline

When using unity, a render pipeline needs to be picked. A render pipeline allows Unity to perform the steps needed so that it can render a 3D or 2D scene. This influences the performance of your Game.

When researching render pipeline I narrowed my decision down to two pipelines, HDRP and URP.

The Universal render Pipeline

This pipeline allows you to boost performance without having to use code to do it. It is compatible with all unity platforms and is the best pipeline.

The high-Definition render Pipeline

On terms of performance this render pipeline focuses more on the graphics and is suitable for high quality graphics projects.

Decision

From my research the URP has shown that it's the best option putting performance over everything else whilst still giving beautiful visuals. This pipeline will also be used in creating the inspect feature for my project by creating multi-layer rendering.

Figure 13 - Before and After using URP

2.3.5 3D Modelling

For this project, models of various animals will be needed. Instead of buying or downloading premade assets from the asset store. It was decided that I would be modelling the animals myself. This is because you often must pay for good quality assets. Modelling these animals, myself shows a higher level of skill and increased complexity of my project.

When doing 3D modelling you can either use Unity itself or you can choose an external modelling tool outside of Unity.

The 3D Modelling tools that were researched and considered are as follows:

- ZBrush
- SketchUp
- Blender
- 3Ds Max
- Maya

ZBrush

ZBrush is one of the most advanced 3D sculpting tools allowing the user to mimic traditional sculpting techniques. It was published in 2009 by Pixologic, Inc.

If you want to do detailed sculpting work with high-poly work, ZBrush is ideal. However, the learning curve is quite steep and it's not very good for renders. The cost of a ZBrush license is $795.

SketchUp

SketchUp can be used for a variety of 3D modelling. It has a large community which can be very useful and the capabilities of SketchUp are good. However, the rendering is limited, and coding is needed to be able to use it which will make it less efficient.

Blender

Blender is free and open source, and its modelling capabilities are very broad. Blender has tons of support and a large community making it easy to learn and use without much experience making it good for small developers. In terms of disadvantages, it doesn't have many except for the effects on AMD hardware making them slower. Some issues that might be experienced is exporting the models into unity. The textures and materials might not be exported correctly. Another issue is blender models can be made in detail and when using the models in unity they might cause unity to run slower or handling the models might cause problems for example if you created a tree model in Blender, there could be thousands of leaves on that tree. Unity could have a hard to processing the model. (9)

3Ds Max and Maya

3Ds Max and Maya are both made by the same company Autodesk. 3Ds Max has probably the broadest range of features out of all the modelling tools. It is mainly known for its modelling capabilities whilst Maya is more geared towards animation. Both software's take a long time to learn which would make them unsuitable to a developer like myself who is short on time. The two software's also cost money to use which is not ideal.

Decision

The 3D modelling tool Blender was chosen to use as it has lots of support online and its learning curve isn't as steep as the other. It is also free and more suited to the scale of my project. I have a small amount of experience in blender as well, blender was the obvious choice.

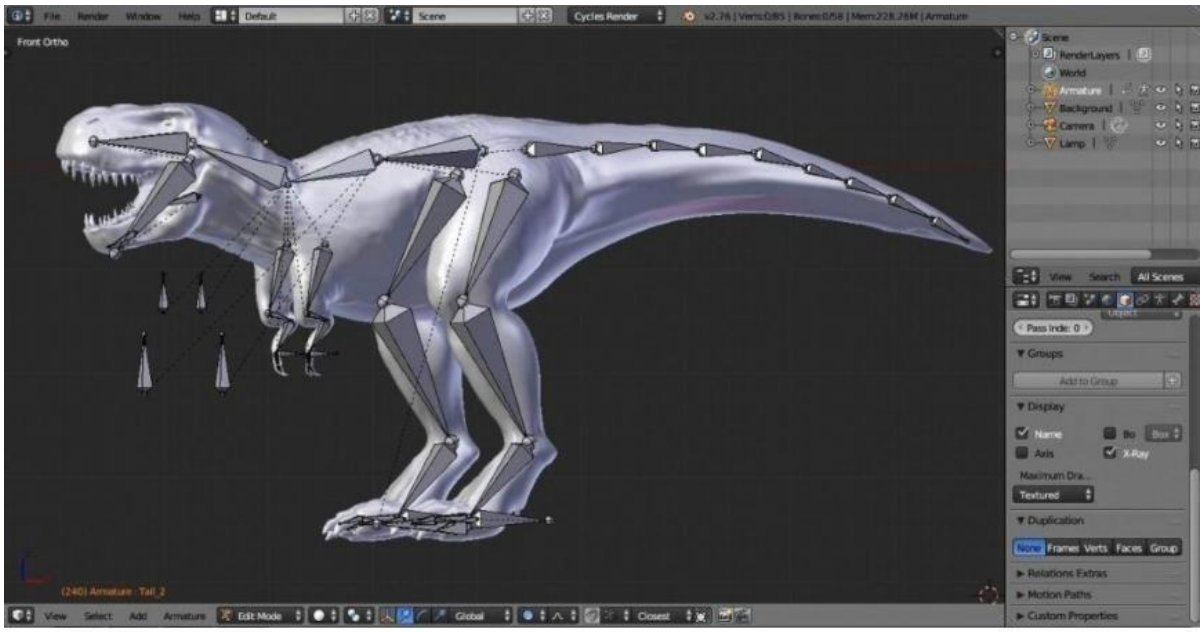


Figure 14 - Blender

2.4. Other Research

2.4.1 Cutscene Research

For this project, I will need to create and use cutscenes in Unity. In my research, I have found the best way accomplish this will be by using the Unity Timeline Editor, scripts written in C# and animations. (10)

Timeline Editor: In Unity, the Timeline Editor is built in tool which enables the user to create cinematic content such as cutscenes and audio sequences.

By using Timeline editor, camera objects in Unity will be able to use animations and audio sequences and piece them together to make cutscenes which can be controlled using C# scripts which can attach to the cameras.

2.4.2. Quiz System Research

As this is an educational game, the user will need some way of testing themselves on the information they have learnt. To do this, research into creating quizzes in Unity will be researched.

Below is the research that was looked at:

First, the best techniques were looked at for designing Multiple choice quizzes. After looking at various sources, (31) there were five different techniques that were decided to

be followed. They are listed below:

1. Don’t list too many answers.
2. Avoid trick questions.
3. Use simple question formats.
4. Make tests challenging, but not too difficult.
5. Follow up with feedback.

By following these tips, the quiz will be designed in an efficient and high-quality manner.

Next, the different coding techniques for creating a quiz in Unity were researched. There is an abundance of ways to accomplish this in Unity, but the two different ways that were researched was to either create a very basic quiz using a small number of classes and a series of case statements where the questions and answers were hardcoded, or to use a method which is considered better practice by creating lots of different classes to manage the game and using scriptable objects. This will allow the developer to create questions from within unity instead of hardcoding and allow the developer to be able to customize the quiz system to their preference.

Decision

It was decided to use the more complex method, where scriptable objects, structs, and various classes and coding techniques to create a more complex, efficient, and more customizable quiz system.

2.4.3. Inventory System Research

In this project an inventory system will be used to store wildlife that the player has found. In this section the deciding on what method of how this inventory system will be created is being researched.

When researching how to create an inventory system in Unity, there were two methods looked at. The first one was to download a premade inventory asset from the unity asset store. The second method was creating an inventory system in Unity using C# to write scripts and classes to build a system that handles items and has a User Interface to display the inventory. (11)

When using a premade inventory system, the advantages were:

- Less workload involved in process of creation.
- Quality of inventory systems were high.
- Easy to use.
- Saves times on learning how to make an inventory system.

Disadvantages:

- All the premade systems cost money.
- Takes away from the complexity of the project.
- Harder to customize or change the inventory system.
- Might not suit the specifications of this project.

When creating an inventory system in Unity the advantages were:

- Self-made inventory system is more customizable.
- Can be tapered to the exact specifications of this project.
- Adds more complexity to the project.
- Is free to create the system in Unity.

Disadvantages:

- The quality of this inventory system might not meet the standard of the project.
- More workload involved in this process.
- Takes more time to learn how to create this system.

Decision

The method chosen to do was creating a self-made Inventory system in Unity. This was chosen was because when writing the code and creating the user interface yourself, it allows for better customization to fit the specifications of this project, and this games system. It adds more complexity to the project instead of just using a premade inventory system making the self-made system the better choice.

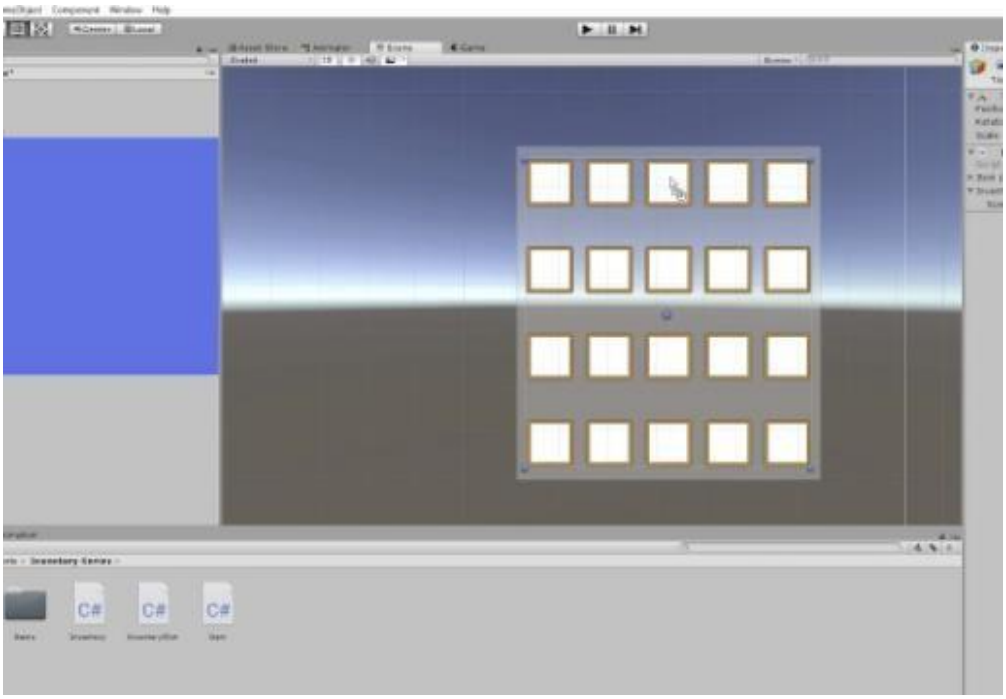


Figure 15 - Sample Inventory System

2.4.4. Inspection system Research

In this project an inspection system to inspect various types of wildlife will be needed. Research on what method was going to be used was researched in this section.

When researching how to create an inventory system. There were two methods researched. The first method was creating an inspection system in Unity and the second was using a premade asset from the asset store.

The first method consists of creating various C# scripts and classes in unity, and using a technique called Ray casting to create this system. Ray casting is when an invisible ray goes from one point to another and detects whether an object is in the path of the ray.(12)

The advantages of creating the system in Unity are:

- Easier to taper the specifications to the game.
- Doesn't cost any money.
- Adds complexity to the project.
- Ray casting is a reliable and efficient method.

Disadvantages:

- Higher workload and difficult to complete.
- Raycasting can be difficult to learn.
- Quality of system might not be as good as the premade asset.

Advantages of downloading a premade inspection system:

- The quality of the system could be very high.
- Easy to use the systems.
- Saves a lot of time creating one.

Disadvantages:

- All the premade systems cost money to buy.
- The project would be less complex if used.
- Harder to customize to suit specifications of the game.

Decision

It was decided to create the inspection system in Unity and not download a premade one. Whilst the premade ones are of high quality and quicker to implement, the custom inspection system will be more suited to the specifications of this project, and it doesn't cost any money to create.

Figure 16 - Sample Prompt System

2.4.5. Save system Research

In this project, the user will need to save their data and their progress in game. For this a method of saving the data will be used. Normally, for small amounts of data you can just use the built in PlayerPrefs (class that stores preferences) in Unity, but for this project other tools will be needed.

The different ways for saving data that were researched and considered are as follows:

- XML and JSON files
- Binary Files
- Easy Save – Asset
- MySQL with PHP

XML and JSON: This way is quite simple to do but it's not very efficient, and the files are easily modified by other users which is a security risk. These files are also used outside of Unity.

Binary Files: Binary files are a lot harder to modify making the security a lot better and are used inside Unity. These files are perfect for games that want a more complex save system than just using the PlayerPrefs in Unity. However, it can become slow to work with binary files as your project gets bigger. to use binary files in Unity you must write C# scripts and create classes to implement this system.

Easy Save: An easier way and a long-term solution instead of binary files would be a free asset like "Easy save". It saves a lot of time and allows you to save unity specific data, encrypt the data, handle a lot of data at once and save and load data from the internet.

MySQL with PHP: When using MySQL with unity there is a steep learning curve compared to other options. For projects like this MySQL is probably not needed for what is needed to be accomplished.

Decision

The method of using binary files to create a save system was chosen because it is more efficient than the other methods listed. It is well suited to the size and type of game that is being created for this project and it will allow us to create a system using C# scripts and classes so the user can save and load their game.

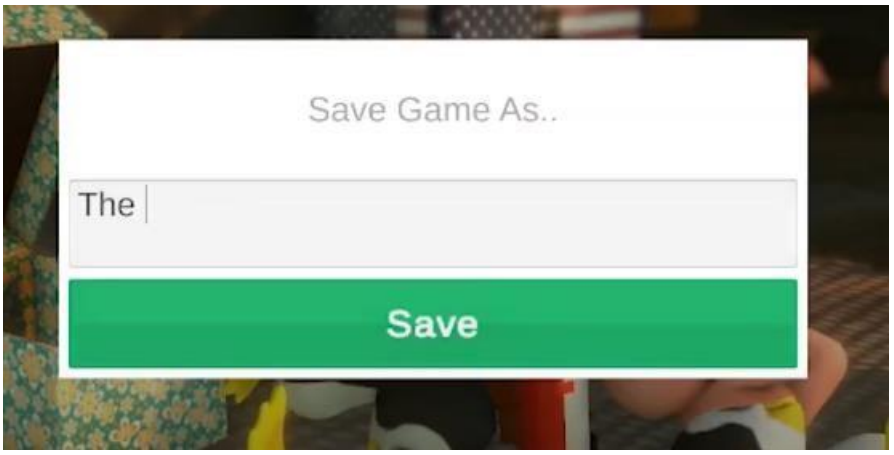


Figure 17 - Sample Save System

2.4.6. Procedural Generation

Procedural generation is a programming technique that allows the developer to create data according to the programs instructions that were made by the programmer. During the development process, lots of different games generate parts of the environment or non-player characters procedurally to save time, instead of creating the assets individually. An example of this is if the developer wanted to create a forest in Unity, instead of creating each tree and bit of grass individually, they would write a program to generate the forest all at once when the game starts.

Terrain

In Unity to be able to procedurally generate terrain or levels, you must use a noise function. noise functions mean that they generate pseudorandom values.

The noise that will be used in this project will be Perlin noise. This allows you to write code to generate pseudo random patterns that consists of waves that increase and decrease across the terrain or level. An example of this would be in Minecraft where each time a world is created, it is pseudo randomly generated by code using Perlin noise. This will be used to generate the map terrain and the forest wildlife in the game.(13)

Advantages:

- This method can create lots of content for a game.
- Can create large games more efficiently.
- Can be used in low budget games but make them seem like they are higher budget.
- Better gameplay variety.

Disadvantages:

- Uses more hardware resources.
- Creating scripts for this method is harder.
- terrain can be repetitive.

Reason for choosing:

The main reason for choosing this method is because it is suited to this low budget project. It is efficient at creating large worlds whilst adding an element of complexity to the project when coding. It also allows the game to generate a new world each time they start a new game.

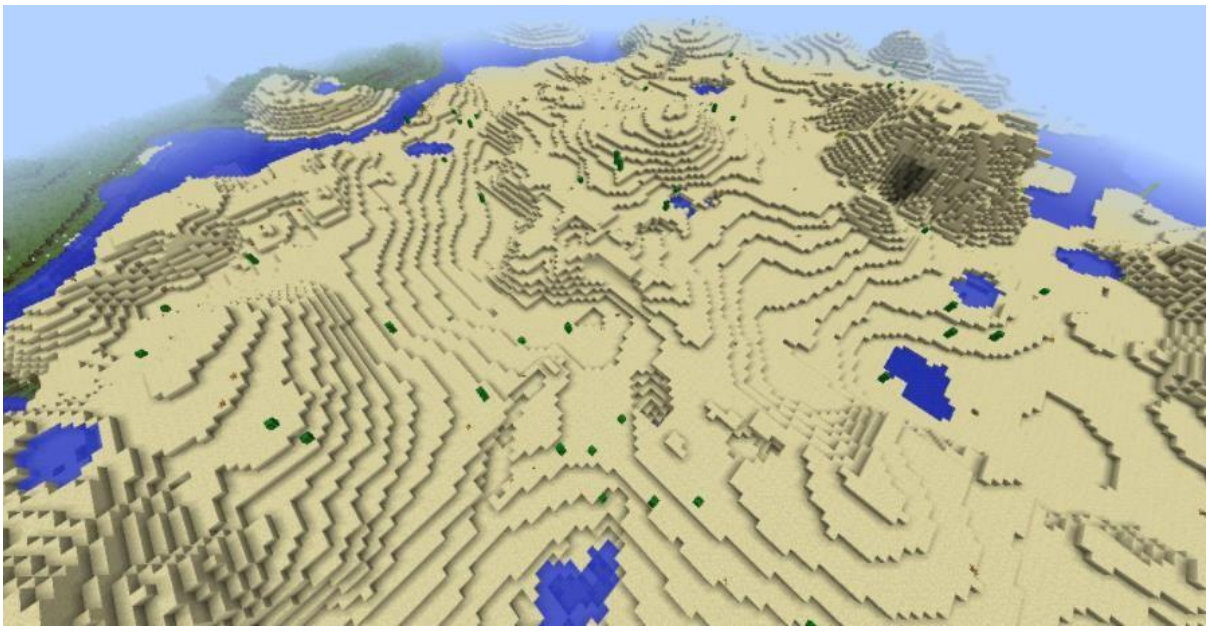


Figure 18 - Minecraft Perlin Noise

2.4.7. AI Research

In this project a form of AI will be needed to allow the animals in the game to interact with their surroundings, the user, and behave in a certain way. To achieve this, various forms of AI were researched and decided upon to be used in this project.

Finite State Machines

The main form of AI and behavioural management that was looked at was Finite State Machines. The idea of an FSM is that a machine can be in a certain state from a range of different given states. For example, in a game if an AI player is given the state of walking around and it detects another nearby player it will go into an attack state, however, if the player moves out of sight the AI will go back to wandering.

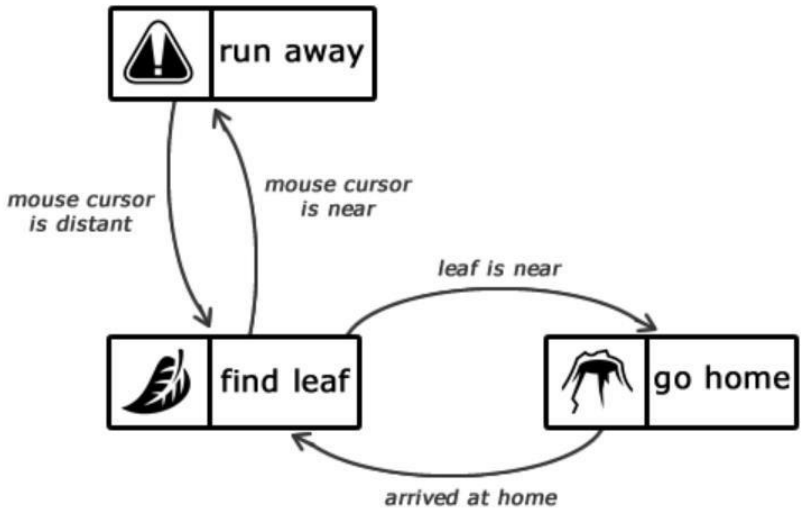


Figure 19 - FSM

Bolt

When researching different techniques of AI, I came across a visual scripting tool for unity called Bolt. This technique allows you to work on unity projects and aspects such as AI without using any code. This could be a solution for tasks that are too difficult and take up too much time to complete within the given time constraints.

ML-Agents

In Unity, there is a plugin called ML-Agents. This plugin enables the developer to create

environments that train the AI agents in their game. This means that the agents will be able to learn by themselves. This method is used in AAA games and allows you to teach intelligent behaviours to AI agents. This method tends to have a higher level of difficulty, but it also has lots of support from Unity.

Figure 20 - ML Agents

AI Pathfinding using Unity's NavMesh System

In games, characters often must navigate around obstacles in the level. This is a common scenario in games, so Unity provides a built-in pathfinding system, called NavMesh. This NavMesh system allows you to write C# scripts to set instructions for how the non-Playable character, in our case an animal should act with the environment.

An example of this, if the animal wanted to navigate around a specific area but if it came within a certain distance of our player, the animal would run away or maybe chases after the player according to the instructions in the C# script that were set by the programmer using the NavMesh AI system.

From my research I have found that NavMesh is very helpful and allows you to do hundreds of pathfinding calls per frame which is very efficient. This is a component that will certainly have a part to play in this project. It will allow me to create C# scripts and write programs that will be able to set specific behaviours for each animal in this game.(14)

An issue that might arise when using NavMesh is when the NPC's need to navigate around

obstacles and terrain. Whilst researching Unity's pathfinding system there seems to be lots of minor error and bugs with the NavMesh system, when navigating around complicated obstacles. For example, if characters are moving through narrow pathways; they seem to get stuck and are unable to move away from the other characters.

Decision

The method that was chosen for the animal AI in this project was Unity's AI pathfinding system NavMesh. The reason for this is because it will allow me to write C# scripts for each animal in the game to set what behaviours they have and how they will react to the environment and the player. This method of AI is perfect for this project because it isn't too complex making it suitable for a small development team such as mine, but it also has a coding element to it which still adds a certain level of complexity to the AI.

2.4.8. Animation Techniques

Keyframe Animation

Keyframing is the most popular and most simple animation technique. This technique allows you to draw a shot and move different parts of your object to then measure the position and time in frames. For example, the animator moves a limb of a character's body whose movement is saved each keyframe. When this is completed, the movement is turned into an animation. An advantage of this is that the movements creating can be made to look super realistic. However, it is quite hard to make these movements as realistic as they can be.

Motion Capture Animation

Motion capture is a technique that uses reality and animation combined. This method means you must use equipment such as bodysuits for a person to act out the movements and then are recorded as animations. This method is used in AAA games as it allows the animators to capture realistic facial and body expressions, but it generally cost a lot more to do compared to other animation techniques.
For this project the two best options for animation tools are the Animation Controller in Unity or the animation editors and tools in Blender.

Procedural animation

Procedural animation is a type of animation that will generate the animation automatically rather than creating the actions in advance. This is done by writing code, for

this project the code would be written in C# scripts in Unity and attached to the Player. This technique doesn't look as good as other animation techniques but is far more efficient. It allows the developer to re-use all the animations on the different characters as it can use similar code to drive these animations. (15) An example of this is if you have multiple animals that are similar in the way they move. You would only need to change the code slightly for each animal to animate them. When using other forms of animation this is not the case.

Rigging

For animations to be added to a character model such as an animal or a player, a rig must be created. A rig is a skeleton that is made of shapes to simulate a skeleton. This skeleton is attached to the model and can move the models' limbs around like in real life. Rigging is necessary to animate animals and will be needed in this game.
This is an example of a rig for a cheetah that acts as a skeleton for the animal:

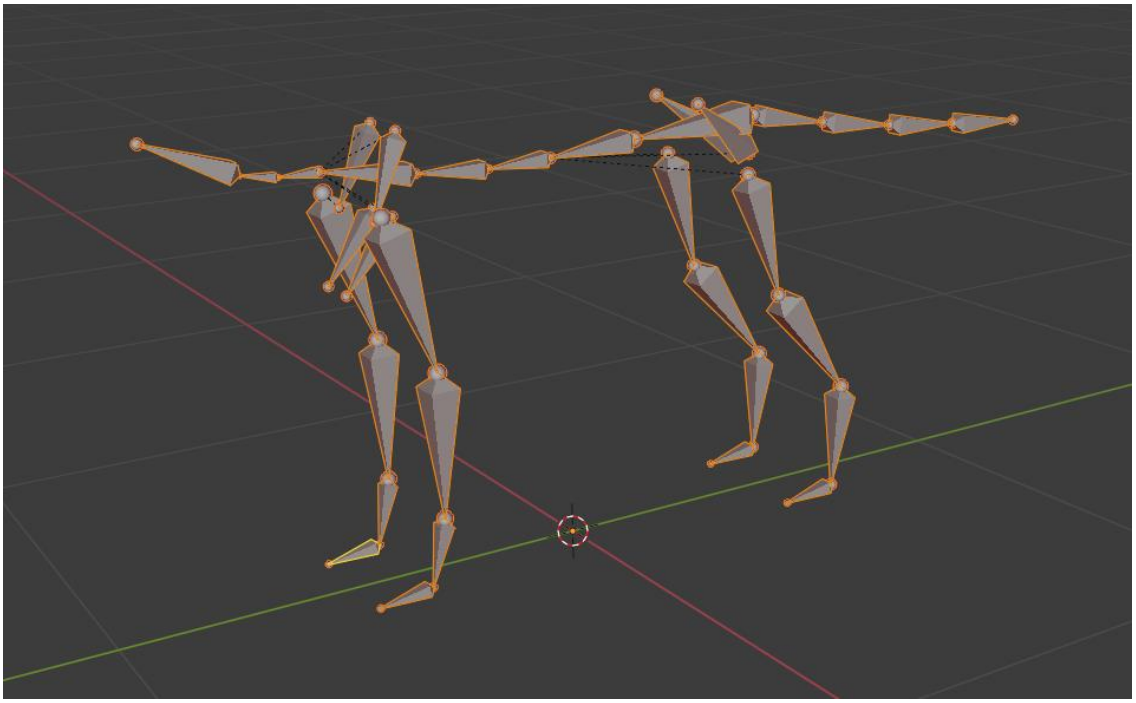


Figure 21 - Example of Animal Rig

Decision

The decision was made to use keyframe animation. This technique was chosen as there is limited time for this project meaning other animation techniques will be quite complex to complete. Keyframe animation will enable the animals to be animated in Blender and

exported straight to Unity to be used. All the animal models will also use rigging.

2.4.9. Audio in Unity

When researching audio in unity, the research has shown unity's build in components are best for audio in unity. Components such as audio player, audio listener and audio manager combined, along with the use of scripts. They are simple to use and will suffice for this project.

When searching for downloadable music and sounds for this project. A website called mixkit.com was found. This site offers free downloadable audio sources that could be put in this game.

The decision was made to use this website and Unity's built-in audio features because they are easy to use and don't need to be complex.

2.4.10 Unity Assets

Even though many assets in this project will be modelled either in blender or unity, there are still some that will be downloaded from the Unity Asset store.
These assets include:

- The Sky Shader.
- Frog model.
- Grass Shader.

2.5. Existing Final YearProjects

Two final year projects from previous years were researched. These projects were both in some way relevant to this project to aid in the research and design.

2.5.1. Project No.1

Title: EvolVR – Investigating Procedural Ecosystems and Evolution
Student: Ryan Byrne

The goal of this project was to create a virtually simulated ecosystem of animals from multiple animal groups to better demonstrate the evolutions animals make over multiple generations. The environment is procedurally generated so that animals can be viewed adapting to a multitude of different habitats of varying hostility and seeing how traits of the animal might develop differently from one placed in a different bionetwork. The user will be able to traverse through the world in a safari-like experience where they can get close to animals and see their behaviours in a much more personal and realistic manner as if they were a real-life animal zoologist. (16)

This previous year project influenced this project as it had aspects of wildlife and procedural generation in it. The approach to procedurally generate the terrain added a level of complexity to the project that inspired the method that will be used in this project to create the terrain.

2.5.2. Project No. 2

Title: How Games Can be Augmented for Inclusion
Student: Max Blennerhassett

This project attempts to address the problem of games being inaccessible to sensory impaired individuals, particularly those with eyesight impairments. The game created for this project is named “Pentris”. Pentris is a game focused on accessibility, it can allow far more focus for accessibility rather than actual
gameplay or narrative. Depending on the game’s design, particularly the genre it classes itself as, a game may be far more limited in the level of accessibility functionality that can be realistically implemented without changing the core design of the game. (17)

This previous year project influenced this project as it tries to address an issue with games being inaccessible to a particular user. In the case of this project the aim is to address the issue of normal educational techniques having low interaction, particularly with students.

2.6. Conclusions

This Section of this report explores technologies surrounding this project. The relevant research was considered, described, compared, and decided on whether it was suitable for the system being created. Alternative solutions to this project were explored. Games engines, programming languages and modelling tools were also looked at. Other research that was done such as animation techniques, AI and assets were explored. Finally, previous year projects were then considered.

3. Design

3.1. Introduction

In this section, all aspects concerning the design of this project are discussed. The Project setup, software methodology, and system architecture is discussed. The requirements are gathered and then the front end is created. Two medium fidelity prototypes are shown along with multiple use cases. The system design is then discussed and finally, the models and the audio are presented. (29)

3.2. Version Control and Setup

The first step in starting this project is setting up a GitHub Repository (30) for Version Control. This means the work that is done on the project will be backed up and saved online. This will prevent loss of data. in terms of compatibility, GitHub is the perfect application to use with Unity as it is easy to push and pull data.

Team Gantt and Mila note were used for project management tools. These were used to document any ideas for the project and to plan out each step of the process. This made it easy to track the progress of the project and keep up with the deadlines.

3.3. Software Methodology

When considering the different design methodologies for this project, there are several choices. The Three main options that were considered were the ADDIE model, Agile Design and Design Thinking. (18)

Agile Design

This method is used for breaking up tasks and decreasing the amount of planning. These tasks generally have short time frames that will only last a few weeks. This methodology lets the designers work with the other team members and gets feedback by showing the customers the progress. The customer interaction is a big part of this model.

Figure 22 - Agile Model

Design thinking

This model is used to solve complex problems and develop ways to solve these problems. The focus of this model is on the problem and product. However, this model also uses customer input to solve these problems as well.

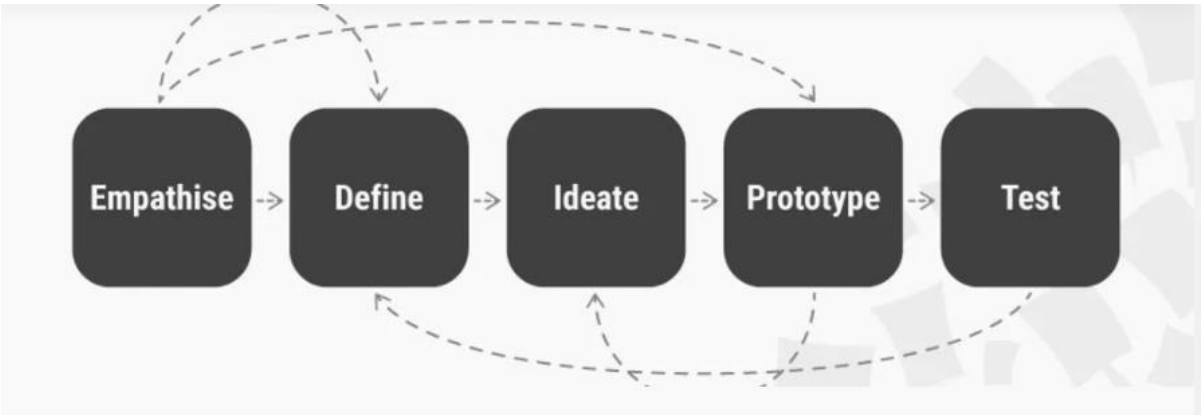


Figure 23 - Design Thinking Model

ADDIE Model

The Final methodology and the Chosen model is used in this project is the ADDIE model. This model is mainly used in educational applications for instruction and learning. It is the most common model for instructional design, and it is quite basic but very effective. (19) There are 5 Main Stages to the ADDIE Model:

- Analysis
- Design
- Development
- Implementation
- Evaluation

This model requires each stage to be done in order and then focuses on reflection and iteration.

The reasons behind selecting this methodology were because the ADDIE is perfect for instructional design which this game focuses heavily upon. The model is the most common model and has been used for a long time proving its reliability. It creates clear learning objectives and well-structured content whilst also putting emphasis on assessment which helps with leaning outcomes.

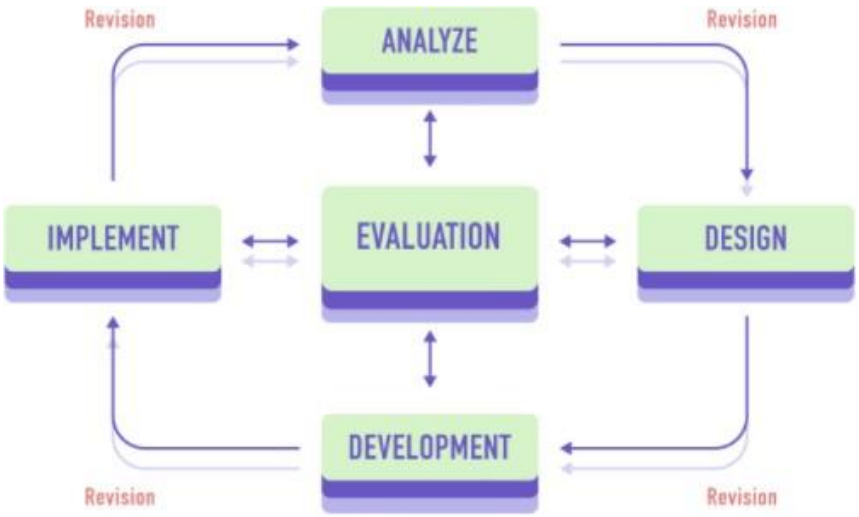


Figure 24 - ADDIE Model

3.4. Overview of System

3.4.1. Architecture

The architecture of the system organizes the components into layers. All the components are on the user's computer, so the architecture is single tiered. The Unity System architecture is used. This shows how all the different components and different software of the system will communicate. (20)

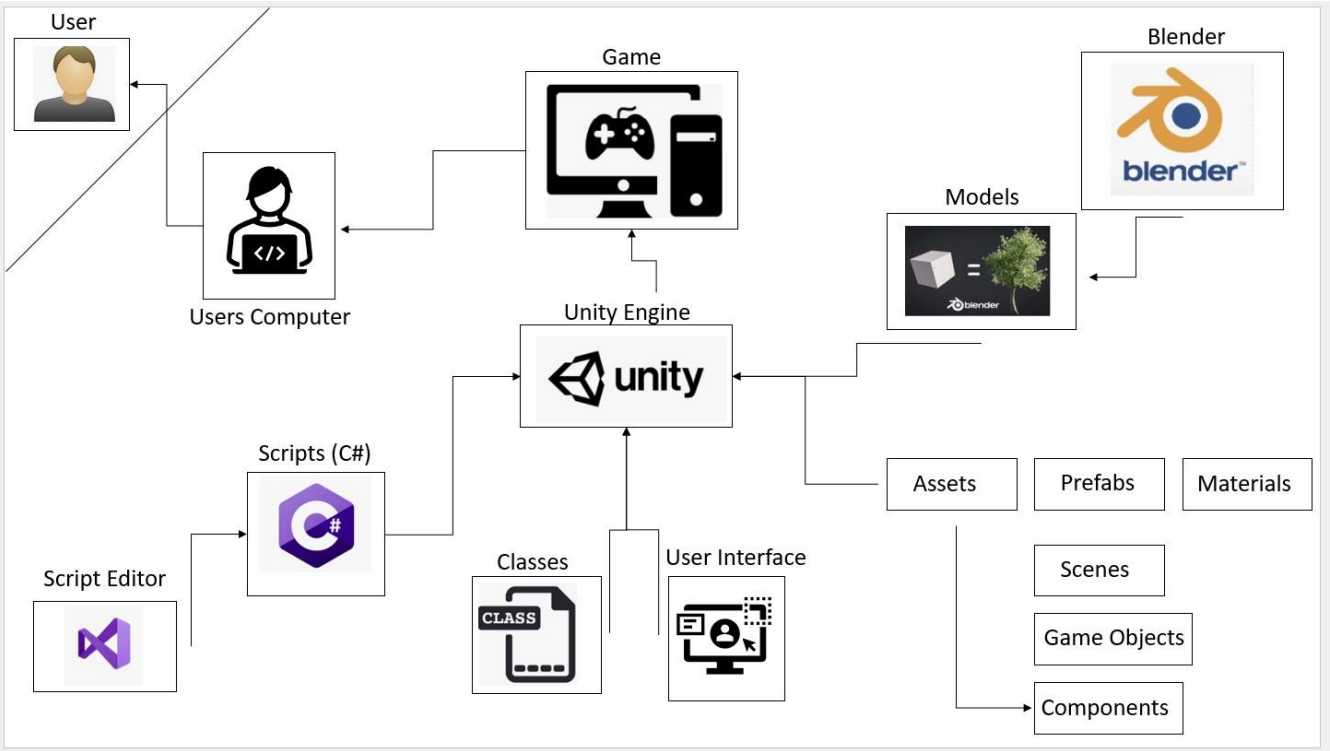


Figure 25 - System Architecture

3.4.3.1 Requirements Table

The table below shows the requirements for this system. The table contains the task ID, the task, the priority of that task, and a brief description of the requirement.

The three different levels of priority are as follows:

Low: Not essential for project completion, low priority task should only be completed once all medium tasks have been fulfilled.

Medium: Hold some importance for completion of the system but should only be completed after the High priority tasks have been successfully accomplished.

High: These requirements must be completed for this system to work. They hold the most priority.

ID	Task	Priority	Description
1	Loading Screen	Medium	When launching the game, loading into the terrain, or accessing anything in the game that takes more than a few seconds to render or load, a loading screen will be displayed. This will show the user that it is loading indicating that it will take a few more seconds or minutes.
2	Start Menu	High	Start menu will be the first interactable screen displayed in the game. This screen will be needed to access all the functions of the game. Start game, options, quit game.
3	Pause Menu	Medium	Pause menu used whilst moving throughout the terrain. Gives the user the option to quit, access options, save game, and resume.
4	Cut Scene	Medium	A cut scene will play when the user starts the game, this will explain what the game is about and how to play it
5	Narrator	Low	A narrator will talk during the cutscene. Using a pre-recorded sample or Microsoft edge.
6	Terrain Generation	High	The environment in which the user will be exploring will be procedurally generated using code. Trees, rocks bushes, etc.
7	Character Controller	High	A character controller will be used to the user can move throughout the terrain and play the game.
8	Character Model	Medium	Create a character model.

9	Animal Modelling	High	Model animals inside of blender. Export them to unity.
10	Animal Rigging	High	Rig animals so they can be animated.
11	Animal AI	High	Use code to create realistic animal behaviours.
12	Plant Modelling	Medium	Model plants and trees.
13	Object Scanning	Medium	Scan objects using 3d scanner to be used as assets.
14	Database Creation	High	Use database to store information for animals and notebook feature, be able to save the game.
15	Music	Low	Add background music.
16	Audio	Low	Add audio such as animal sounds, terrain sounds.
17	Inspect Feature	High	Inspect feature allows user to inspect wildlife so they can view information on it in the notebook.
18	Notebook Feature	High	Educational feature with information on wildlife. User can access it at any time.
19	Information feature	Medium	View relevant information.
20	HUD	Low	HUD displayed on screen when user is moving throughout the terrain.
21	Quizzes	Medium	Quiz used to test user on information learned.

22	Controls	Low	Control option display in menu
23	Instructions	Low	Instruction displays in menu.
24	Download Game	Medium	Allow user to download game form the internet.
25	Tutorial	Low	Tutorial at start of the game.
26	Videos Info	Low	Videos displayed on notebook for source of information.
27	Adding to database	Medium	Adding information and wildlife to notebook/database.
28	Animal Movement	High	Animating animal movement so they can imitate realistic animal behaviour.
29	Animation	Medium	Animation.
30	Exporting Phase	High	Export and render game.
31	Testing	High	Test all system requirements.

3.5. Front-End

The front end of this project consists of menus to navigate through the game, the rest of the game is set in a 3D environment which will be demonstrated through the user's view.

3.5.1. Prototypes

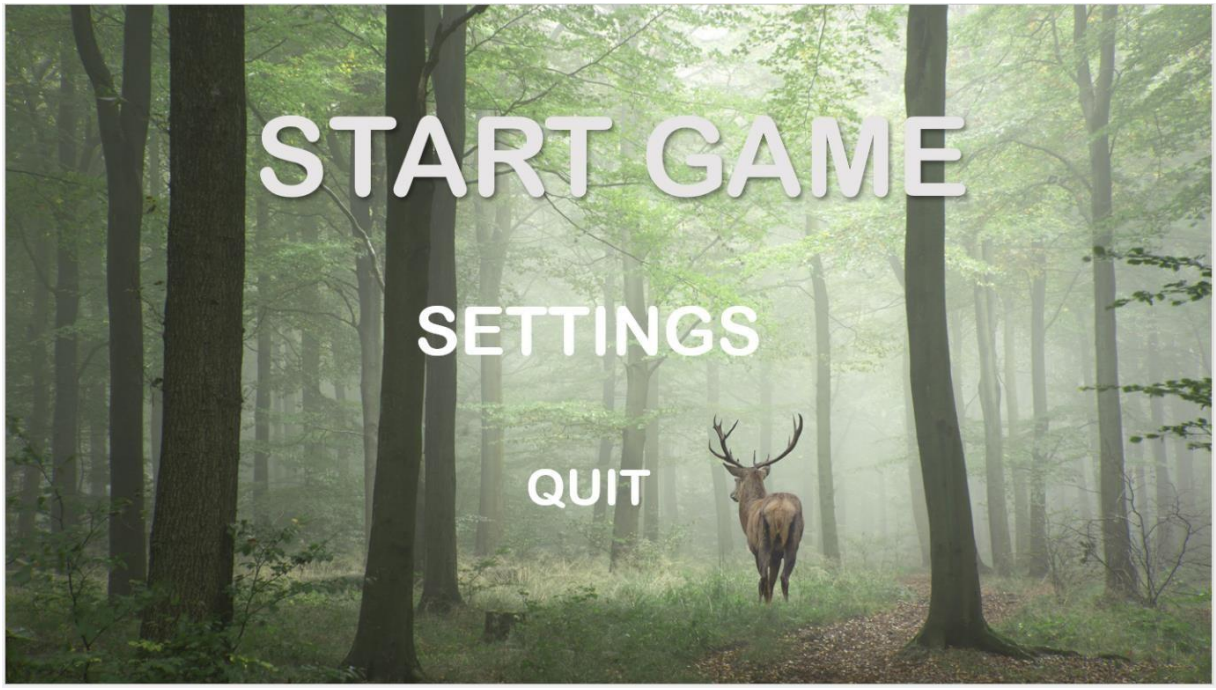
When designing the UI for the game. Two prototypes were made.

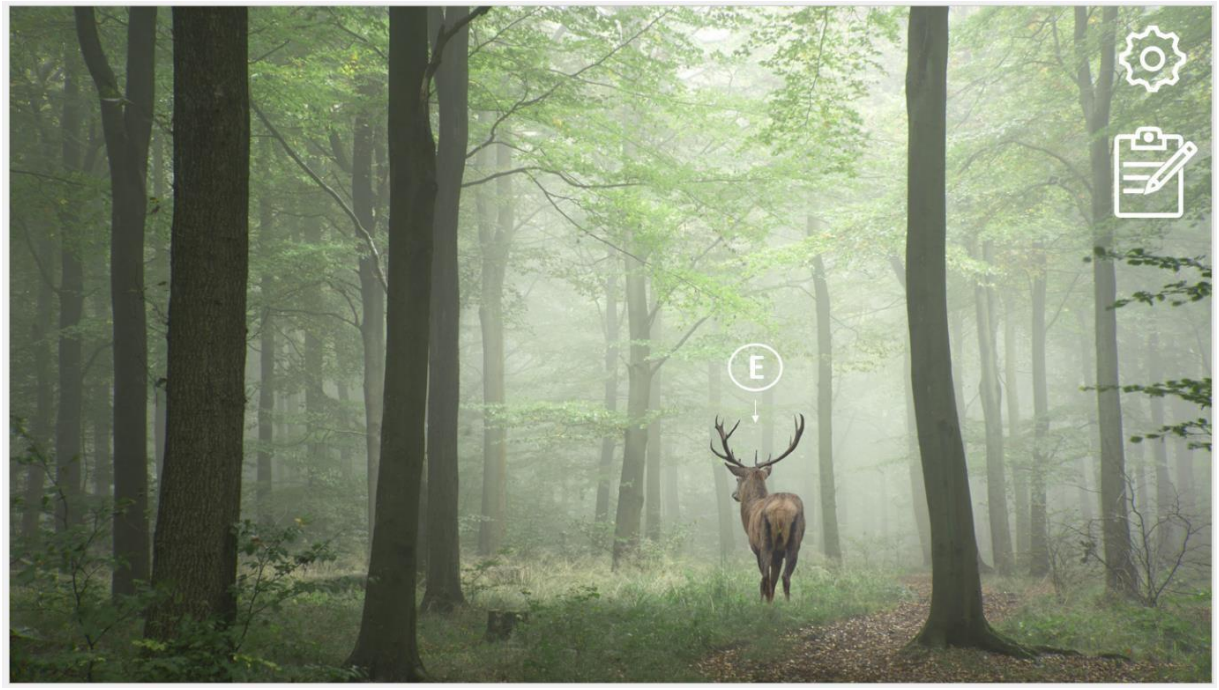
For the first iteration of prototypes, Screen layouts were made on PowerPoint. These prototypes were made to plan out the design process of the game and show what it would look like before making it in Unity.

The second prototype was made after more research was done on exactly how the interfaces and game was going to work. This prototype gives descriptions and a more accurate and detailed representation of what the final product might look like.

First Prototype iteration

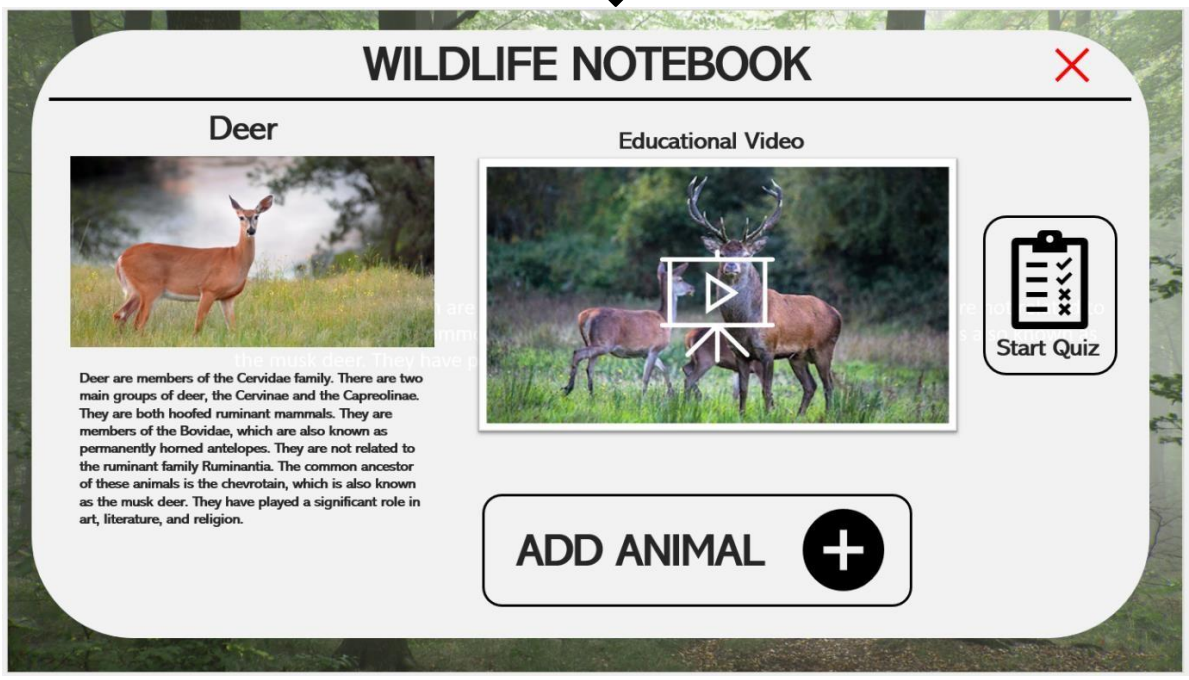
Figure 26 - First Iteration Prototypes

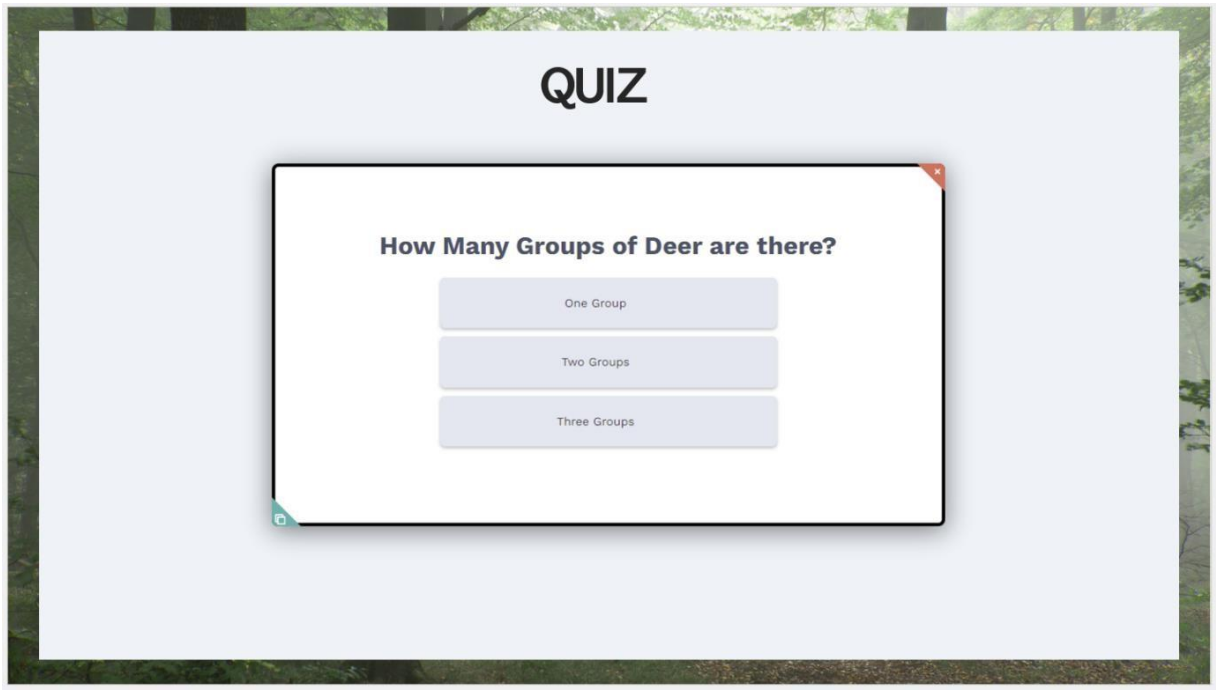


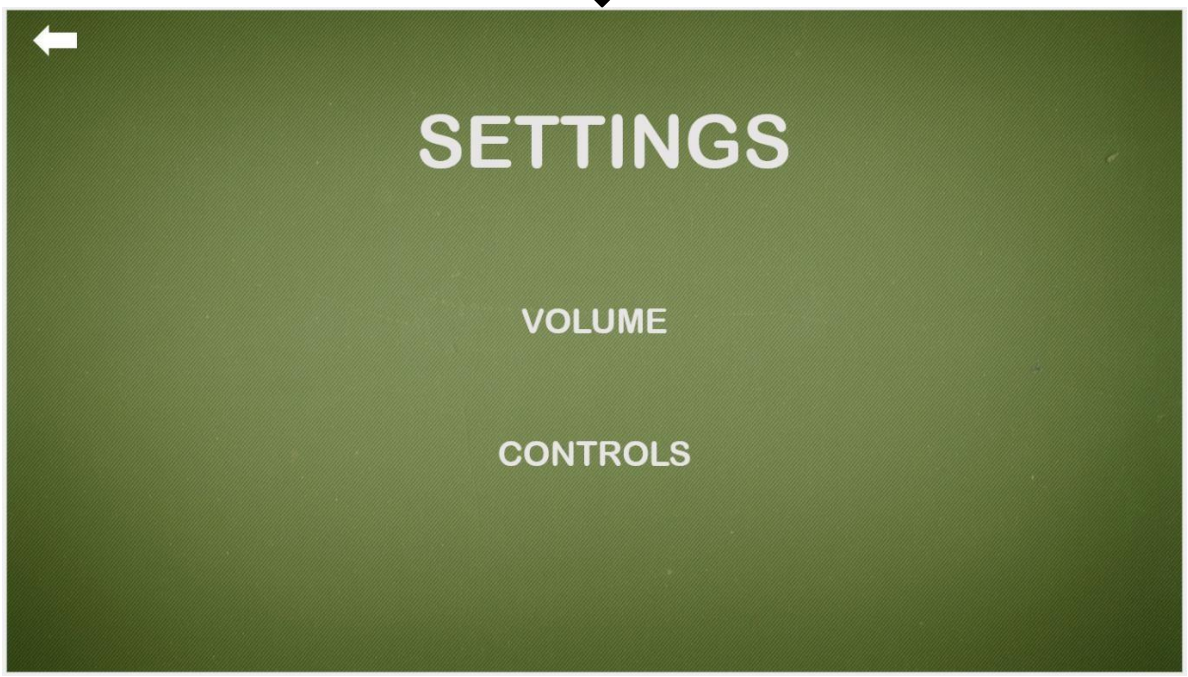

E


CONTINUE
OPTIONS
LEAVE GAME

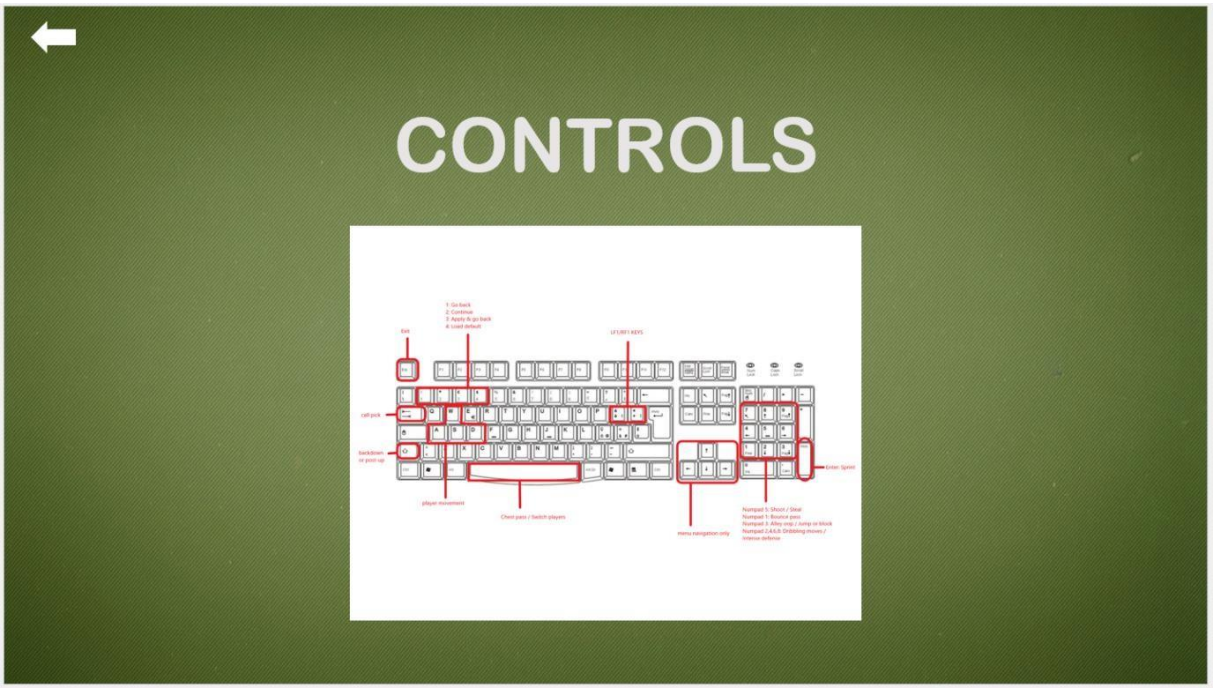

CONTINUE
OPTIONS
LEAVE GAME


WILDLIFE NOTEBOOK
Deer
Deer are members of the Cervidae family. There are two main groups of deer, the Cervinae and the Capreolinae. They are both hoofed ruminant mammals. They are members of the Bovidae, which are also known as permanently horned antelopes. They are not related to the ruminant family Ruminantia. The common ancestor of these animals is the chevrotain, which is also known as the musk deer. They have played a significant role in art, literature, and religion.
Educational Video
Start Quiz
ADD ANIMAL


QUIZ
How Many Groups of Deer are there?
One Group
Two Groups
Three Groups


SETTINGS
VOLUME
CONTROLS


LOADING...


CONTROLS

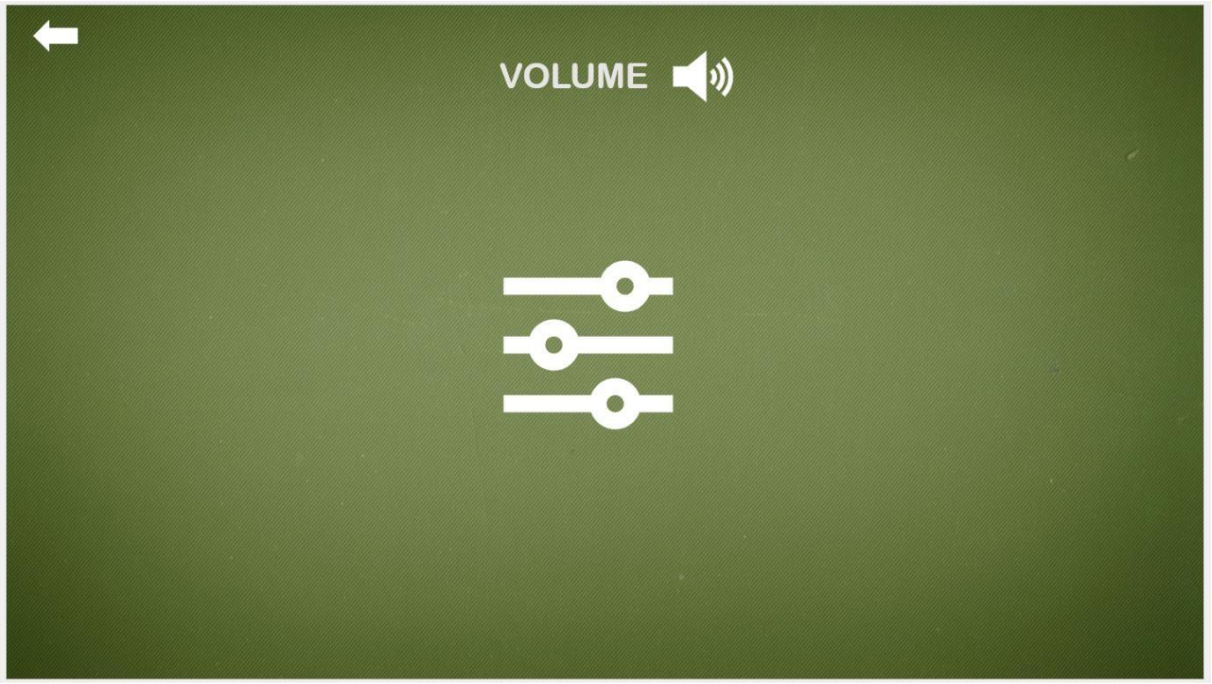


Figure 27 - End Iteration

Second Prototype Iteration

The second iteration of the prototype design discussed and shown below:

This prototype provides a lot more detail of how the system will work as it considers the research done for the design.

The screens will be shown as if the user is using the game in order. The first screen that the user will see is the loading screen. This screen will appear once the user has launched the game, this screen will also appear whenever the game is loading something or initializing something.

The loading screen will have a background picture from the game of a nature scene. It will then have a progress bar on the screen to show when the next scene will be loaded along with text saying, “LOADING GAME” or “LOADING ENVIRONMENTS”.

Loading Screen:

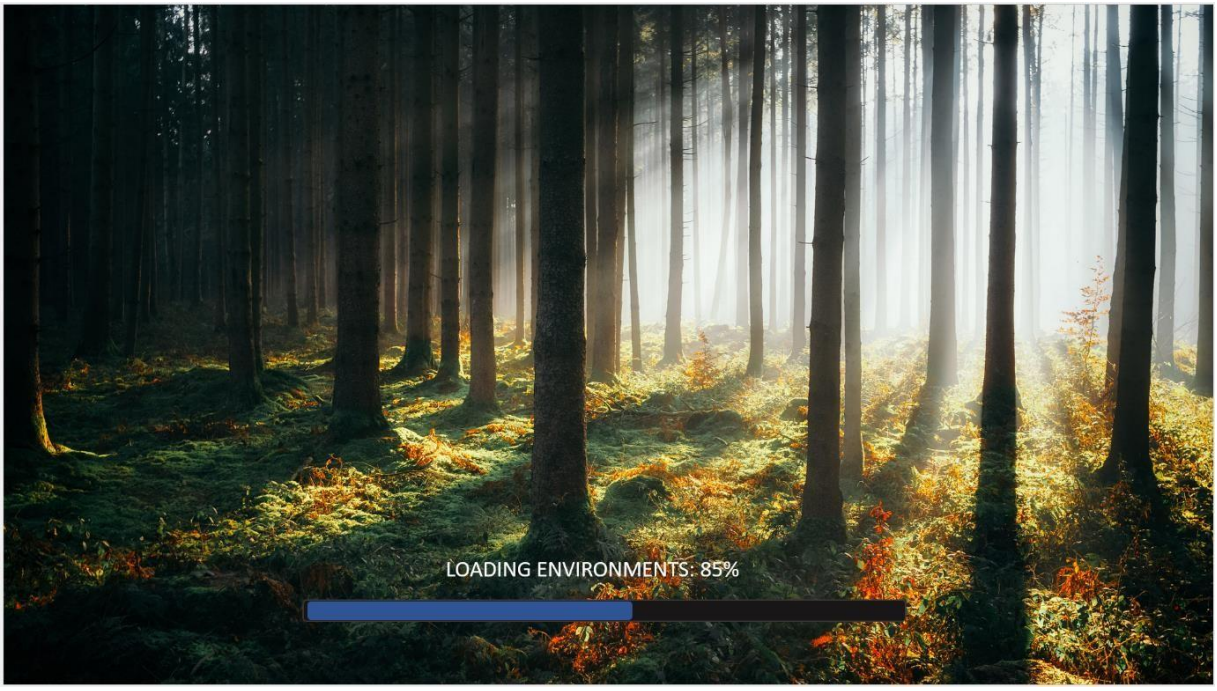


Figure 29 - Loading Screen

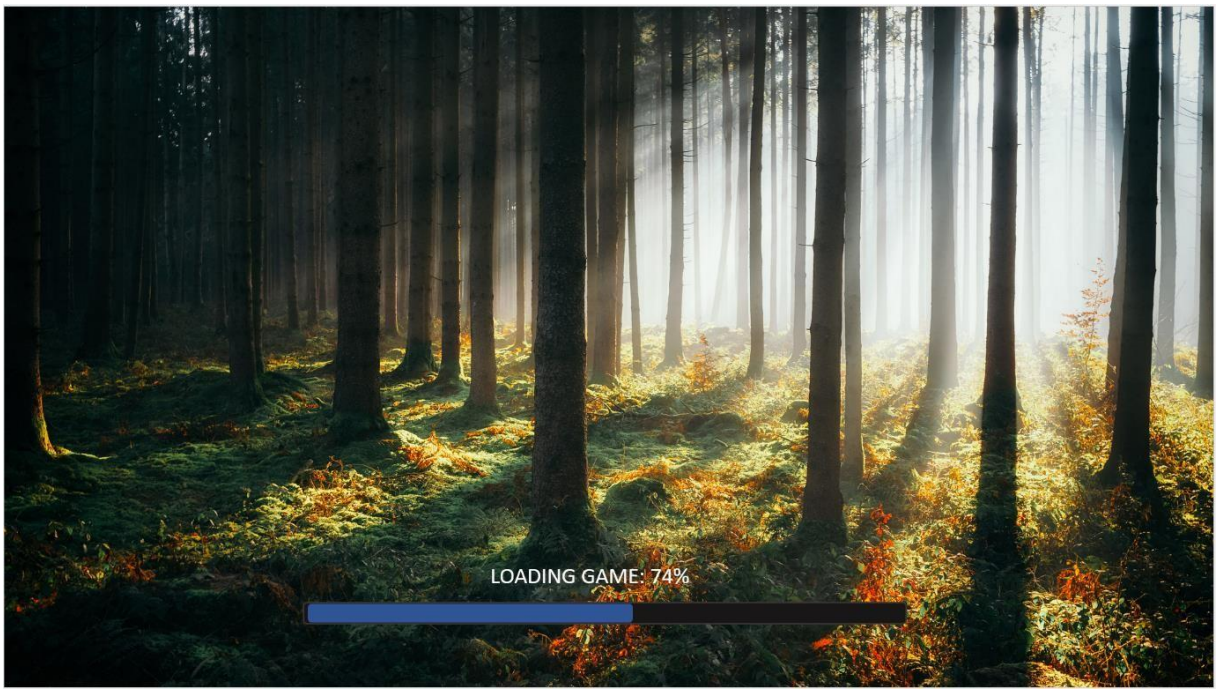


Figure 28 - Loading Screen 2

The next screen the user will see is the Start Menu screen. This screen will be the main navigation screen for the game in which the user will have three options. A “PLAY” button and load into the environment, a button to access
the “OPTIONS”, and a “LEAVE GAME” button where the user can exit the application.

The start Menu will have a live background. This will consist of a scene created from the game assets with trees and plants along with a red squirrel character model from the game in the background. The squirrel will be animated.

Start Menu:

Figure 30 - Start Menu

The next screen shows the “OPTIONS” screen when the button is clicked. This will use the same background and layout style as the start menu. The “OPTIONS” menu consists of a volume adjuster and an image of the controls. The volume adjuster will allow the user to turn down the volume and the controls will show what buttons to press to use the game.

Options Screen:

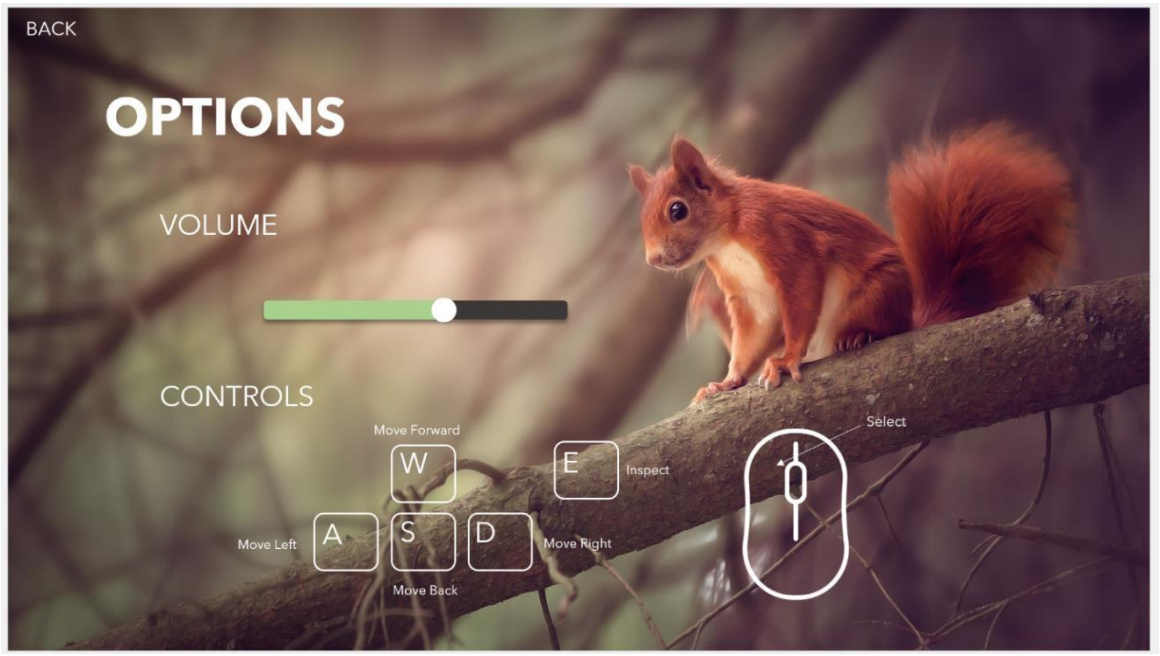


Figure 310 - Options menu

To exit this screen the user will be able to select the back button.

When the user selects the Play button, they will load into the forest environment, where they can explore and look for wildlife. Below is a sample picture of what they will see along with the HUD. The HUD has a pause menu icon, an inventory icon, and an inspect icon will also appear over wildlife that the user can inspect.

Explore Screen:

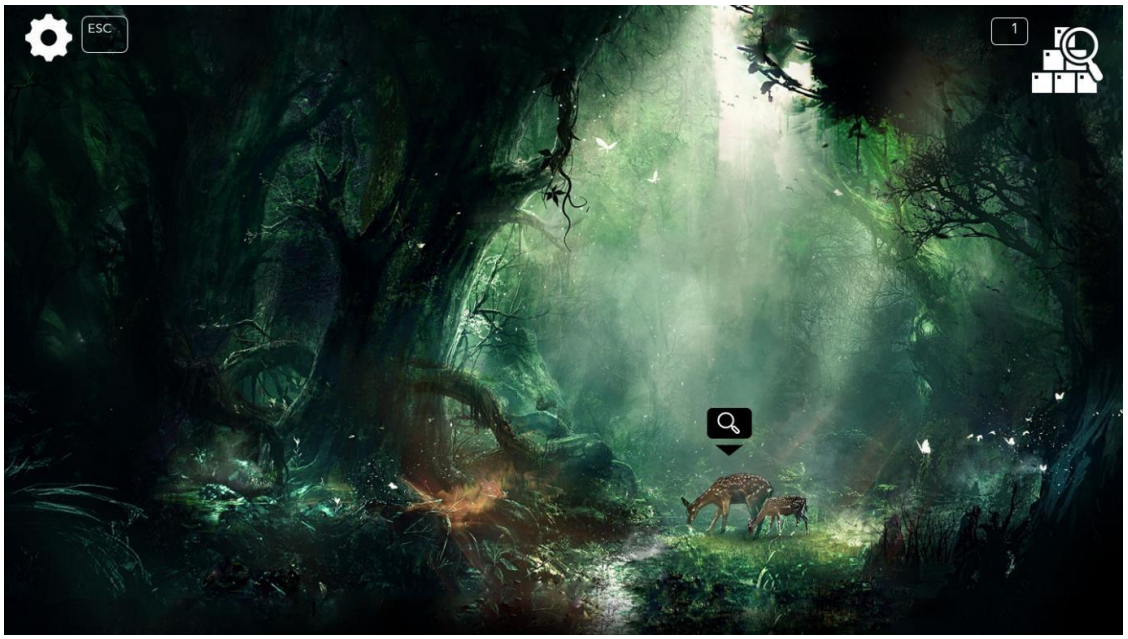


Figure 32 - Prompt

When the user selects the inspect option ‘E’, it will bring the user to the inspect screen which will zoom in on the object and give the user the option to rotate it and add it to their inventory or click the ‘EXIT’ button. This inspects screen will have a blurred background blurred.

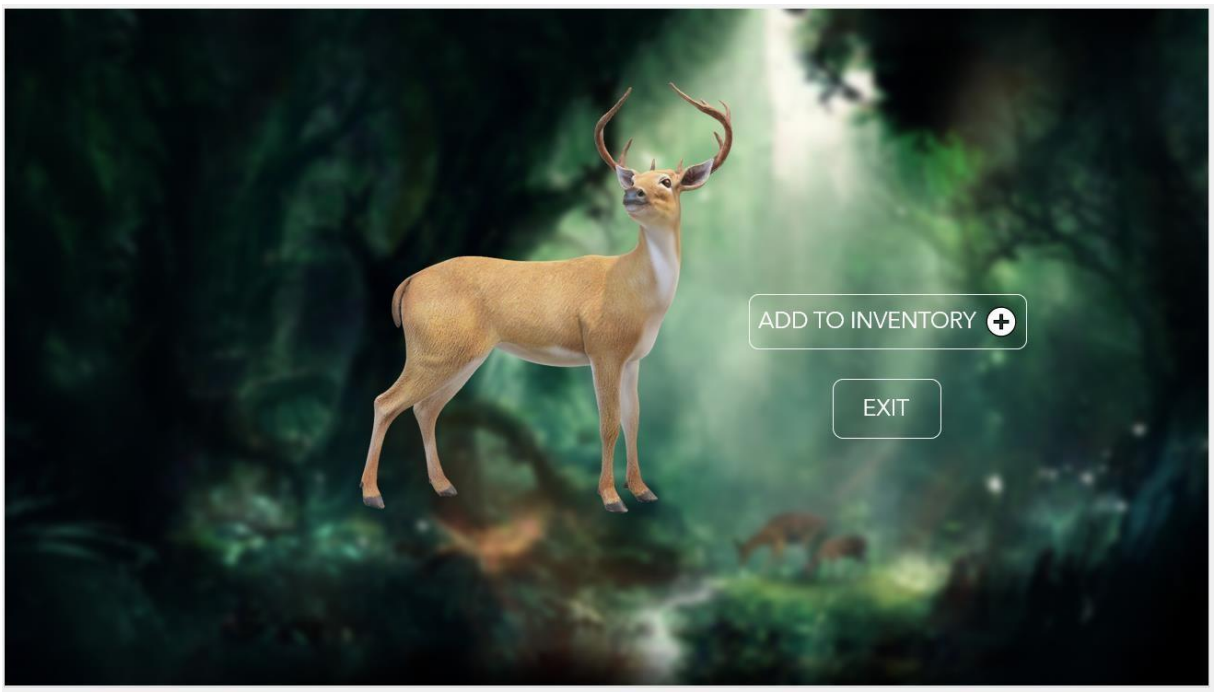


Figure 33 - Inspect Screen

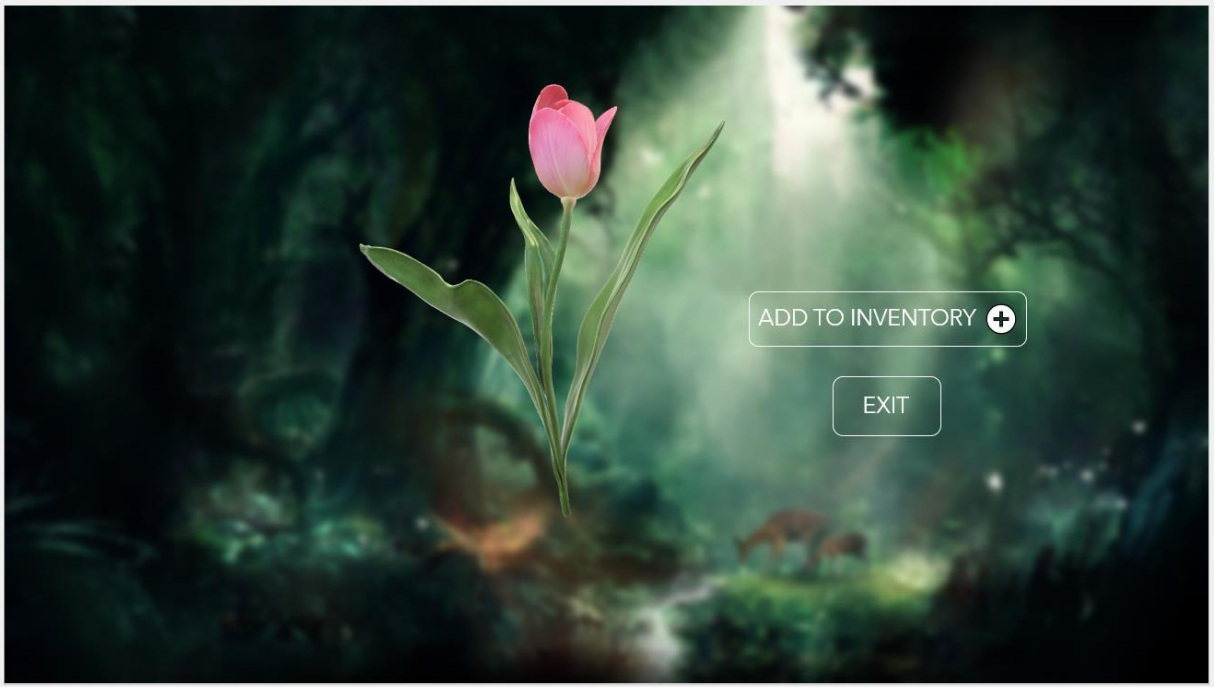


Figure 34 - Inspect Screen 2

Once you have added the animal or plant to your inventory, the user will be able to view the animal whenever the user wants to by pressing number 1 on the keyboard to go into the inventory which will show the inventory screen with a grid of all the wildlife the user has collected, along with an 'EXIT' button.

Inventory Screen:

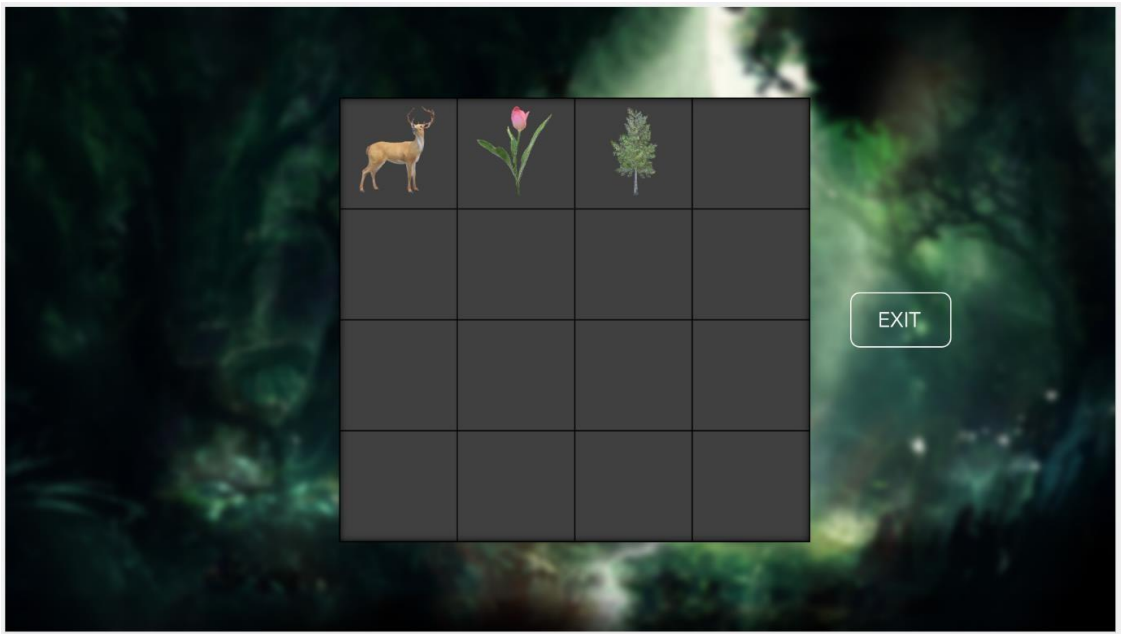


Figure 35 - Inventory

Once the user is on the inventory screen, they will have the option to click in the wildlife objects they added to the inventory and display information about the wildlife. This screen will have a back button and a quiz option button.

Information Screen:

Figure 36 - Information Screen

When the user selects start quiz the user will be able to do a quiz based on the information of the wildlife they are looking at. The questions will be multiple choice and a score of how many the user got correct will be displayed at the end. The user will then be given the choice to retake the quiz or exit the quiz.

Quiz Screen:

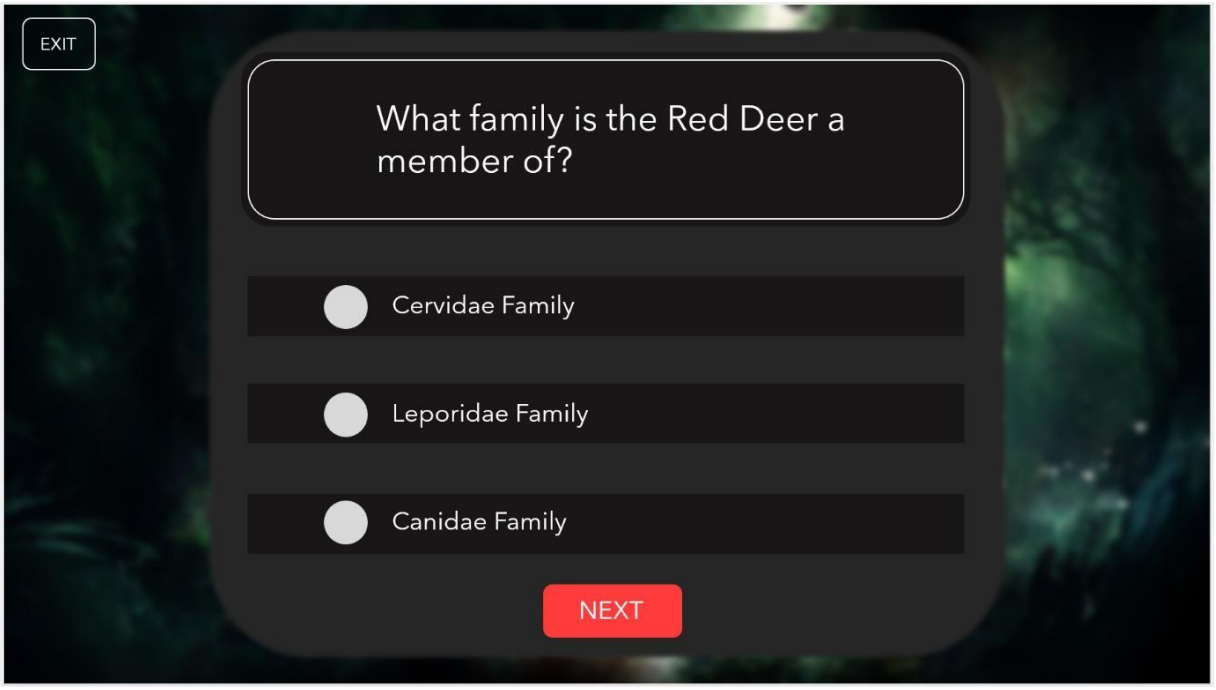


Figure 37 - Quiz

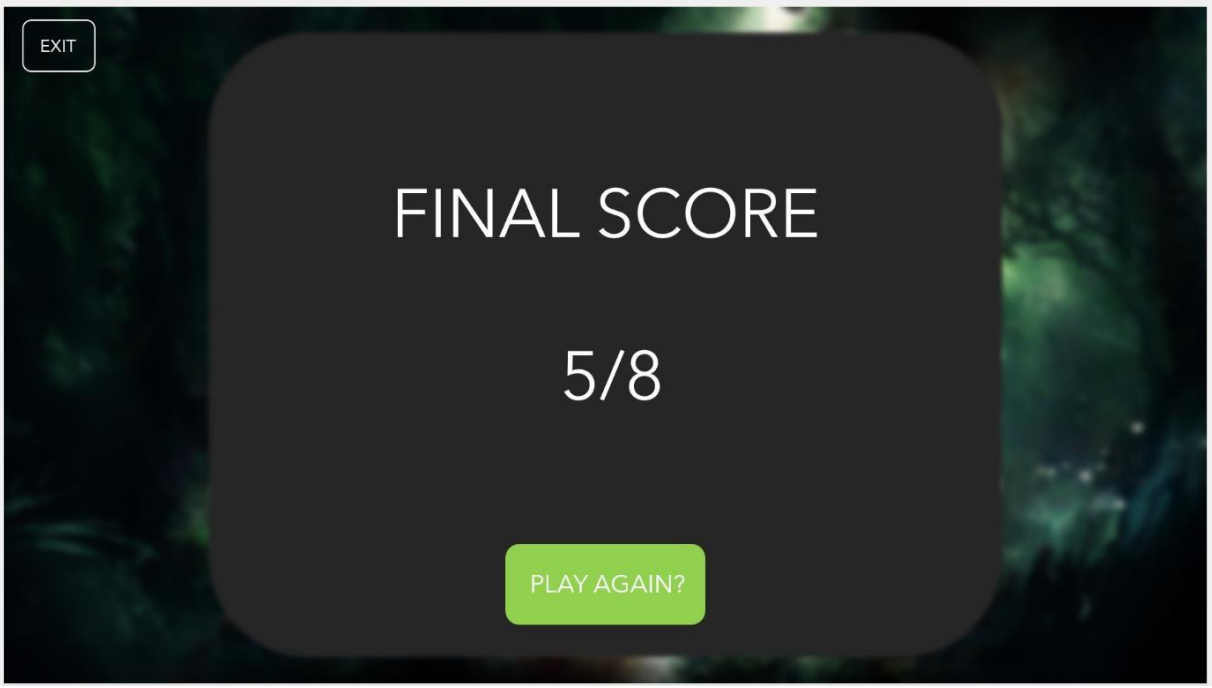


Figure 38 - Final Score

When the user is in the game, they can access the pause menu by pressing the escape button on the keyboard or clicking the settings icon in the top left-hand corner. The pause menu will have the following options, ‘RESUME’, ‘SAVE GAME’, ‘OPTIONS’ and ‘EXIT TO MAIN MENU’.

Pause Screen:

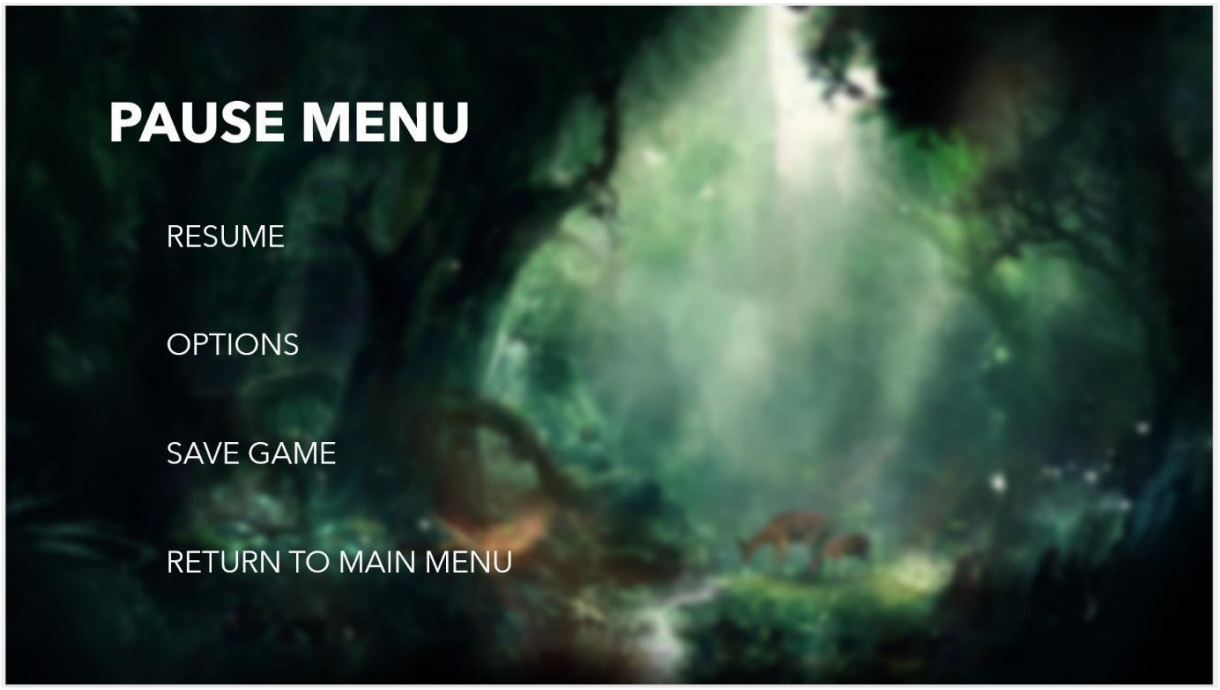


Figure 39 - Pause Menu

When the user selects the options button, they will be brought to a screen that follows the same format as the main menu’s options screen, where you will be able to adjust the volume and view the controls.

The ‘RESUME’ button will bring you back to the game and the ‘RETURN TO MAIN MENU’ button will bring you back to the main menu.

Options Screen:

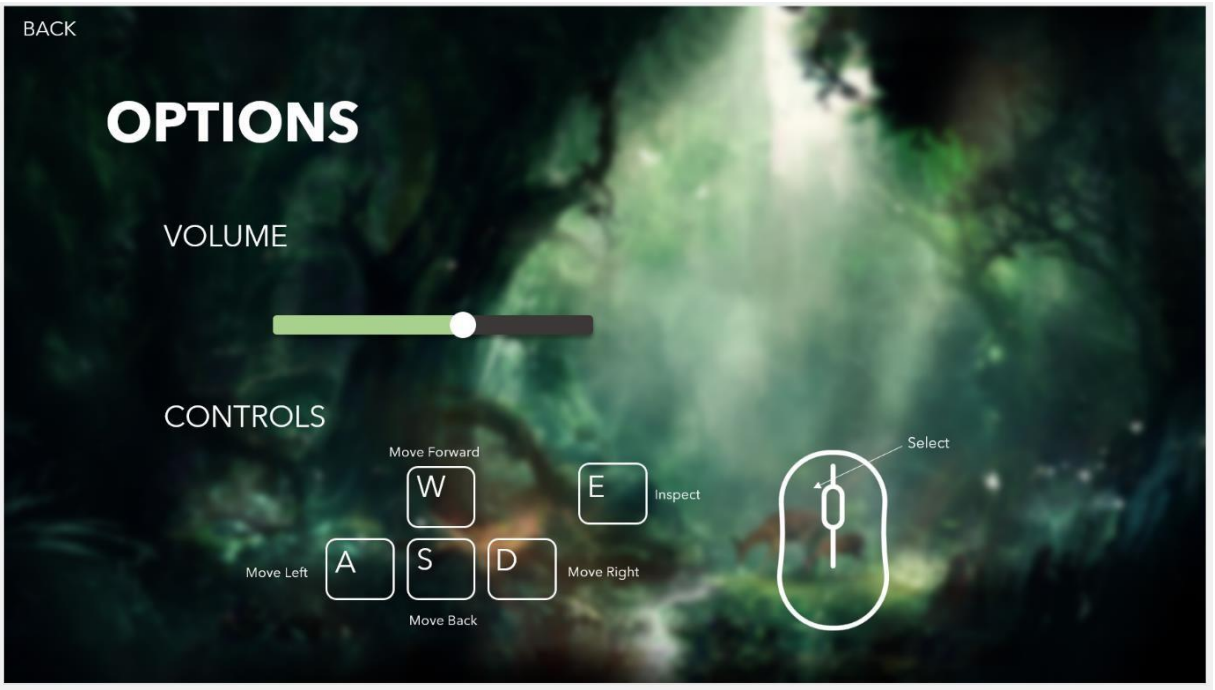


Figure 40 - Options

If the user selects the save game option, the user will be able to save their progress and placement in the game.

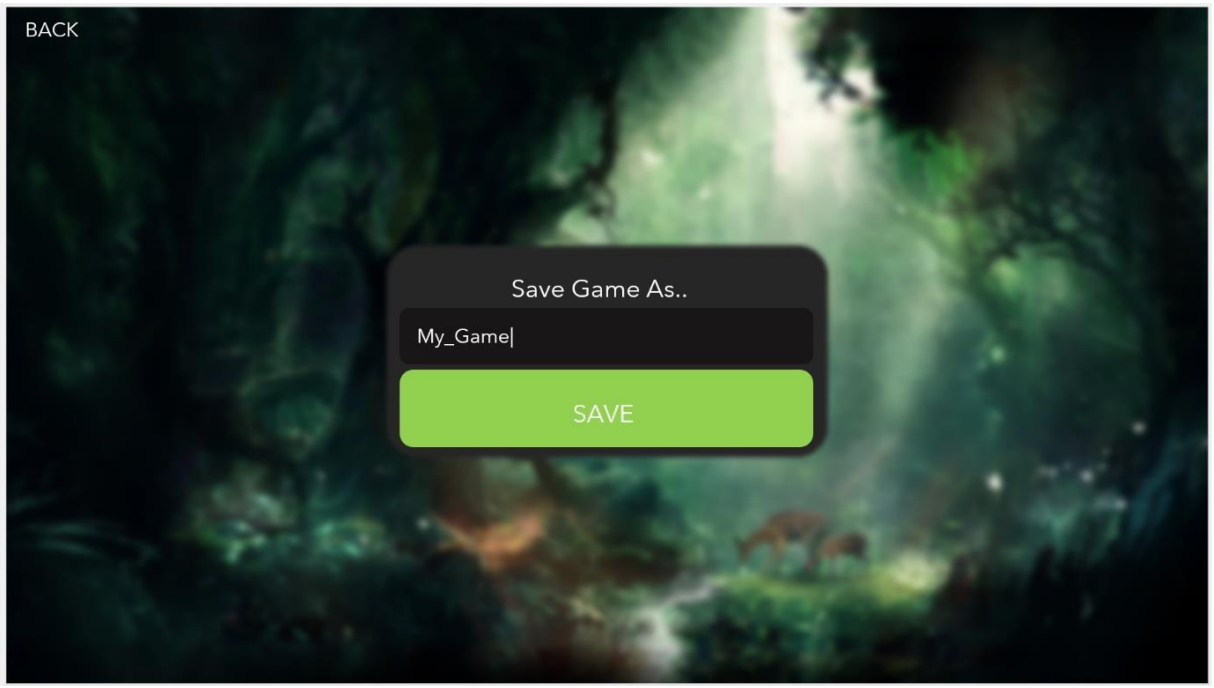


Figure 41 - Save Screen

Now, next time the user plays the game they will be able to select the saved game they want to load.

Load Screen:

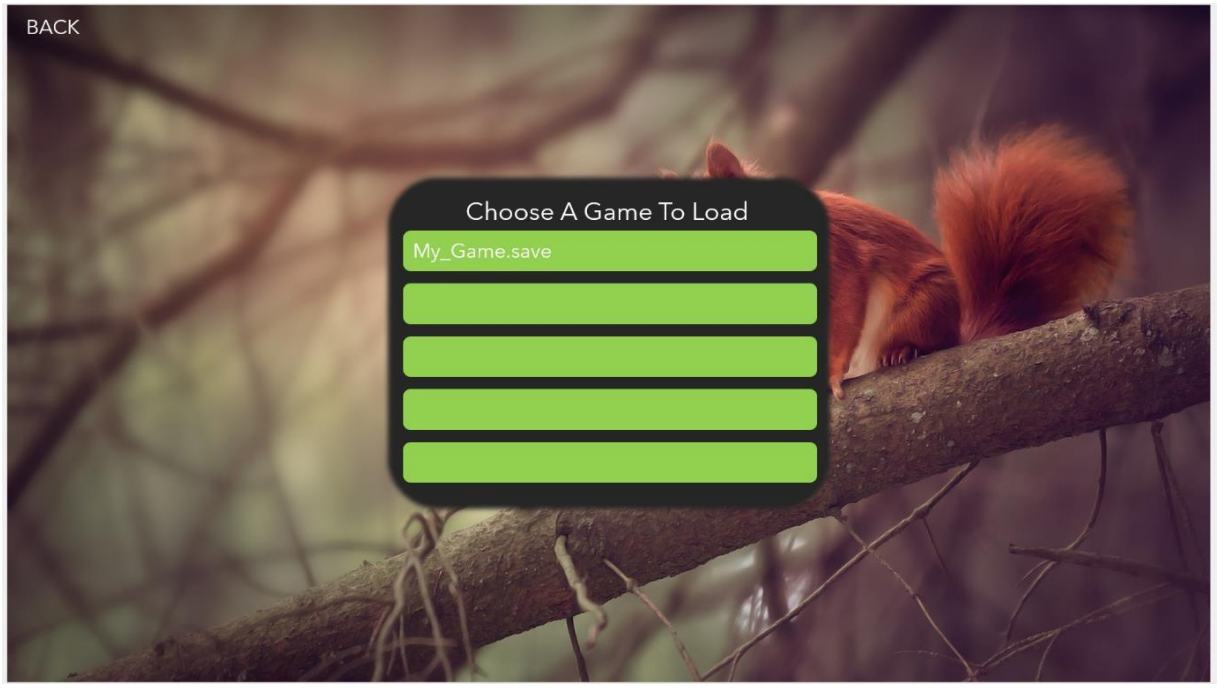


Figure 42 - Load Screen

3.5.1. Use Cases

There are many use cases for this project. Shown below are the use cases for each main feature of the game, the first three iterations of the use case diagram will be shown that explore all the main features briefly. Each main feature of the use case will then be described in detail using use case scenarios. These scenarios will have descriptions, a goal, a main flow with steps and an error flow each.

Use Case Diagram 1st iteration

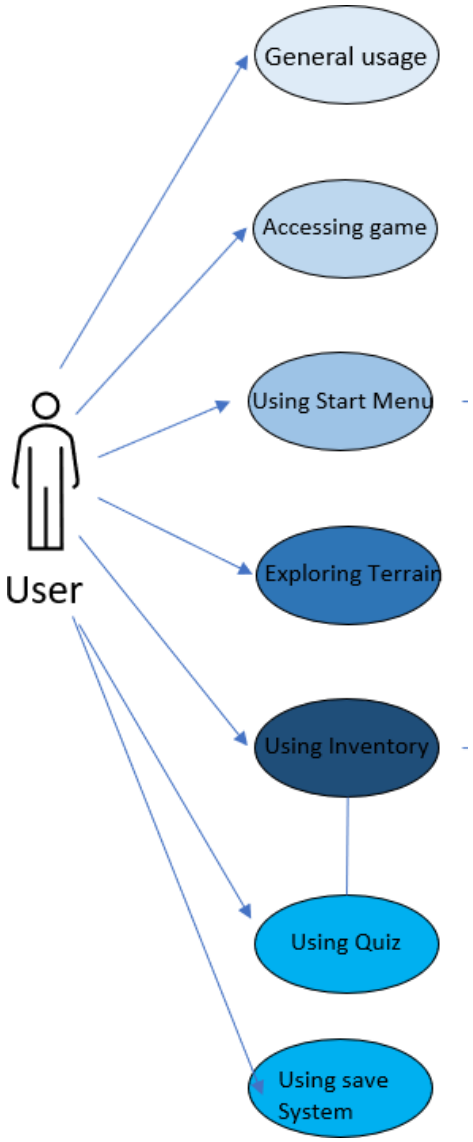


Figure 43 - Use Case 1st Iteration

Use Case Diagram 2nd iteration

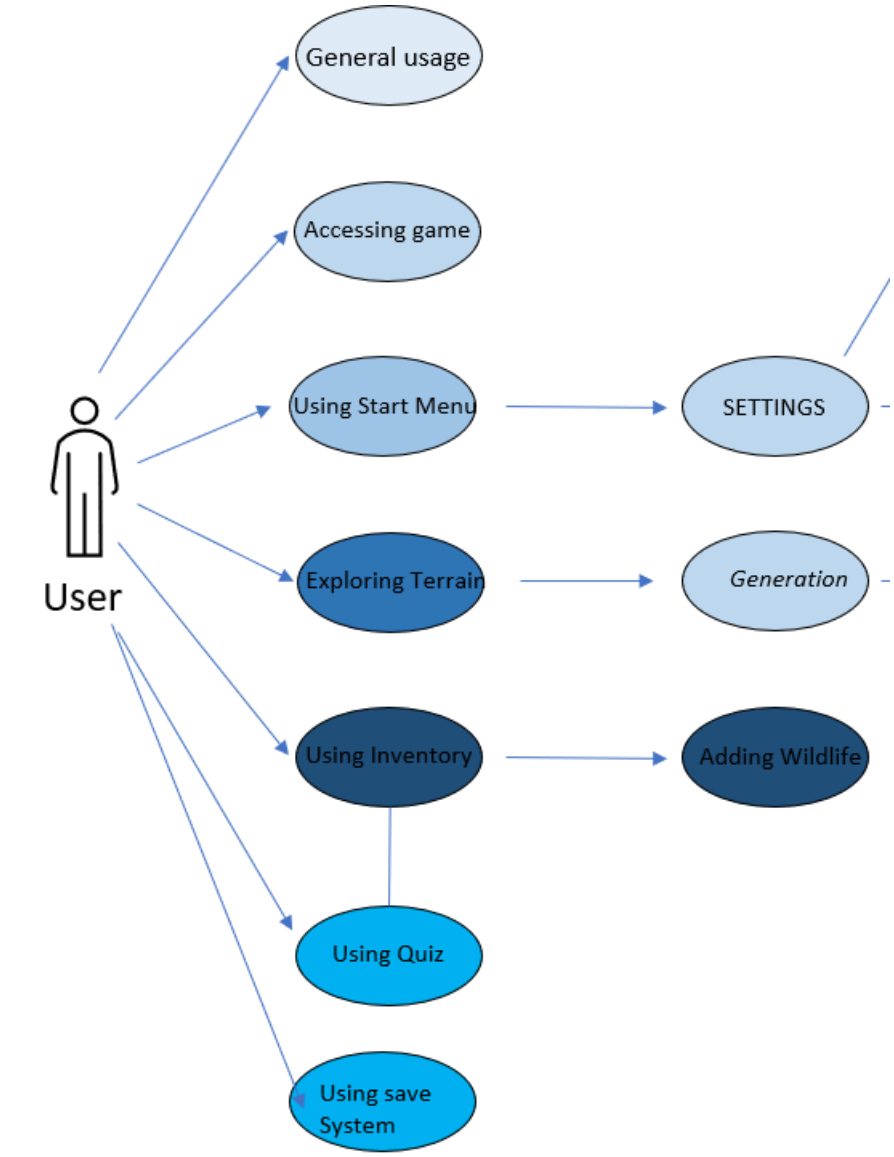


Figure 44 - Use Case 2nd iteration

Use Case Diagram 3rd iteration

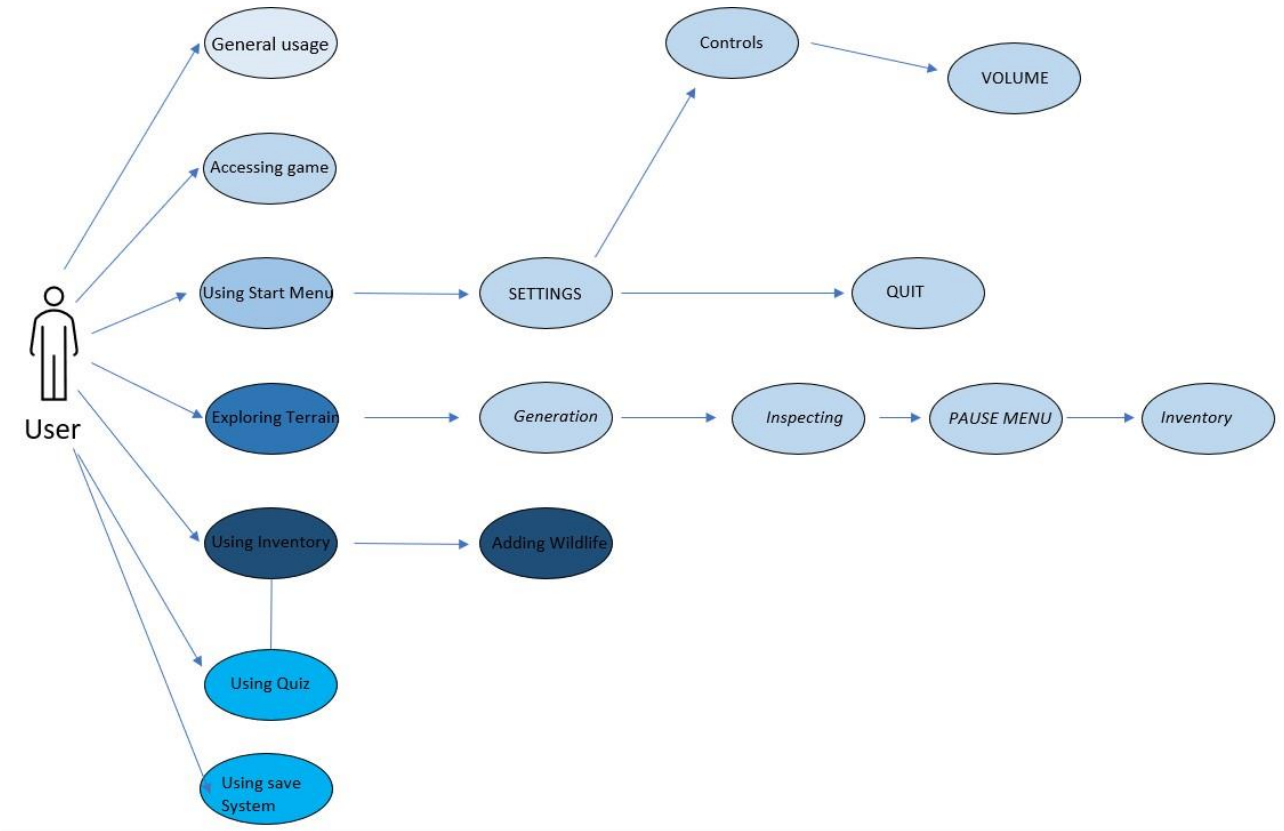


Figure 45 - Use Case 3rd iteration

Use Case Scenarios

USE CASE	**1**	**General Usage of Wild-Ireland**
Description of Goal in Context	**Navigating through application. Briefly looking at each feature. Use game as if user for first time to get all requirements.**	
Post Conditions, Success End Condition	**System will work well. Each feature works successfully with no errors.**	

Main Flow:		
Step	**Action**	**Alternate**
1.1	**Download Game**	**E or EF**
1.2	**Launch Game**	
1.3	**View Loading Screen**	
1.4	**View Start menu**	
1.5	**Select Settings**	
1.6	**Adjust Volume**	
1.7	**Look at Controls**	
1.8	**Select back**	
1.9	**Select Play**	**E or EF**
1.10	**View Cutscene**	
1.11	**Walk Around Terrain**	
1.12	**Inspect Animal or Plant (wildlife)**	**E or EF**
1.13	**View inventory**	
1.14	**View Video**	
1.15	**Start Quiz**	**E or EF**
1.16	**Add Animal to inventory**	
1.17	**Exit Notebook**	
1.18	**View Pause Menu**	
1.19	**Select Leave Game**	
1.20	**Select Quit in Start menu**	
EXCEPTIONS or ERROR Flow Description		
Step	**Branching Action**	
1.1.1	**Download Failed due to users' memory full.** **User Clears memory and retries download.**	
1.9.1	**Game not loading in. Cutscene having trouble playing**	

	Error message shown, user exits game and restarts.	
1.12.1	**Inspect prompt not showing.** **User moves closer or tries another animal.**	
1.15.1	**Quiz not showing questions** **Error message displayed and user restarts quiz.**	

USE CASE	**2**	**Accessing Game**
Description of Goal in Context	**User will be able to download game, Install the game on their computer. Launch the game.**	
Post Conditions, Success End Condition	**Download goes smoothly. Each feature works successfully with no errors.**	
Main Flow:		

Step	**Action**	**Alternate**
2.1	**Go to download link**	**E or EF**
2.2	**Click download link**	
2.3	**Download Game**	**E or EF**
2.4	**Install game**	
2.5	**Click icon**	
2.6	**Launch Game**	
2.7	**Load into game**	**E or EF**
2.8	**View Start Menu**	

EXCEPTIONS or ERROR Flow Description

Step	**Branching Action**	
2.1.1	**Users PC does not meet game requirements**	

	Access on different computer.	
2.3.1	**Memory Full.** **Delete data and retry download.**	
2.7.1	**Error loading into game.** **Restart game and try loading in again.**	

USE CASE	**3**	**Using Start Menu**
Description of Goal in Context	**User will be able navigate through menu features. All features will work successfully.**	
Post Conditions, Success End Condition	**Navigation goes smoothly. Successful with no errors.**	

Main Flow:

Step	**Action**	**Alternate**
3.1	**Launch Game**	
3.2	**View start menu**	
3.3	**Click options**	
3.4	**Click controls**	
3.5	**View controls**	**E or EF**
3.6	**Press back**	
3.7	**Click volume**	**E or EF**
3.8	**Adjust volume**	
3.9	**Press back**	
3.10	**Press start game**	
3.11	**Press quit game**	

EXCEPTIONS or ERROR Flow Description

Step	Branching Action	
3.5.1	Controls don't match Change controls.	
3.7.1	Volume on wrong settings	

USE CASE	4	Exploring Terrain/Environment
Description of Goal in Context	User will be able navigate/move through environment. All features will work successfully.	
Post Conditions, Success End Condition	Exploration goes smoothly. Successful with no errors. Inspecting, generation, notebook, Pause Menu.	

Main Flow:

Step	Action	Alternate
4.1	Press start game	
4.2	Load into environment	E or EF
4.3	Move throughout terrain	
4.4	Find wildlife	
4.5	Click prompt	
4.6	Inspect animal/plant	
4.7	View notebook	
4.8	View information	
4.9	Press back	
4.10	Explore more	
4.11	Press saves	E or EF
4.12	Inspect more objects	
4.13	Interact with animal AI	

4.14	**Press Pause Menu**	
4.15	**Select exit to main menu**	
EXCEPTIONS or ERROR Flow Description		
Step	**Branching Action**	
4.2.1	**Environment Fails to generate** **Make procedural scripts more efficient**	
4.11.1	**Game fails to save.** **Error message Prompts.**	

USE CASE	**5**	**Using Inventory**	
Description of Goal in Context	**User will be able to access information on the inventory. All features will work successfully.**		
Post Conditions, Success End Condition	**inventory features are displayed and functional. Successful with no errors. Videos, paragraphs, pictures, Quiz, exit, add feature.**		
Main Flow:			
Step	**Action**		**Alternate**
5.1	**Press button for inventory**		
5.2	**View info on wildlife**		
5.3	**Read text**		
5.4	**View Pictures**		
5.5	**Watch videos**		**E or EF**
5.6	**Add species to inventory**		
5.7	**Succesfully added prompt**		**E or EF**
5.8	**Click quiz**		
5.9	**Partake in quiz**		
5.10	**Press done**		

5.11	**Press exit notebook**	
5.12	**Continue exploring**	
EXCEPTIONS or ERROR Flow Description		
Step	**Branching Action**	
5.5.1	**Videos wont load.** **Error message displayed.**	
5.7.1	**Failed to add to database.** **Animal not added.** **Displays error prompt to retry.**	

USE CASE	**6**	**Using Quiz**	
Description of Goal in Context	**User will be able to access quiz in the notebook and test themselves on the knowledge they have learnt. All features will work successfully.**		
Post Conditions, Success End Condition	**Notebook features are displayed and quiz is functional with no errors. Successful with no errors. Answer questions, display score, re-take quiz.**		
Main Flow:			
Step	**Action**		**Alternate**
6.1	**Press button for Notebook**		
6.2	**Press quiz button**		
6.3	**Accessing quiz**		
6.4	**Start quiz**		
6.5	**Answer questions**		**E or EF**
6.6	**Tick boxes**		
6.7	**Click next question**		
6.8	**Submit quiz**		**E or EF**
6.9	**Prompt message**		

6.10	**Press done**	
6.11	**View score**	
6.12	**Re-take quiz**	
6.13	**Click done**	
6.14	**Exit quiz**	
6.15	**Return to game.**	
EXCEPTIONS or ERROR Flow Description		
Step	**Branching Action**	
6.5.1	**Questions not appearing** **Restart quiz,**	
6.8.1	**Quiz not submitting.** **Results not appearing.** **Error message shown.** **Retry.**	

USE CASE	**7**	**Using Save System**	
Description of Goal in Context	**User will be able to successfully add animals to notebook database and save the game with working database using Easy-Save.**		
Post Conditions, Success End Condition	**Everything will be added to the database Easy-Save successfully with no errors.**		
Main Flow:			
Step	**Action**		**Alternate**
7.1	**Launch Game**		
7.2	**Click start game**		

7.3	**Load terrain**	
7.4	**Add animal to inventory**	

7.5	**Press pause menu**	
7.6	**Click save game**	**E or EF**
7.7	**Exit game**	
7.8	**Click start game**	
7.9	**Select load game**	
7.10	**Load saved game**	**E or EF**
EXCEPTIONS or ERROR Flow Description		
Step	**Branching Action**	
7.5.1	**Save game name already used** **Prompt message to choose another name**	
7.8.1	**Failed to load game** **Prompt user to try again.**	

3.6. System Design

In this section, the system design is outlined. The art style, the models, the animation, the animal AI behaviours, the audio, and the map/level designs are created.

3.6.1. Inspection system

As discussed in the research section, there is an inspection system in this game. This means that when the user or player is roaming around the environment upon coming across an animal or some trees or plants, they will receive a prompt as shown in the 2nd iteration prototype that will enable them to inspect the object. When they have inspected the object, they will be able to view the wildlife and press a button to be added to their inventory.

This system will be created by using scripts in C# to use methods like Ray casting as discussed in the research section to allow the user to inspect the object and to be given a

prompt to inspect it. Classes will be created to handle this system and it will be linked to the inventory system. This is because the user will need to be able to add the inspected item to the inventory.

Figure 46 - Inspection System

3.6.2. Inventory system

As discussed in the research section, there will be an inventory system in this game. This system will allow the user to add animals or wildlife like plants or trees to their inventory. once these animals are in the inventory, they will be able to view the wildlife and look at information about the specific wildlife they have selected.

This will all be created by writing scripts in C# to create the inventory item data for the objects and build the system to handle the items. Different classes will be created to handle the items that are being collected and display the user interfaces for the inventory system.

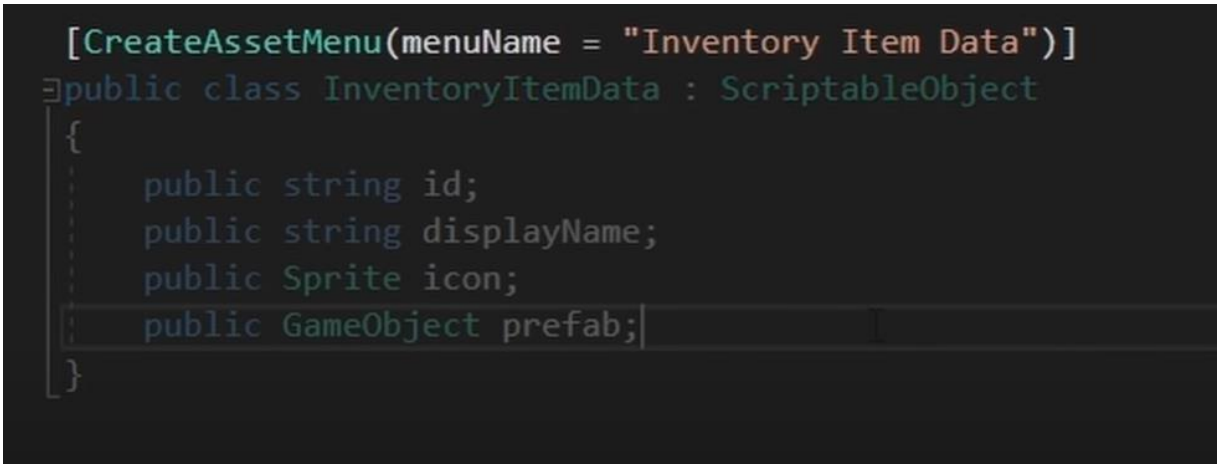


Figure 47 - Sample Inventory Code Snippet

3.6.3. Save system

As discussed in the previous section there will be a system in the game that allows the user to save and load their progress in the game. This system will enable the user to save their game progress from inside the game by using the pause menu and selecting the save game button. This will prompt he user to use a save name. when the game is saved the users wildlife, they have collected in their inventory will be saved. if the user exits the game and tries to choose the play game option again. They will be given a load game prompt where they can select which game save data they want to load before entering the environment again.

This will have three main components and classes. The first is the game classes which will be written in C# that will need data stored. Second is the save game class which is also written in C# that will hold the data. thirdly there will be the serialization system to handle the saving and loading of the Binary File as discussed in the research section.

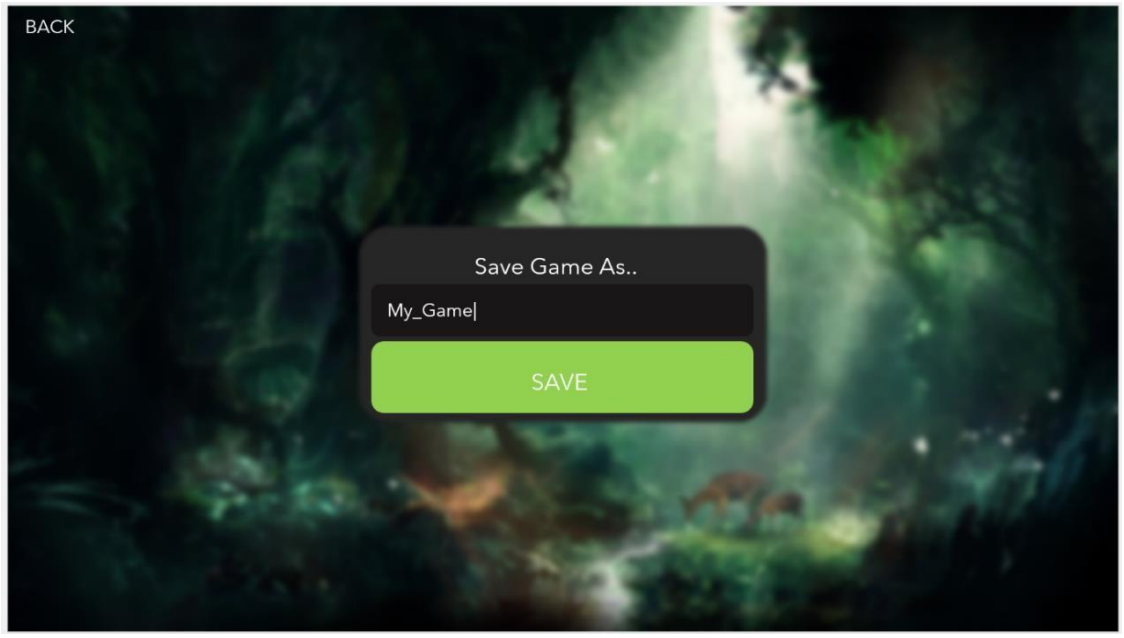


Figure 48 - Prototype Save System

3.6.4. Keyframe Animation

As discussed in the research section the animation for this project is done using keyframe animation. Meaning that the animations for the animals will be created in blender by recording movements frame by frame and turning them into animations.
Each animal will have their own walking and idle animation. Most of them will be quite similar apart from a few minor adjustments. For example, some animals will have different armatures (skeletons) which means that they'll have unique movements. Walking and running cycles will have to be looked at for each animal to get the correct movements.

This method of animation will be applied to every animal.

An issue or risk that might occur with keyframe animation is that it is very tricky to get the timing and the exact movements of the animals for each frame. This is an issue as it makes it difficult to record.

Rigging

When working with models and animations, for the models to be animated like the animals, they must be rigged. A rig is a skeleton that is attached inside the model / animal

so that the limbs can be moved around to create animations for the models. To achieve this, rigs will be created in Blender and attached to the animal models, which will then be animated. This will allow the animals to have animations such as walking running or jumping in Unity. Below is an example of a rig created in Blender that was attached to a model of a fox that was created.

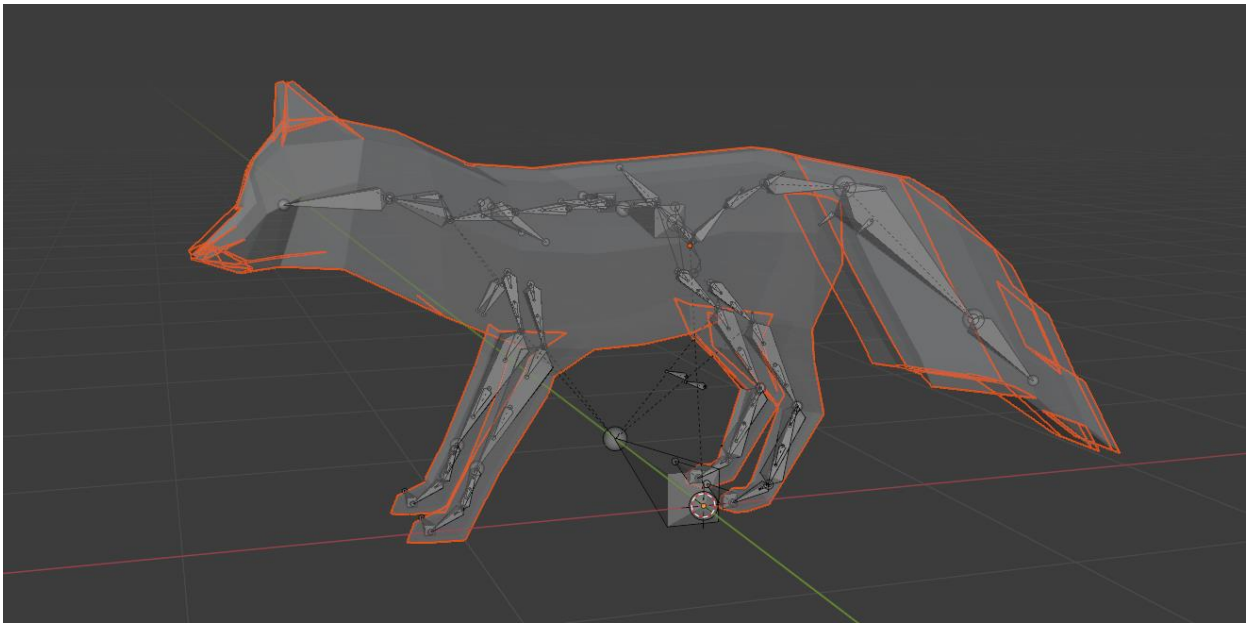


Figure 49 - Armature for Rigging Example

3.6.5. Player Movement

As this game is a first-person game, it needs a player to be able to move, look, and navigate throughout the terrain. To do this a player object will be created, then a camera will be attached to the player so the user can view from that players point of view. The player movement will have to be specified using a script written in C#. this will allow the user to press keys and move their mouse to move the player and look around the environment.

This script will reference Unity's input system which will allow different keys to be set as certain controls. The controls for now are as follows: 'W', 'A', 'S', 'D', and 'SPACE'. These are used to move the player forward, back, side to side, and jump. The mouse will be used to look around.

3.6.6. Quiz Design

When designing the quiz, the five different tips were followed as discussed in the research section. The quiz will be multiple choice and be used to test the user on the knowledge they have learnt from the game.

Only a maximum of four different answers will be able to be chosen from when answering a single question. This is following the 'don't list too many answers' technique. This will allow the user to be more likely to remember the right answer, it's also good for making the quiz more efficient.

The questions used will be very straight forward, there will be no trick questions, or riddles. This refers to the technique of "avoiding trick question's". This means that the student is less likely to get confused. This is good for the quality of the quiz.

The format of the questions will be quite simple, this allows the user to eliminate possible answers which allows them to have a higher chance of getting the answer right.

The test will be made challenging but not too difficult. Researchers have found that students may also commit bad information to memory by selecting the wrong answer on a difficult test. If the student is getting a large proportion of the questions wrong, this will hinder their ability to learn.

Finally, reviewing the quiz will be a very important aspect. Providing feedback will give the student a chance to re do the quiz if they are not happy with it. This will be done by keeping a score of their quiz and displaying it at the end. The user will then be given an option to retake the quiz.

Figure 50 - Quiz Design

3.6.7. Map / Level Design

The map that the users can explore is a forest terrain. Each time the user will start the game a new random map will be generated. Mountains will also be generated in the background. The method in which this will be accomplished is by using a form of procedural generation called **Perlin noise**. This will mean that the forest and terrain will be pseudo randomly generated each time the user plays the game. The forest and wildlife in the game will be in different positions and look slightly different every time. This terrain will act as the place where the user can explore and look for wildlife as a player. This will be done by Writing scripts in C# to allow the objects and forest to be generated.

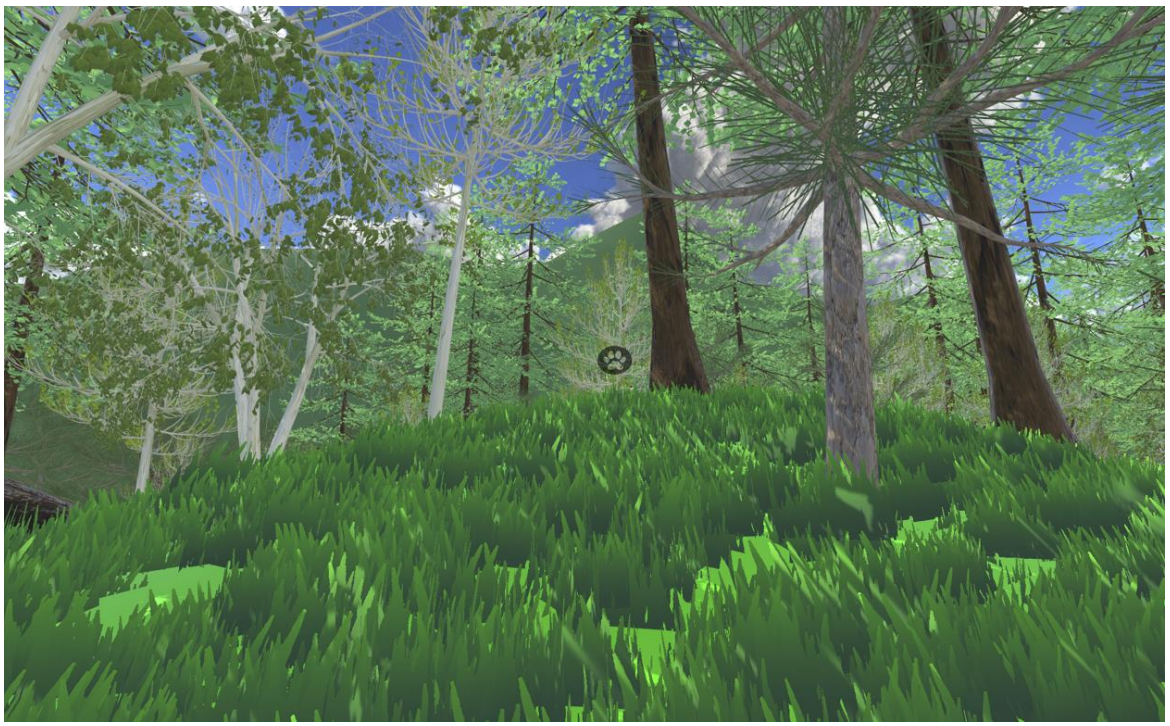


Figure 51 - Perlin noise Terrain

3.6.8. Animal AI

In this system the animals that the user will be able to inspect throughout the terrains will need to use AI to allow them to behave and move throughout the environment. This will be done by using the NavMesh feature in unity as decided in the previous section.
The animals will be able to roam randomly around the terrain. The animals will also stop and rest whilst there are wandering around. These animals will act differently based on each type of animal and when the user gets close to an animal it will react in a certain way. The behaviours for the animals are in the table below. Each animal will have a behaviour script written in C# that allows them to roam around the terrain by wandering and resting using NavMesh.
Each animal will also have a behaviour script that allows them to either approach the

player when they are close to them or sometimes run away. Some animals will only run away from the player, and some will do both behaviours. Some animals will chase other animals, and some will stay still and only move when approached.

Animal Generation

All the animals are being procedurally generated. They are generated at random positions, with different behaviours assigned randomly. There is a random amount of each species generated each time the game starts. For example, when the game starts five foxes might be generated at different positions and they each have different behaviour scripts attached to them. The next time the user starts the game there might only be two of them.

Behaviours Table

Animal	**General Behavior**	**Behavior Around Player**
Red Deer	-Wandering -Resting	-Runs away from player
Red Fox	-Wandering -Resting	-Approaches player -Runs away from player -chases other animals
Red Squirrel	-Wandering -Resting	-idle and runs -Runs away from player
Badger	-Wandering -Resting	-Approaches player -Runs away from player -roams around
Common Frog	-Wandering -Resting	-Runs away from player

3.6.9. Art

A low poly art style is used in this game. This will save time when creating models. However, if there is enough time, a higher quality more realistic style of modelling and art can be created and used throughout the game. The colours used will match the colours of the forests and wildlife in Ireland. (21)

3.6.10. Models

In Ireland there is lots of wildlife. In this project only a handful of animals, plant, trees, and

other objects will be chosen to be put in the landscape. This is because attempting to create too many models will be too time consuming. The chosen species will be discussed below and how they will be designed. (22)

Animals

This project is based on wildlife in Ireland so all animals and wildlife will be chosen that is native to Ireland. Below are the animals that will be in the game. The user will be able to inspect and add these animals to their inventory. There will be five different animals in the game, a list of the animals and a mind map was created to showcase the animals and what they look.

- Red Deer
- Red Fox
- Red Squirrel
- Irish Badger
- Common Frog

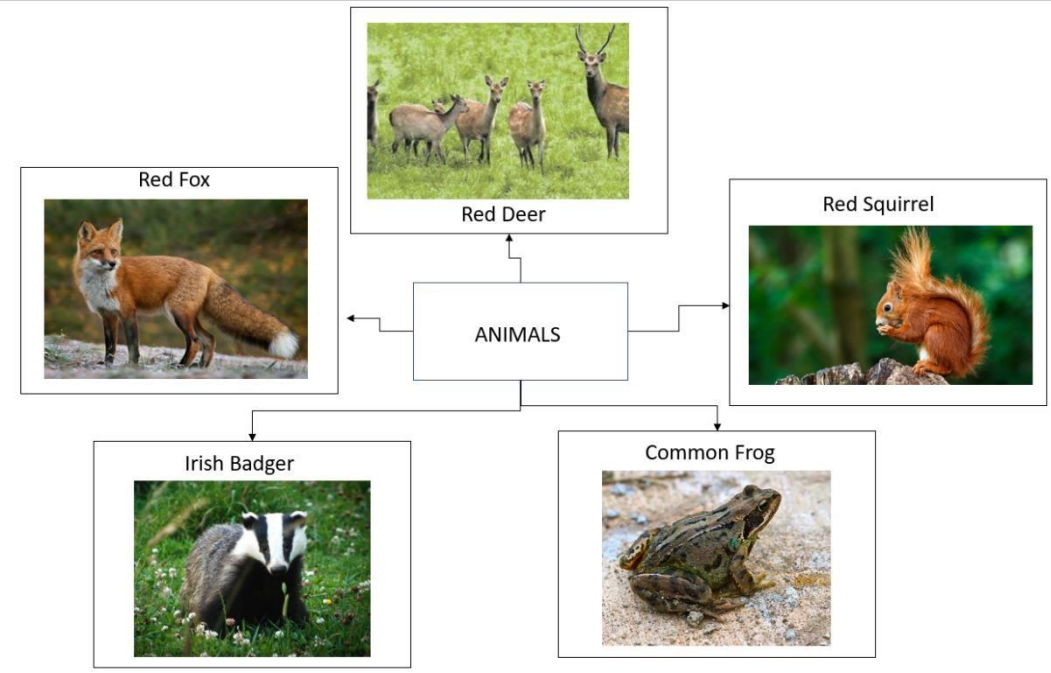


Figure 52 - Animals Mind Map

Plants

There will be seven different plant wildlife in the environment. Each plant will be able to

be inspected and added to the user's notebook inventory. They are as follows:

- Willow Tree,
- Maple
- Pine,
- Aspen
- Silver Birch.
- Douglas Fir
- Ferns
- Grass
- Bushes

Debris

The terrain will also contain various rocks, sticks and pebbles to make the terrain look more realistic.

- Two types of rocks
- Two pebble types
- One stick type

3.6.11. Audio

In this game, the audio will have an important part to play as it allows the user to be immersed in the game and listen to all the different sounds you would hear out in the wild in Ireland today. There will be sound effect for the animals, wildlife and a soundtrack for the game which will play in the background.
The table below shows a list of all the different audio sources that will be needed in the game for each Game Object. (23)

Audio Table

Game Object	**Audio Source**
Deer	Deer Noises
Fox	Fox Noises
Squirrel	Squirrel Noises
Badger	Badger Noise
Frog	Frog Noises
Terrain	Forest Noises Wind Bird noises.
Player	Footstep Noises

Main Menu	When buttons are clicked in the main menu there will be a clicking sound effect. Music when in menu.
Pause Menu	When buttons are clicked in the Pause menu there will be a clicking sound effect.
Cutscene	Narrator Audio and Music Audio
Inventory	When wildlife is added to the inventory audio will play.
Quiz	There will be quiz audio for button clicks and completion.
Loading screen	Music for loading screen

3.6.12. UI Design

When making a game, there are lots of UI elements involved. Most UI elements are created by creating a canvas object and then adding buttons, images, text, and backgrounds to the canvas. For some UI features, C# scripts are also needed. For this game, it was decided that UI elements such as various menus and loading screens were needed alongside a mini map and various HUD elements.

For the menu design, there will is main menu and an options menu which display various buttons and icons for the user to navigate with. A pause menu will also be designed to be accessed from in game, which is a necessary UI element.

Loading screens which act as progress bars to show the user how long it will take to load an aspect of the game and to show the game is loading will be implemented as UI elements for various screens.

Finally, a mini map is used in this game. A mini map shows the location and gives a bird's eye view of the player. This allows the user to always see the player from above. Different UI elements such as button prompts, and a crosshair will also be displayed on the players HUD. A HUD is basically a term used to describe the 'Heads Up Display' on screen.

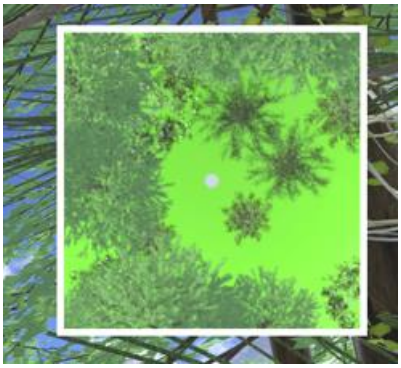


Figure 53 - Mini Map

3.7. Conclusions

In this section, we explore and discuss the different aspects of design. It first looks at version control and setup of the project followed by the design methodology where it discusses the use of GitHub and the chosen ADDIE Model. It then talks about the system architecture along with the system Requirements. After this, the early prototypes are shown by presenting them in picture form. Use case scenarios are thoroughly examined looking at the different outcomes for this game and finally the System Design in relation to models, art style and audio are explained.

4. Development

4.1. Introduction

In this section, the overall approach taken in the system development for this project is shown in detail alongside the finished application. The development will discuss many features such as the procedural generation, the UI, asset modelling, UI creation, the inspection system, the quiz feature, the models and animation, the audio, the cutscene, the menus, the mini map, and the AI. The player movements and components along with all the necessary game components such as how the game was made available to the user is also discussed. All development of usability features will be outlined. First, the initial experimental prototype will be discussed, followed by the entire system development.

4.2. Experimental Prototype

The first part of the development process was to create a new project in unity. A GitHub repository (30) was created as discussed previously, it was cloned into a folder and then the project was pushed to the repository using the command
'Git push origin main'. After this was completed the first prototype development was started.

The goal for the first prototype was to create the 'Start Menu' screen and the 'Options Menu' Screen in Unity. The layouts of the screens and the screens interacting with each other will be shown.

GitHub Link: https://github.com/jakebolger/Wild-Ireland-FYP.git

4.2.1. Experimental Prototype Development

The first screen that was created was the 'Start menu'. The first step was to create a new scene in unity. For this scene the background of the 'Start Menu' consisted of a forest background with trees, bushes, rocks, and a Frog. To do this, models needed to be created for the forest objects.

The first step to create these models was to create them using Blender. Overall, there were six trees created Birch, Aspen, Maple, Willow, Pine, and Douglas Fir. The first tree modelled in 3d was the Birch tree. To create this in Blender the 'Sapling Tree Generator' Add on was used. This allows you to create custom trees from scratch by adjusting the trunk, branch, and leaves of the tree to create the desired tree. Images from google were used as a reference to copy the appearance of the chosen trees. (24)

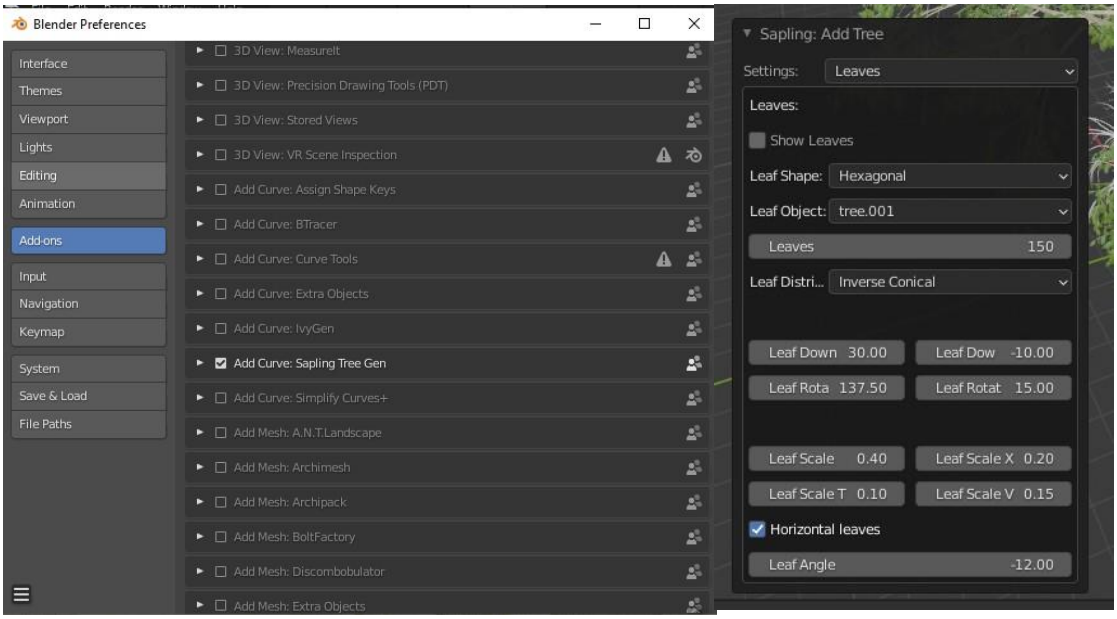


Figure 54 - Tree Gen Add on

Once the models were created in Blender they were exported as FBX files so that they could be imported into the Unity Engine. In Unity the tree models were imported as new assets and placed in the background of the 'Start Menu' Scene.

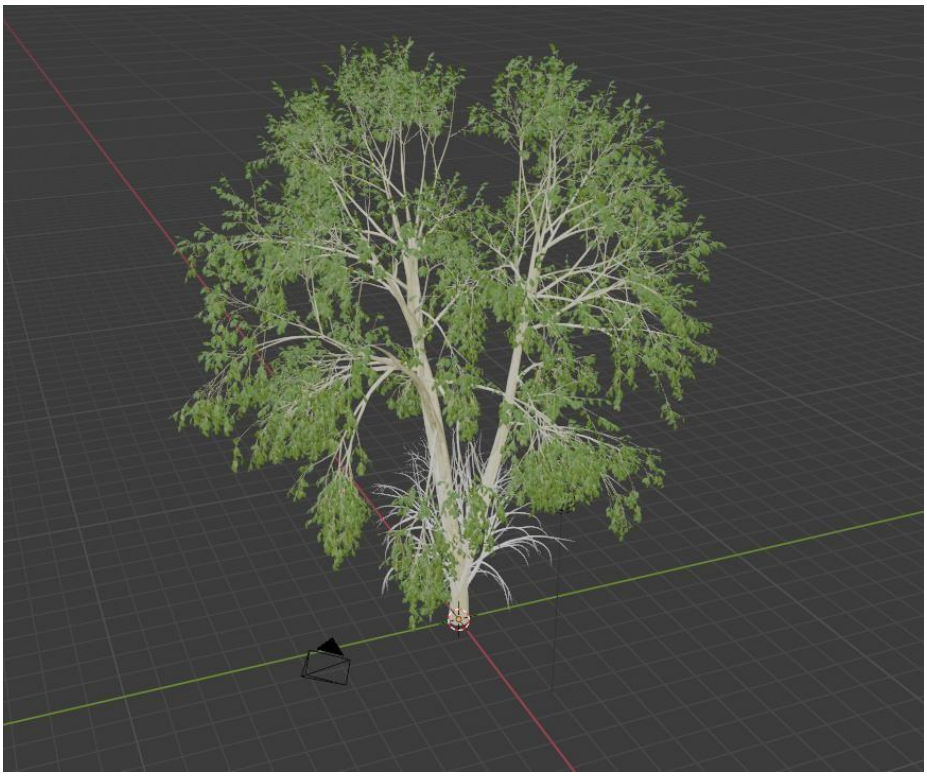


Figure 55 - Birch Tree Model

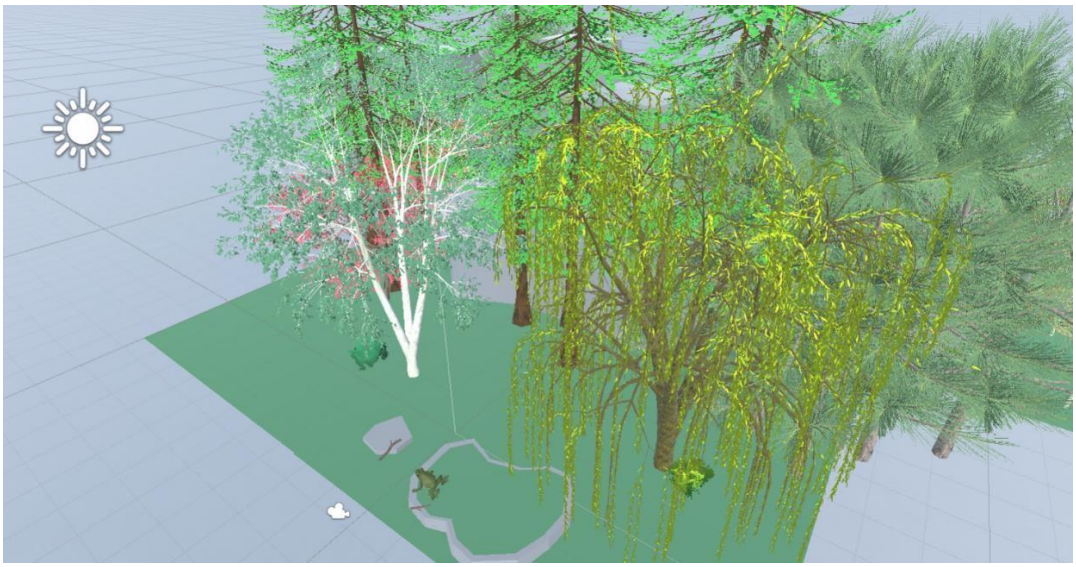


Figure 56 - Scene Placement

The next step was to create the menu UI. To do this a Canvas object was created in Unity which allows you to add text and Buttons to the screen. The title, the Play button, the options button, and the leave game button were added to the menu by creating button objects and text objects and adding them to the scene.

Figure 57 - UI Creation

The next screen that needed to be created was the ‘Options Menu’ the same method was used to create this menu as the ‘Start Menu’ except the buttons were different and had different functions. An options title was added, along with a volume title and controls title. Under the volume text a volume adjuster was created as an object in the hierarchy in Unity. This means the user will be able to adjust the volume.

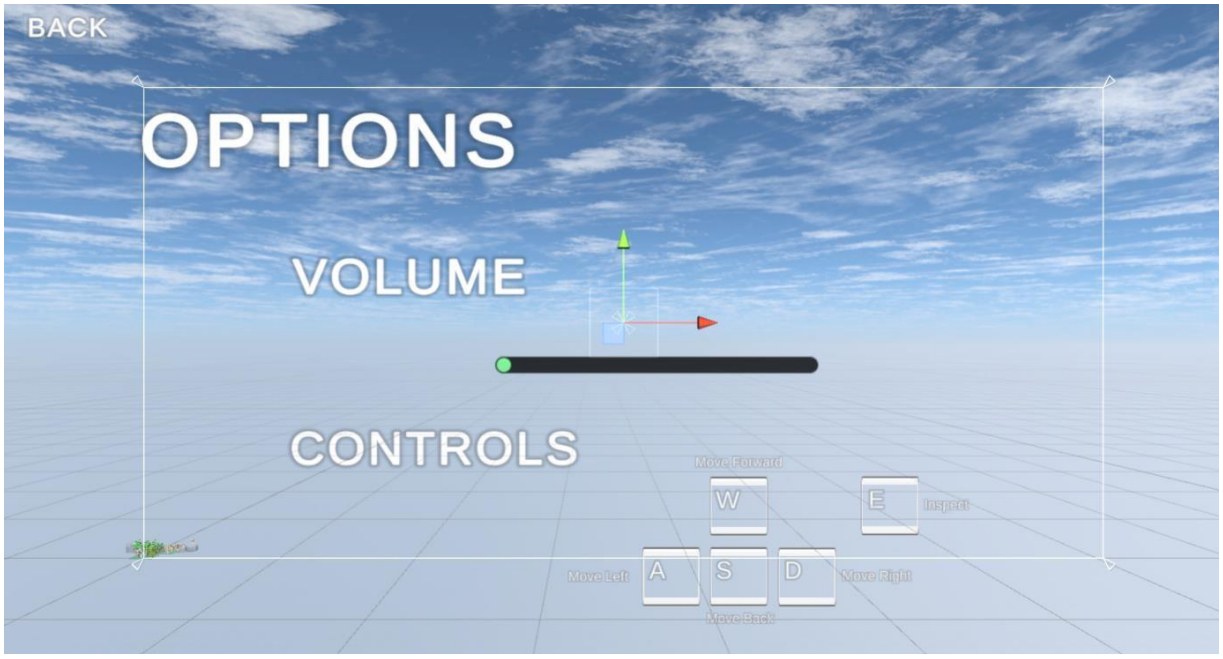


Figure 58 - UI 2

Pictures of the controls were added under the controls title.

Finally, a back button was added to the ‘Options Menu’ to return to the ‘Main menu’.

To allows these screens to interact with each other and switch between the main menu and the options screen a C# script for the Menu was added. This script was added to the Main Menu, and it allow the user to press the play button and load into a new scene. It also allowed the user to press the
‘OPTIONS’ button and switch to the ‘Options Menu’. In the script there were two functions one for the PLAY button and one for the QUIT button.

MainMenu.cs

Miscellaneous Files — MainMenu

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
6   public class MainMenu : MonoBehaviour
7   {
8       public void PlayGame()
9       {
10          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
11      }
12
13      public void QuitGame()
14      {
15          Debug.Log("QUIT!");
16          Application.Quit();
17      }
18  }
```

Figure 59 - Code for Screen Interaction

Next free assets from the assets store were downloaded to get models for the grass, the sky, and the frog model. The reason these assets were downloaded and not created manually was because this is just a first prototype, and the models were used to fill in some of the terrain that was missing. These assets will not be used in the final system and will be replaced by custom models created in Blender. These models were added to the background scene along with an audio source that plays forest sounds on the background of the scene. The initial prototype was then complete. The next stage was to start the actual system development, where the entirety of the project development was completed.

Figure 60 - Menu Screen

4.3 System Development

This section describes the whole of the system development. Most of this project was created in the Unity Game Engine and most of the models were created using the 3D modelling software Blender. Overall, this project has around 50 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. For these scripts to work, they need to be attached to game objects which are created in Unity. For example, if you have a game object that is acting as a player, it will need a script written in C# attached to it which sets and specifies the rules and code to make that player able to move and work as needed.

In this project many scenes were created. In Unity, a scene is something that contains the objects in your game. They are used to create things like individual levels, menu systems, and whatever you would like to include in your game. Most of the game for this project was created in one scene which was named the 'World-Scene', but other scenes for the different menus, and quizzes were also created.

4.3.1. 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 animals that roam around the map, the mountains in the background, the grass that's being created and the forest that is created are all generated through C# 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 of the world, below are the aspects which are most important.

Terrain Generation

First, the terrain that is generated in which the player uses to manoeuvre throughout the environment will be discussed. Usually, to create terrain in Unity, a 3D terrain object would be created using Unity's built-in terrain system which creates the terrain instantly. Whilst this method is simple and quick, it doesn't use any code and adds no complexity to the system. Therefore, procedural generation was chosen to generate each individual part of the terrain using C#.

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 objects and scripts. 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.

The Scripts are quite long so only some snippets will be shown. Below are all the variables that were set that were used to generate the meshes and a couple of the functions to generate the terrain.

Terrain Variables that were set and start() function to set them and call the functions:

```
    private float[,] heightMap;

    //integer that sets the size of the terrain and allows you to change it in the inspector
    //
    public int terrainSize;
    //float for the scale if the terrain
    //
    public float terrainScale;
    public float heightMultiplier;

    //reference for the forestgenerator script
    //
    public ForestGenerator Forest;
    public GrassGeneratorScript Grass;



    //meshfilter material and meshrenderer variable.
    //
    MeshFilter meshFilter;
    MeshRenderer meshRenderer;
    Material material;
    MeshCollider meshCollider;

    private void Start()
    {
        //setting the varibales
        //
        meshCollider = GetComponent<MeshCollider>();
        meshRenderer = gameObject.AddComponent<MeshRenderer>();
        meshFilter = gameObject.AddComponent<MeshFilter>();
        material = new Material(Shader.Find("Standard"));

        // Validation check to ensure terrainSize is bigger than forestSize, as this will create an indexOutOfBounds exception.
        //
        if (terrainSize < Forest.forestSize)
            terrainSize = Forest.forestSize;
        if (terrainSize < Grass.grassSize)
            terrainSize = Grass.grassSize;

        heightMap = new float[terrainSize, terrainSize];

        GenerateHeightMap();
        Generate();
        Forest.Generate(heightMap);
        Grass.Generate(heightMap);

    }
```

Figure 61 - Terrain Generation Variables Snippet

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

```
//Getheight function, takes in a vector3 and gives it a position in unity.
//
float GetHeight(Vector3 position)
{

    // Return a float representing the height of the terrain at the given position in world space.
    //
    return Mathf.PerlinNoise((position.x + 0.1f) / terrainSize * terrainScale, (position.z + 0.1f) / terrainSize * terrainScale) * heightMultiplier;

}

//Generate heightmap function
//
void GenerateHeightMap()
{

    // Loop through each position in our terrain, get the height at that position, and put it in our heightMap array.
    //
    for (int x = 0; x < terrainSize; x++)
    {
        for (int z = 0; z < terrainSize; z++)
        {

            heightMap[x, z] = GetHeight(new Vector3(x, 0f, z));

        }
    }

}
```

Figure 62 - Snippet of Using Perlin Noise

To create the terrain each vertex, triangle, and UV had to be created. for someone who doesn't know what a UV is, it's a process which allows a 2D image to be projected onto a 3D surface. The next snippet of code shows one of the generate functions used to do this. Generate() Function generating Vertices, triangles, and vectors:

```
void Generate()
{

    // Initialise arrays and indexes we need for creating our mesh.
    //
    Vector3[] vertices = new Vector3[terrainSize * terrainSize];
    int[] triangles = new int[(terrainSize - 1) * (terrainSize - 1) * 6];
    Vector2[] uvs = new Vector2[terrainSize * terrainSize];

    int vertIndex = 0;
    int triIndex = 0;

    // Loop through each position in terrain, create necessary vertexs, indices, and UV coordinates.
    //
    for (int x = 0; x < terrainSize; x++)
    {
        for (int z = 0; z < terrainSize; z++)
        {

            vertices[vertIndex] = new Vector3(x, heightMap[x, z], z);
            uvs[vertIndex] = new Vector2((float)(x / terrainSize), (float)(z / terrainSize));

            // Make sure we're not on the last row/column of the mesh as setting a triangle from there would create an indexOutOfBoundsException.
            //
            if (x < terrainSize - 1 && z < terrainSize - 1)
            {
                //adding triangles, represent a quad with 4 point and draw the square with two triangles
                //
                triangles[triIndex] = vertIndex;
                triangles[triIndex + 1] = vertIndex + terrainSize + 1; //+ terrain size gets us down to the next row
                triangles[triIndex + 2] = vertIndex + terrainSize;
                triangles[triIndex + 3] = vertIndex + terrainSize + 1;
                triangles[triIndex + 4] = vertIndex;
                triangles[triIndex + 5] = vertIndex + 1;

                //increment the triangle index
                //
                triIndex += 6;

            }

            //increment vetext index
            //
            vertIndex++;

        }
    }
```

Figure 63 - Generate() Function for Vertices

Once the Script was written 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.

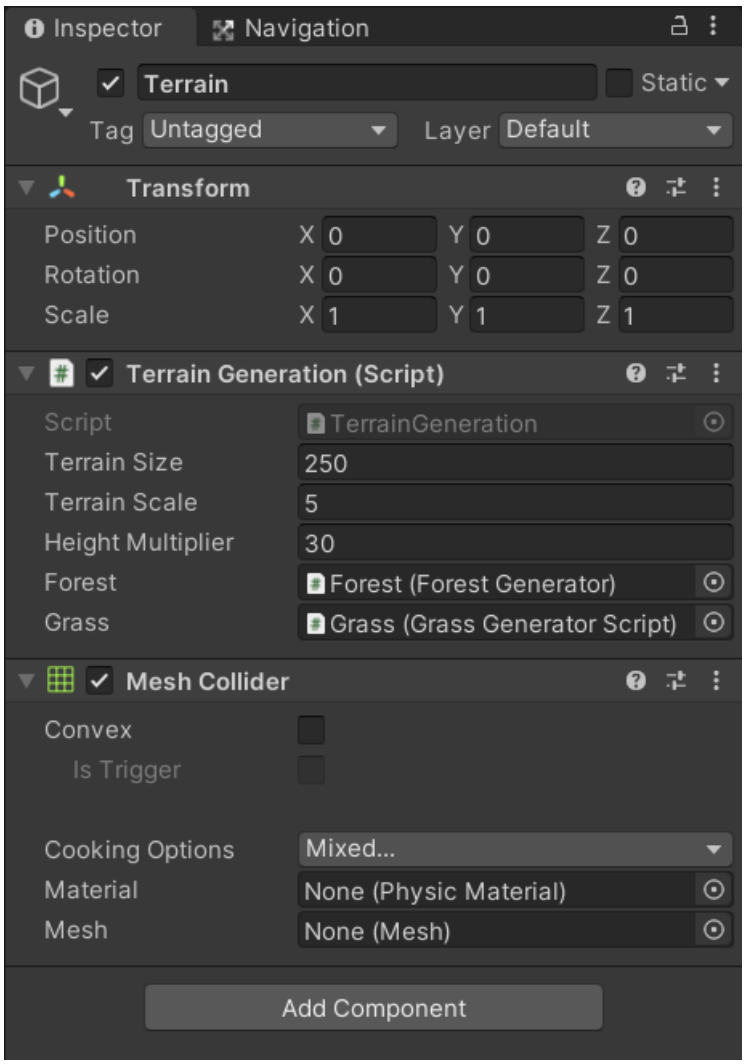


Figure 64 - Terrain Object Inspector

Once this process was complete, whenever the user starts the game, a terrain mesh is created procedurally as shown in the image below.

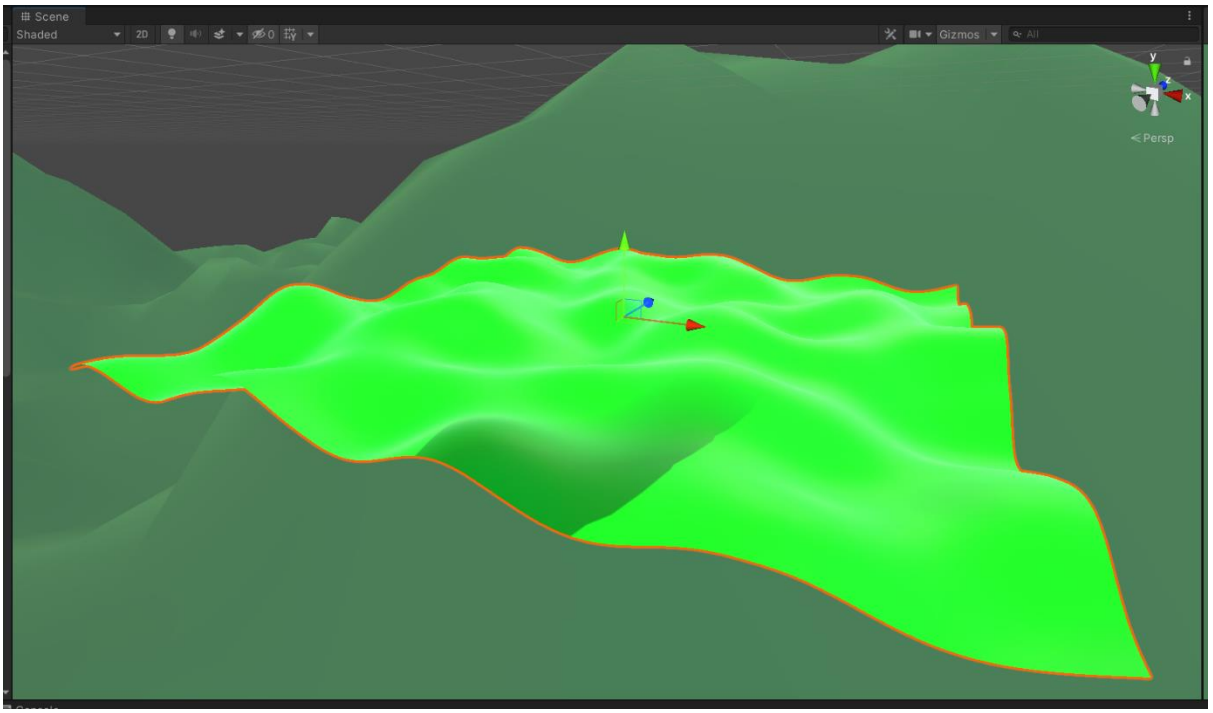


Figure 65 - Procedurally Generated Terrain

Forest generation

Next, the procedurally generated forest was created. This was a game object which used a script which allowed it to generate a forest using tree models from blender (discussed in model section) that is generated randomly with a unique forest every time the game is started. The forests positioning, scale, size, density, rotation, and offset are all randomized. This script uses the Generated terrain script to allow it to be placed on the terrain at the correct heights and the correct size to match the terrain. It does this by using the 2D heightmap array used in the terrain script.
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 generated 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 generator class and the Generate function in it, which allowed the tree models to be placed procedurally and look more realistic rather than a grid.
'Generate()' function which passes in 'heightMap' array:

```
public void Generate(float[,] heightMap)
{
    //looping from 0 to set forest size and incrementing by the element spacing
    //this means every 3 units in unity we will put i elements down
    //
    for (int x = 0; x < forestSize; x += elementSpacing)
    {
        for (int z = 0; z < forestSize; z += elementSpacing)
        {
            for (int i = 0; i < elements.Length; i++)
            {
                //Gets element, giving priority to trees
                //
                Element element = elements[i];

                //if statement with CanPlace Function
                //
                if (element.CanPlace())
                {
                    //Vector3 position
                    //passing terrain heightmap into height
                    //
                    Vector3 position = new Vector3(x, heightMap[x, z], z);

                    //new vector3 offset
                    //creating a vector offset thats a random amount to vary position
                    //
                    Vector3 offset = new Vector3(Random.Range(-0.75f, -0.75f), 0f, Random.Range(-0.75f, 0.75f));

                    //creating a vector for rotation giving it a random rotation / tilted
                    //
                    Vector3 rotation = new Vector3(Random.Range(0, 5f), Random.Range(0, 360f), Random.Range(0, 5f));

                    //creating a vector for scale and randomizing it
                    //
                    Vector3 scale = Vector3.one * Random.Range(0.75f, 1.25f);

                    //creating new game object and instantiating the prefab
                    //
                    GameObject newElement = Instantiate(element.GetRandom());

                    //setting the parent so the prefabs dont float around the hierarchy
                    //
                    newElement.transform.SetParent(transform);

                    //setting the elements position
                    //
                    newElement.transform.position = position + offset;
```

Figure 66 - Function Generating Trees at Random Positions

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

Forest object inspector:

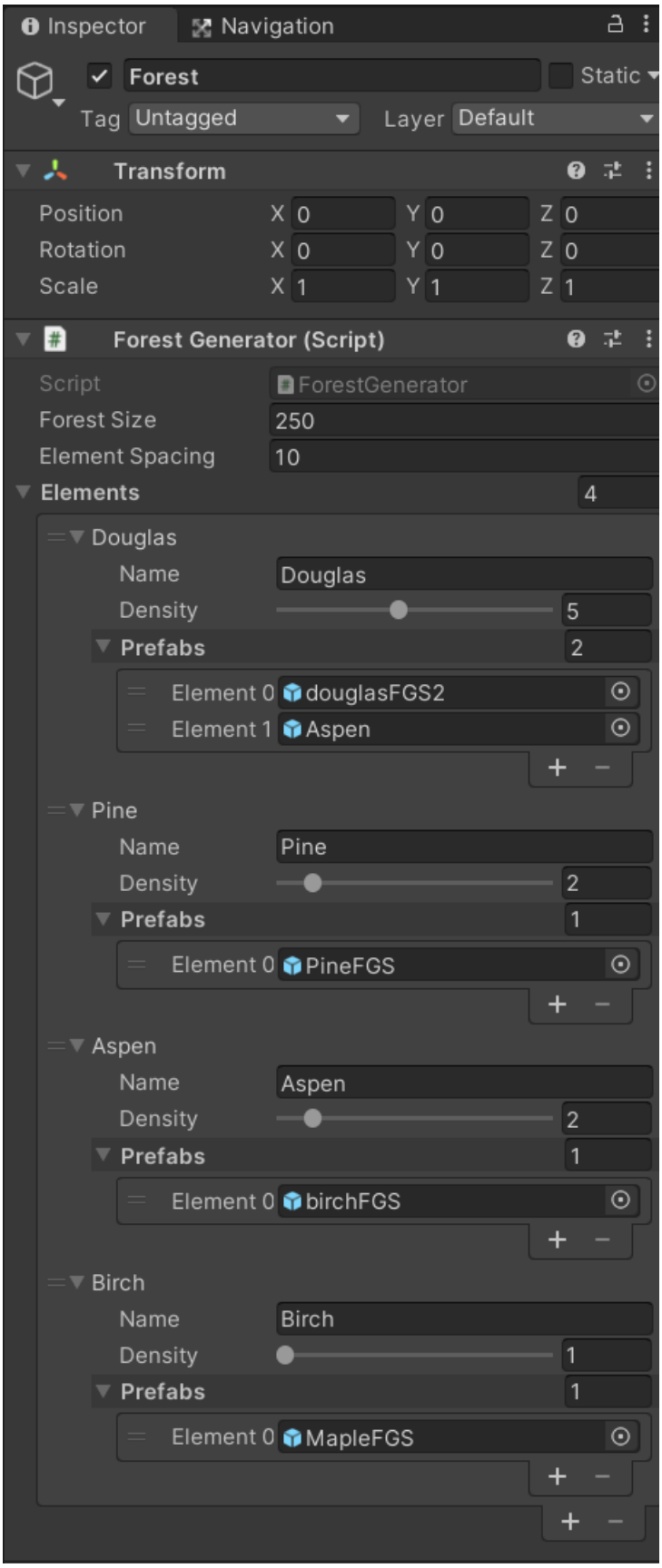


Figure 67 - Forest Object Inspector

Once this object was completed, when the game is started the forest and terrain are both generated correctly as shown in the picture below.

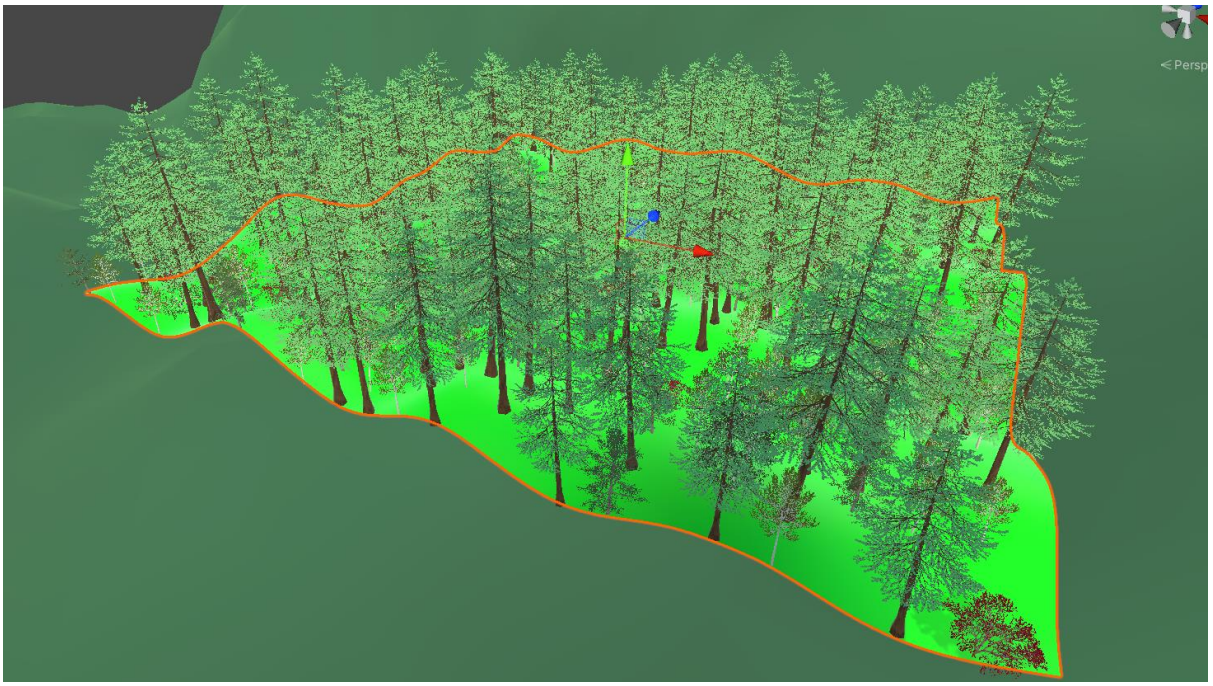
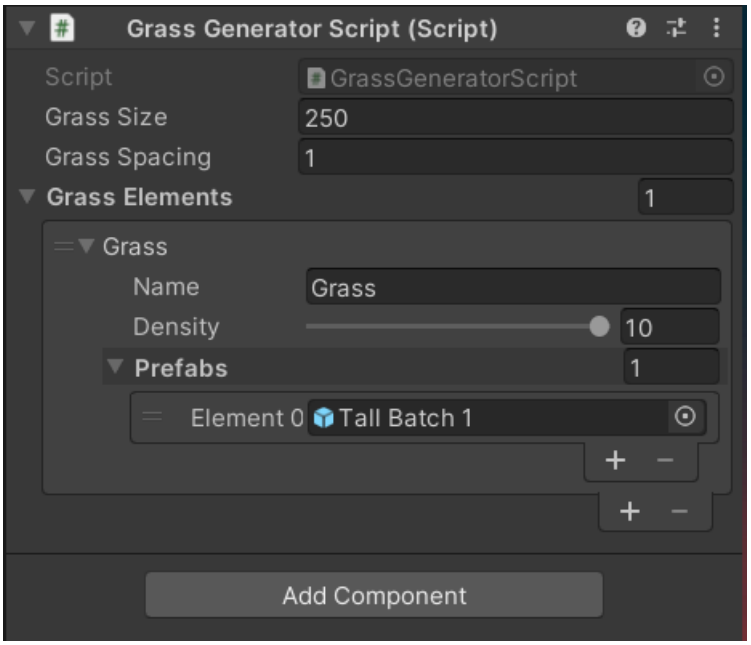


Figure 68 - Terrain and Forest Generating

Grass Generation

Next up is 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 like 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 with the grass generation script attached and the inspector where the object is added and the size, spacing and density are set:



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 at the correct heights and positions of the terrain.

Figure 69 - Grass Generated on Terrain

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 like mountains. The seed is basically a number that is used to generate unique set of mountains each time. The height curve allows me to specify the shape of the curve in the mountains.
Whilst this script also generates the vertices, and triangles just like the terrain script it also has functions that are quite different. Below are two of some of the important functions, one for the seed function and the other for using the octaves and Perlin noise to generate the mountains. There are many more functions in this class, but these are the ones that really set it apart from the other.

Below is the function that gets and uses the seed along with the function to generate the noise heights using Perlin noise.

```
private Vector2[] GetOffsetSeed()
{
    seed = Random.Range(0, 1000);
    // changes area of map
    System.Random prng = new System.Random(seed);
    Vector2[] octaveOffsets = new Vector2[octaves];

    for (int o = 0; o < octaves; o++)
    {
        float offsetX = prng.Next(-100000, 100000);
        float offsetY = prng.Next(-100000, 100000);
        octaveOffsets[o] = new Vector2(offsetX, offsetY);
    }
    return octaveOffsets;
}

private float GenerateNoiseHeight(int z, int x, Vector2[] octaveOffsets)
{
    float amplitude = 12;
    float frequency = 1;
    float persistence = 0.5f;
    float noiseHeight = 0;

    // loop over octaves
    for (int y = 0; y < octaves; y++)
    {
        float mapZ = z / scale * frequency + octaveOffsets[y].y;
        float mapX = x / scale * frequency + octaveOffsets[y].x;

        // Create perlinValues
        // The *2-1 is to create a flat floor level
        float perlinValue = (Mathf.PerlinNoise(mapZ, mapX)) * 2 - 1;
        noiseHeight += heightCurve.Evaluate(perlinValue) * amplitude;
        frequency *= lacunarity;
        amplitude *= persistence;
    }
    return noiseHeight;
}
```

Figure 70 - Mountain Generation Snippets

Below is an image of the inspector where the attributes that were created in the scripts of the Mountains could be changed such as the height curve, size, scale, seed, and other attributes without having to go back into the script and change them.

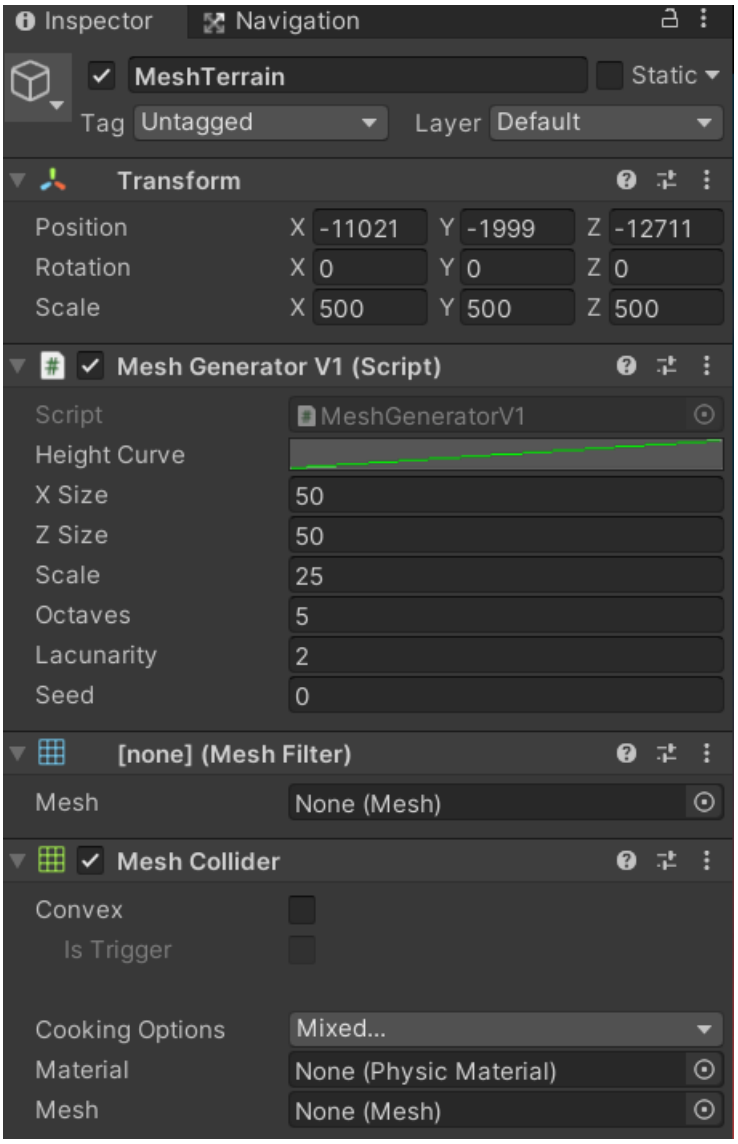


Figure 71 - Mountain Generator Object

The images below show an example of the mountains being generated. A few different examples were taken from the game to show that each time the user plays the game a unique set of mountains would be generated. As you can see the mountains are different each time. This is the same for the forest and the terrain.

Procedurally generated mountains:

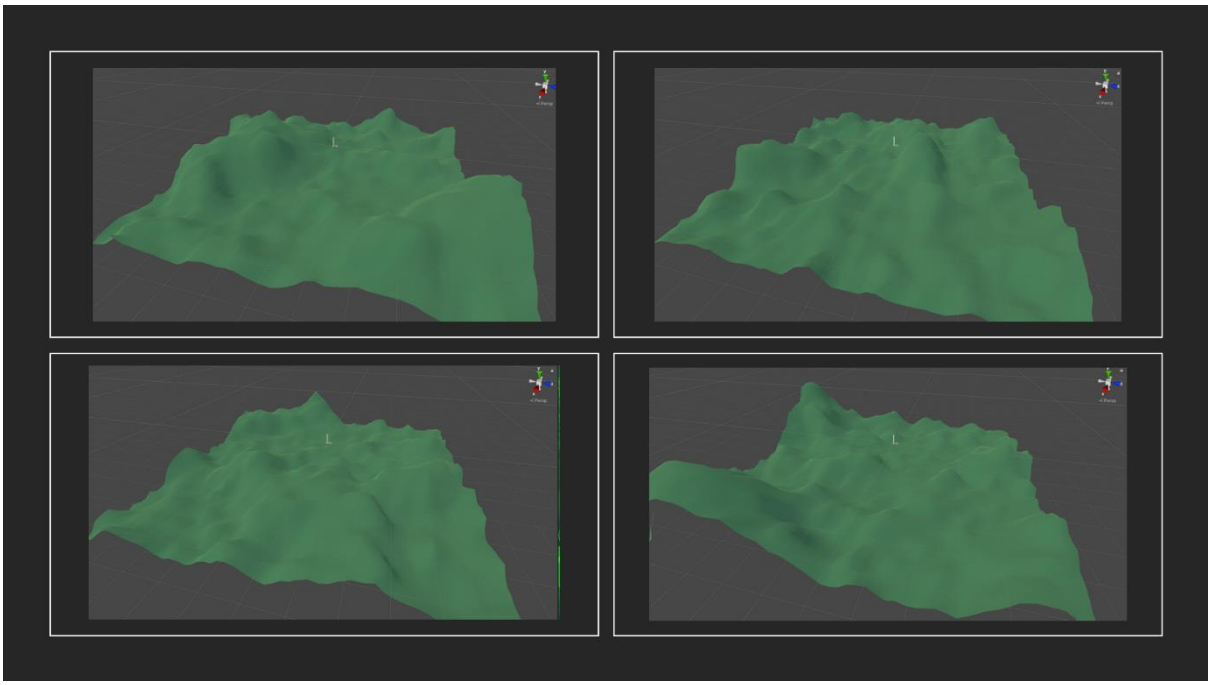


Figure 72 - Generated Mountains

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 specify 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 animals to be generated throughout the map. For example, for the foxes, the scripts allowed a random number of them to be generated each time the 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/generators for the animals. For example, there was a spawner object created for the foxes, the badgers, the squirrels, and the frogs. Each spawner had a script for the animal in which it was generating along with the animal itself. So, the user might play the game and seven badgers could be generated some with certain behaviours and others with different ones which are placed randomly throughout the map. Then, the user might play the game a different time, and this time 5 badgers could be generated at random positions, and they might all have the same behaviour or even different behaviours again.

Below is an example of one of the animal generators scripts to generate the foxes. There are three types of foxes that could be generated, meaning three with different behaviours and attributes. This could generate a random amount of them at random positions once

the game starts.

```
public class FoxGen : MonoBehaviour
{

    public GameObject FoxOne;
    public GameObject FoxTwo;
    public GameObject FoxThree;
    public int xPos;
    public int zPos;
    public int FoxCount;
    public int RandomRange;

    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(FoxDrop());
    }

    IEnumerator FoxDrop()
    {
        RandomRange = Random.Range(1, 3);

        while (FoxCount <= RandomRange)
        {
            xPos = Random.Range(-48, -94);
            zPos = Random.Range(121, 178);
            Instantiate(FoxOne, new Vector3(xPos, 7, zPos), Quaternion.identity);
            Instantiate(FoxTwo, new Vector3(xPos, 7, zPos), Quaternion.identity);
            Instantiate(FoxThree, new Vector3(xPos, 7, zPos), Quaternion.identity);
            yield return new WaitForSeconds(0.1f);
            FoxCount += 1;
        }
    }


}
```

Figure 73 - Snippet of Fox Generator Scripts

Below is the object and inspector of the fox spawner, allowing me to add the fox objects to the scripts references to be generated.

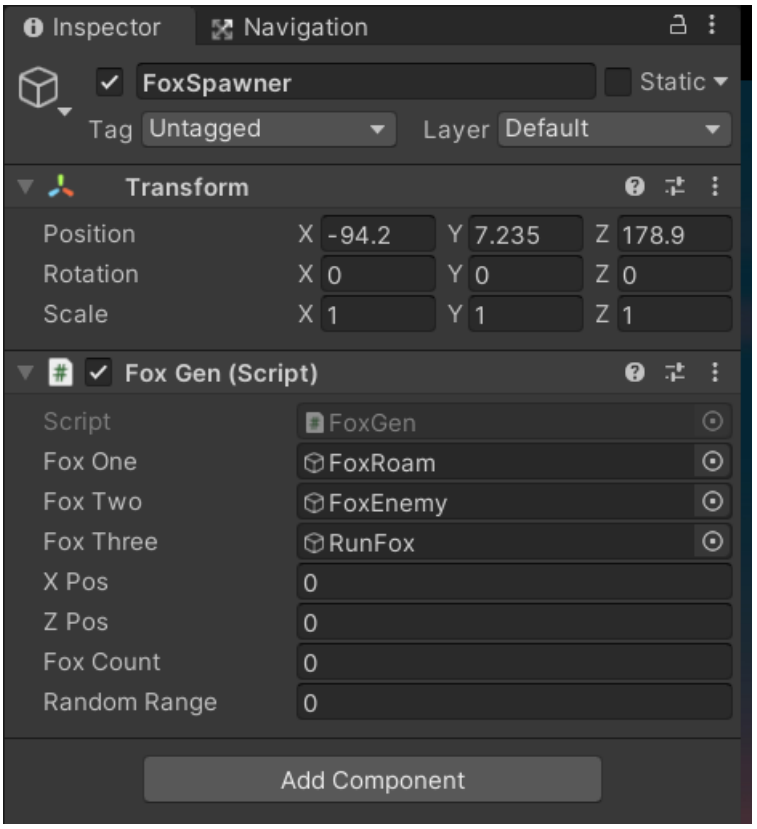


Figure 74 - Fox Gen Object

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 squirrels being Generated at random locations and with different behaviours:

Figure 75 - Squirrel Area

4.3.2. 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 each time the game was played. 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 so the AI can move throughout these areas.

In total there seven different C# scripts involved in the making of the AI system. Four of them were used to set the behaviours of the animals and the other three were used to build the areas on which these behaviours would work. These areas were called the NavMeshes. The first step of the process was to create C# 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 script 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.

Shown below is a snippet of some of the code used to create the aggressive behaviours. There is quite a lot of code used for this system so only small important parts will be shown.

```
// Update is called once per frame
void Update()
{

    //calcualting distance between the two vector3s which is the player and the agent(enemy) - this.transform
    //
    Distance = Vector3.Distance(PlayerVar.transform.position, this.transform.position);


    //if statement to set what distance the enemy follows the player
    //
    if (Distance <= 10)
    {
        isAlert = true;
    }
    //tells enemy when to give up
    //
    if (Distance > 10f)
    {
        isAlert = false;
    }
    //if isAlert to false. starts to follow player if its close.
    //
    if (isAlert)
    {

        badgerAgent.isStopped = false;
        badgerAgent.SetDestination(PlayerVar.transform.position);
    }
    //if its not alert set isstopped top true which stops the enemy if you get too far away.
    //
    if (!isAlert)
    {
        badgerAgent.isStopped = true;
    }
}
```

Figure 76 - Update Function for Attacking Player

Below is an image of a badger with these traits chasing the player in game:

Figure 77 - Badger Attacking Player

Another image below shows a badger with aggressive traits in an idle position as the player hasn't come close enough for the badger to detect them. Beside it is a white badger with different behaviours and colours which wanders around and runs away instead.

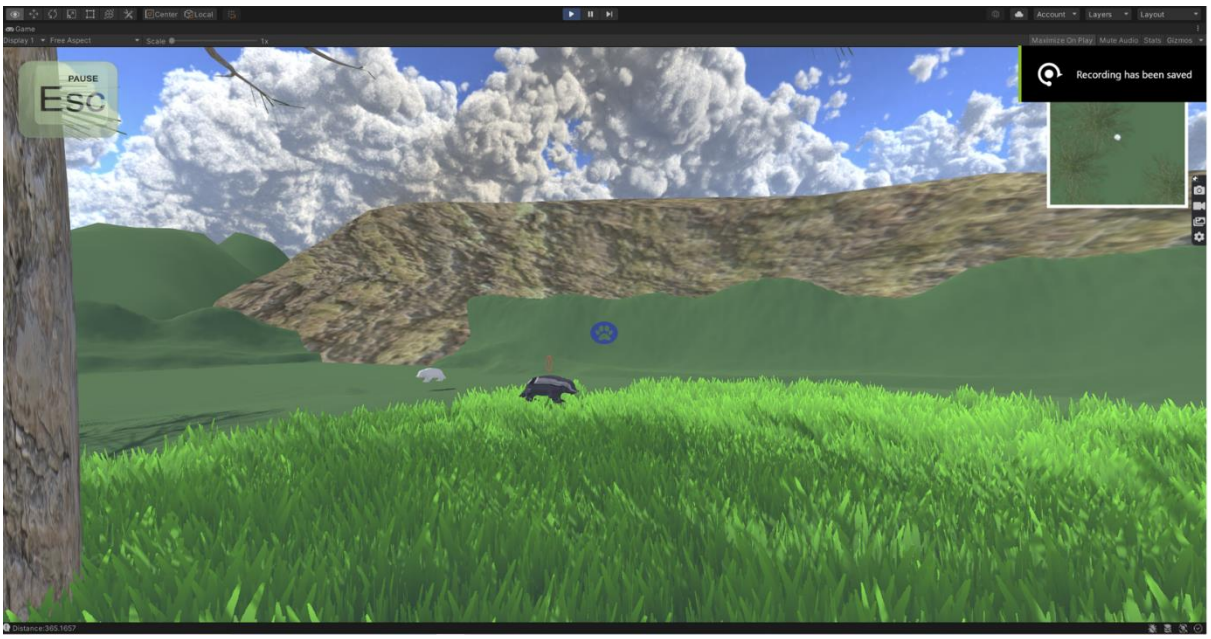


Figure 78 - Wandering badger and attacking badger

The next type of behaviour scripts was set to be the most assigned behaviour script. The behaviours included the animal roaming or wandering around the environment and if the player approached 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.
Small snippet from this script which shows running away function and part of the roaming function.

```
// Update is called once per frame
void Update()
{
    float distance = Vector3.Distance(transform.position, Player.transform.position);

    Debug.Log("Distance:" + distance);

    //run away from player

    if (distance < EnemyDistanceRun)
    {
        Vector3 dirToPlayer = transform.position - Player.transform.position;

        Vector3 newPos = transform.position + dirToPlayer;

        _agent.SetDestination(newPos);
    }
}

private void OnTriggerEnter(Collider other)
{
    if(other.tag == "1")
    {
        _agent.SetDestination(pos2.position);
    }
    if (other.tag == "2")
    {
        _agent.SetDestination(pos3.position);
    }
    if (other.tag == "3")
    {
        _agent.SetDestination(pos4.position);
    }
    if (other.tag == "4")
    {
        _agent.SetDestination(pos5.position);
    }
    if (other.tag == "5")
    {
        _agent.SetDestination(pos6.position);
    }
    if (other.tag == "6")
    {
        _agent.SetDestination(pos7.position);
    }
```

Figure 79 - Roaming function and Run away

Shown below is an image of a squirrel that was generated with these behaviours which wanders around form different points and then runs away if approached.

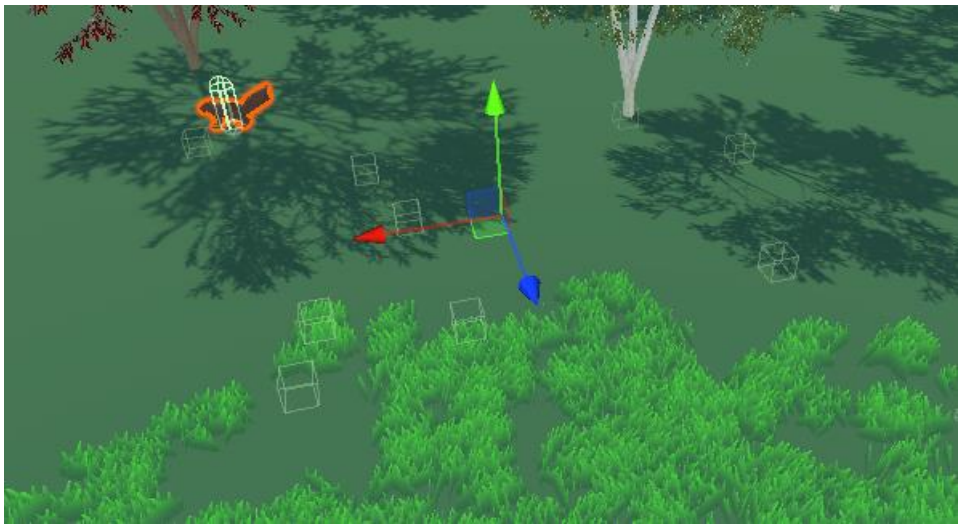


Figure 80 - Squirrel Wandering from different points

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

Below is an image of an instance where the game was started and multiple foxes with the chase behaviour were generated, and they are all following a fox that was generated with the running away behaviour from these other foxes.

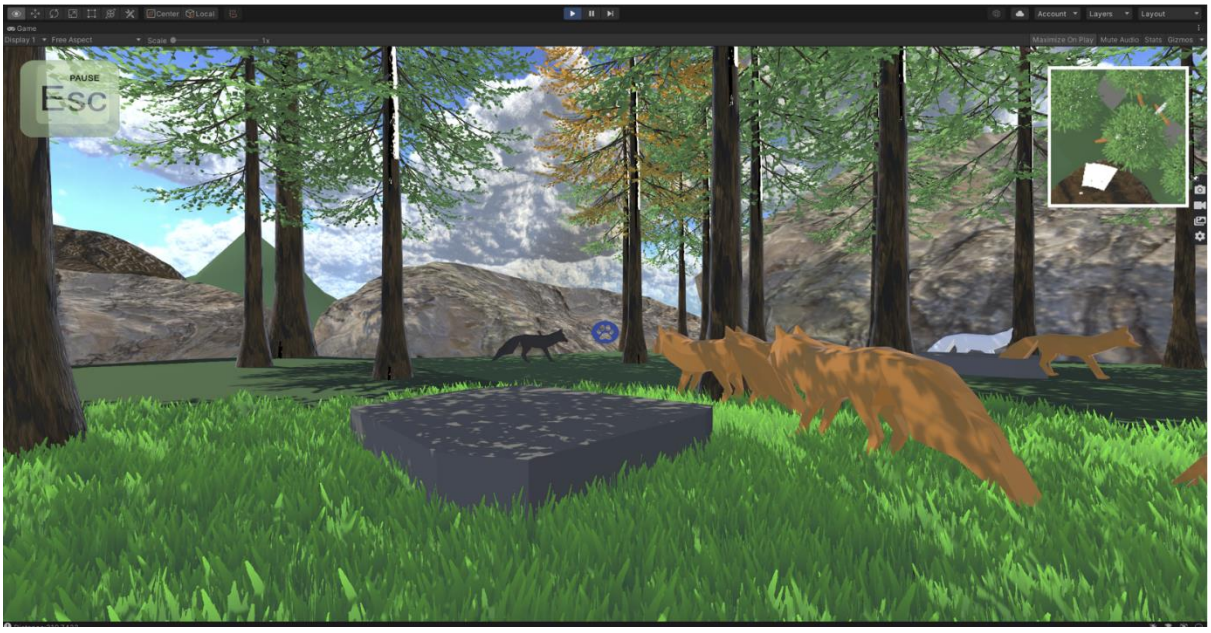


Figure 81 - Fox area Chasing other foxes

Figure 82 - Second image of foxes chasing each other in Groups

Below is a small snippet of some of the code for the animals that chase other animals:

```
private void Awake()
{
    Agent = GetComponent<NavMeshAgent>();
}

// Start is called before the first frame update
private void Start()
{
    StartCoroutine(FollowTarget());
}

private IEnumerator FollowTarget()
{
    WaitForSeconds Wait = new WaitForSeconds(UpdateSpeed);

    while (enabled)
    {
        Agent.SetDestination(Target.transform.position);
        yield return Wait;
    }
}
```

Figure 83 - following coroutine

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 are being generated needed to be built so that the animals could interact with the environment. To do these scripts were created to set the areas as NavMesh Surfaces so that the AI could use the scripts they were assigned, which uses Unity's AI pathfinding technology called

NavMesh.

To do this each surface that is generated is assigned a script called NavMeshSurface. This script represents the walkable area for the animals to walk on and where the NavMesh should be built. This is quite a long script so only snippets will be discussed. Below is an example of one of the terrain objects having the scripts attached:

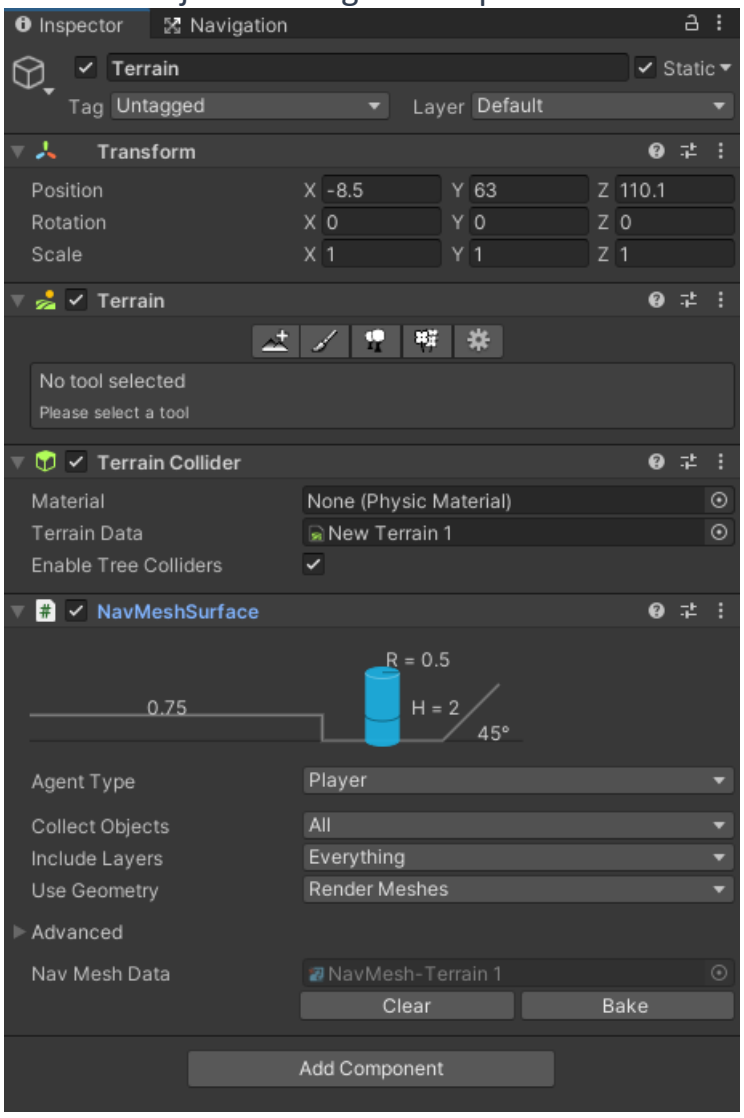


Figure 84 - NavMeshSurface script attached to object

This script was extremely long so an image of the function for building the NavMesh is shown below which allows the surface to be created so the scripts the AI use can work using the NavMesh Agent variables.

```
public void BuildNavMesh()
{
    var sources = CollectSources();

    // Use unscaled bounds - this differs in behaviour from e.g. collider components.
    // But is similar to reflection probe - and since navmesh data has no scaling support - it is the right choice here.
    var sourcesBounds = new Bounds(m_Center, Abs(m_Size));
    if (m_CollectObjects == CollectObjects.All || m_CollectObjects == CollectObjects.Children)
    {
        sourcesBounds = CalculateWorldBounds(sources);
    }

    var data = NavMeshBuilder.BuildNavMeshData(GetBuildSettings(),
            sources, sourcesBounds, transform.position, transform.rotation);

    if (data != null)
    {
        data.name = gameObject.name;
        RemoveData();
        m_NavMeshData = data;
        if (isActiveAndEnabled)
            AddData();
    }
}
```

Figure 85 - Building the NavMesh Function

To make this AI system a bit more complex two more scripts were used in the system which allowed some of the animals to jump to another NavMesh surface which created a 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 in case, they were generated somewhere that they weren't supposed to be.

The script which allowed the animals to jump from different surfaces was attached to some of the animals that were generated. This script was called 'Animal Link Mover'. You can see below it was attached to one of the foxes that was generated.

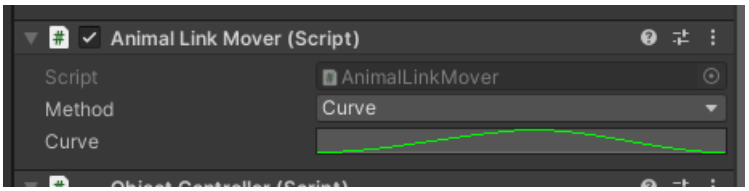


Figure 86 - Jump Curve script reference object

This allows me to specify what curve the animal jumps at.

Once these scripts were written they were attached to various animals and surfaces. 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, ten squirrels are generated, four of them might wander around the terrain and the other six might be idle and run away from the player.

4.3.3. Inspection System

The next feature that played an important part of this project is the inspection system which uses a technology in C# called Ray casting. 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. The user can inspect animals and object called information cubes which are cubes used to represent information about the different trees. To get an idea of what this inspection system looks like, images are shown below:

Figure 87 - Info cube inspection

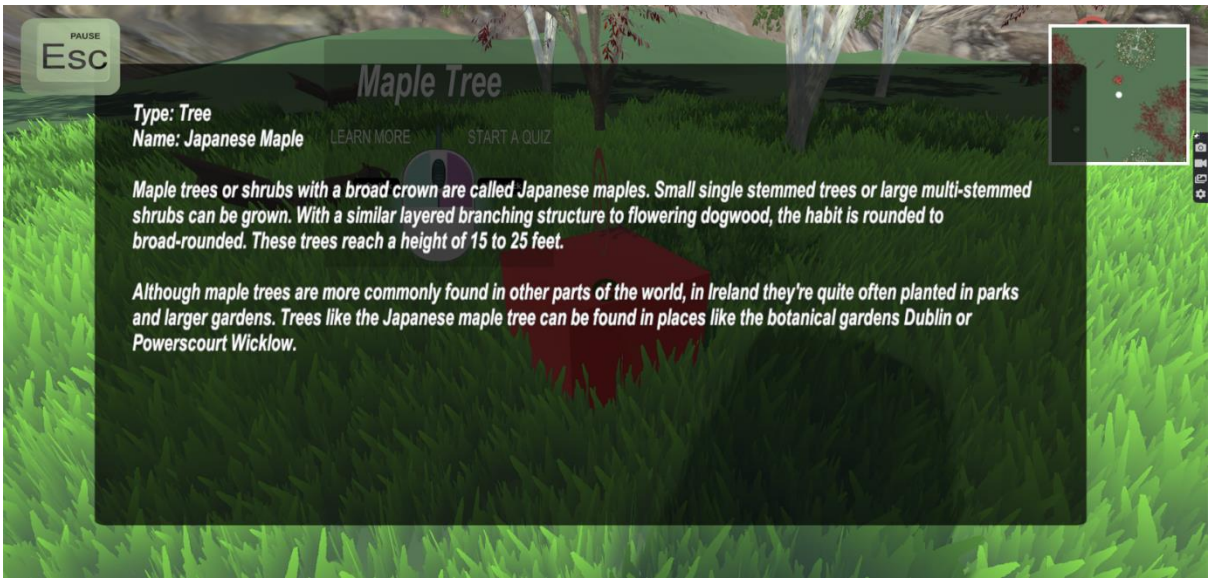


Figure 88 - information for maple tree

Objects like the common frog and other wildlife will also have an exclamation symbol displayed above it so it is easier for the user to find it and they know that they are able to inspect it.

Figure 89 - Frog being Inspected

In total, there were four C# scripts created in the making of this feature. The first one was a class called 'Inspect Ray cast' this script uses the Ray casting technique which was mentioned before. This technique allows the player to project an invisible laser (Ray cast) to be able to detect any object that's in the player line of sight. When the player looks at a piece of wildlife, they will be able to click to view information about the wildlife they are inspecting. A crosshair which you can see in the images above is used to represent whether the animal is detected or not. It goes green when an animal is in front and blue when not.

Figure 90 – Cross Hair Changing colour

This is a large class so only a small snippet of part of the script will be shown showcasing the ray cast being created and allowing the crosshair to change when it detects and animal.

```
private void Update()
{
    RaycastHit hit;
    Vector3 fwd = transform.TransformDirection(Vector3.forward);

    if(Physics.Raycast(transform.position, fwd, out hit, rayLength, layerMaskInteract.value))
    {
        if (hit.collider.CompareTag("InteractObject"))
        {
            if (!doOnce)
            {
                raycastedObj = hit.collider.gameObject.GetComponent<ObjectController>();

                raycastedObj.ShowObjectName();
                CrosshairChange(true);
            }

            isCrosshairActive = true;
            doOnce = true;
```

Figure 91 - Ray Casting function Snippet to detect object

Once this class was finished a script was created called 'Inspect Controller' so the UI could be managed for the inspection system to be linked with the 'Inspect Ray cast' class. This allowed the UI for the information, text, and images to be displayed when using the Ray cast 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:

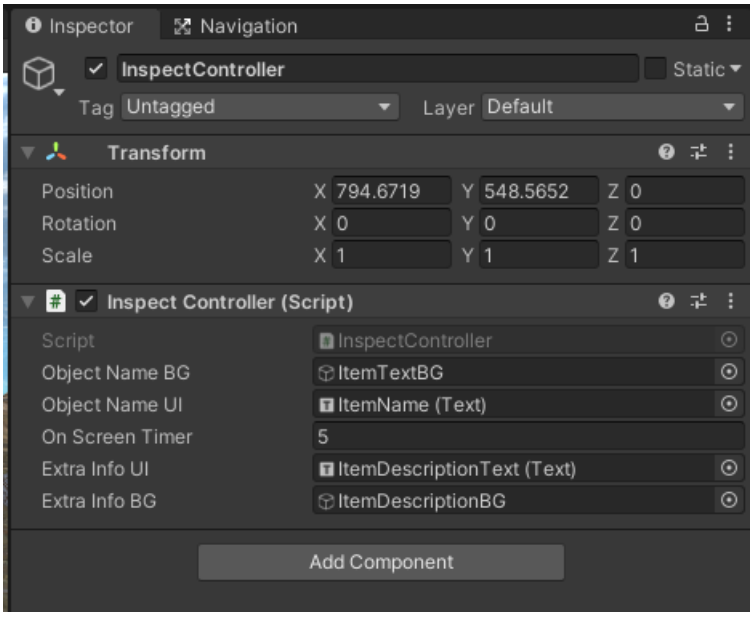


Figure 92 - Inspect Controller Object

Below is a quick snippet of some of the code for the script for showing the information on each object:

```
public void ShowName(string objectName)
{
    objectNameBG.SetActive(true);
    objectNameUI.text = objectName;
}

public void HideName()
{
    objectNameBG.SetActive(false);
    objectNameUI.text = "";
}

public void ShowAdditionalInfo(string newInfo)
{
    timer = onScreenTimer;
    startTimer = true;
    extraInfoBG.SetActive(true);
    extraInfoUI.text = newInfo;
}

void ClearAdditionalInfo()
{
    extraInfoBG.SetActive(false);
    extraInfoUI.text = "";
}
```

Figure 93 - Setting UI to turn off and on

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 script attached to it. For example, every frog that is generated has the object controller script attached which references the ‘Inspect Controller’ script which in turn references the ‘Inspect Ray cast’ script which then allows the inspection system to work.

Here is the object controller class which acts as a controller for the inspection system that’s attached to the wildlife.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;


public class ObjectController : MonoBehaviour
{
    [SerializeField] private string objectName;



    [TextArea] [SerializeField] private string extraInfo;




    [SerializeField] private InspectController inspectController;

    public void ShowObjectName()
    {
        inspectController.ShowName(objectName);
    }

    public void HideObjectName()
    {
        inspectController.HideName();
    }



    public void ShowExtraInfo()
    {
        inspectController.ShowAdditionalInfo(extraInfo);
    }
}
```

Figure 94 - Object controller calling other scripts

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

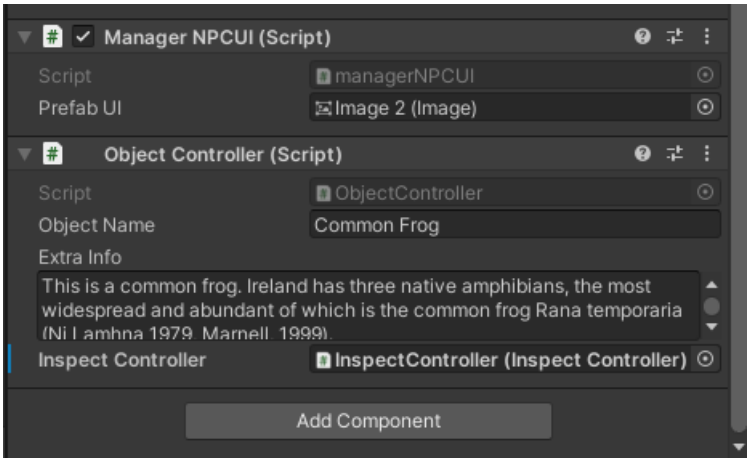


Figure 95 - Object controller references for information

The last part involved in the inspection system was the ‘Manager NPCUI’ class. You can

see that the frog example above has this script attached. This allowed a symbol to be displayed over certain animals' bodies to notify the player that it is nearby so it's easier to find. This script was attached on to various animals that were generated. An example of this is shown below for the frog:

Figure 96 - Frog with Icon Hovering over it.

When the user approaches the animal the exclamation symbol gets smaller and when you move further away it gets bigger.
Below is an image of some of the 'Manager NPCUI' class. Which shows how the exclamations image used gets bigger and smaller and sticks to the animal.

```
public class managerNPCUI : MonoBehaviour
{
    //private GameObject objPlayer;
    public Image prefabUI;
    private Image uiUse;
    private Transform tr_head;
    private Vector3 offset = new Vector3(0, 1.5f, 0);

    // Start is called before the first frame update
    void Start()
    {
        //objPlayer = GameObject.FindGameObjectWithTag("Player");
        uiUse = Instantiate(prefabUI, FindObjectOfType<Canvas>().transform).GetComponent<Image>();
        tr_head = transform.GetChild(0);
    }

    // Update is called once per frame
    void Update()
    {
        uiUse.transform.position = Camera.main.WorldToScreenPoint(transform.position + offset);
```

Figure 97 - functions showing image over animal

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 Ray casting allows the prompt for the animal's 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.

Figure 98 - Looking at object

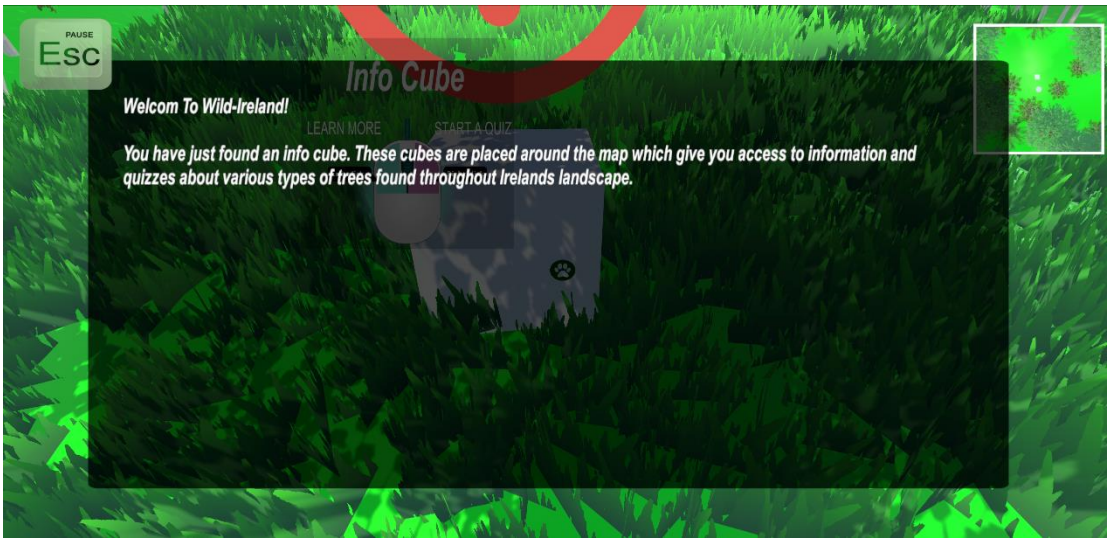


Figure 99 - Left click to view information

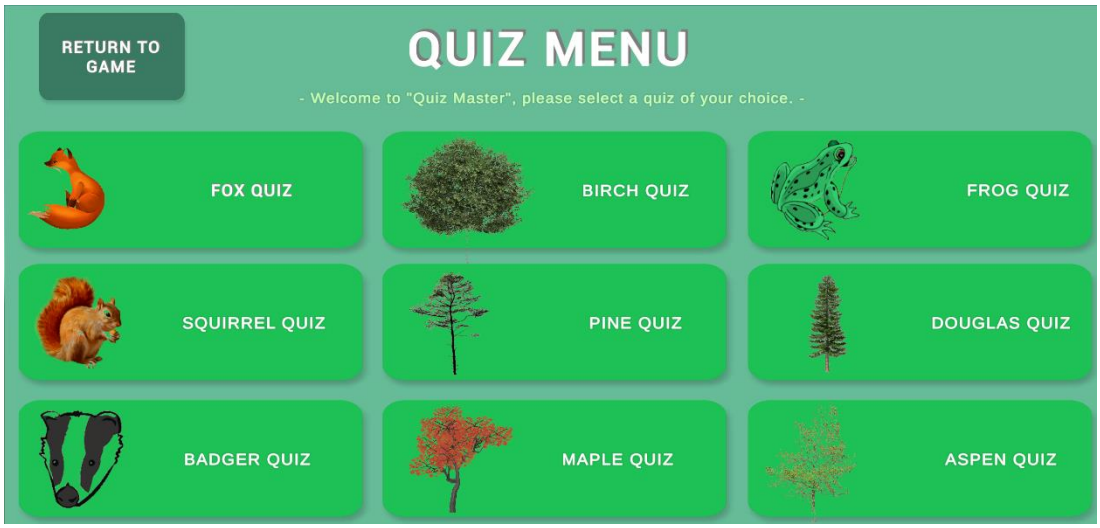


Figure 100 - Right click to view quiz menu

4.3.4. Quiz System

The next feature that will be discussed is the quiz system feature. This feature is a big part of the game and required lots of different scripts and classes written 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 in this feature but only a few will be discussed in detail. Although the methods used weren't as complex as the AI or procedural generation, this feature took a lot of time proved to be one of the harder aspects to code.

In total there were nine different quizzes in the game, one for each type of animal and tree. All the quizzes are accessed by the user through the Quiz Menu as shown above. The quiz systems two main classes which are the most important and the longest of the scripts used to create the quiz are called 'Game Manager' and 'UI Manager'. The 'Game Manager' script had 10 different variations of it for each quiz, but as they only shared minor differences only the main 'Game Manager' script will be discussed in detail. 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.

Below is an image of the UI created for one of the quizzes:

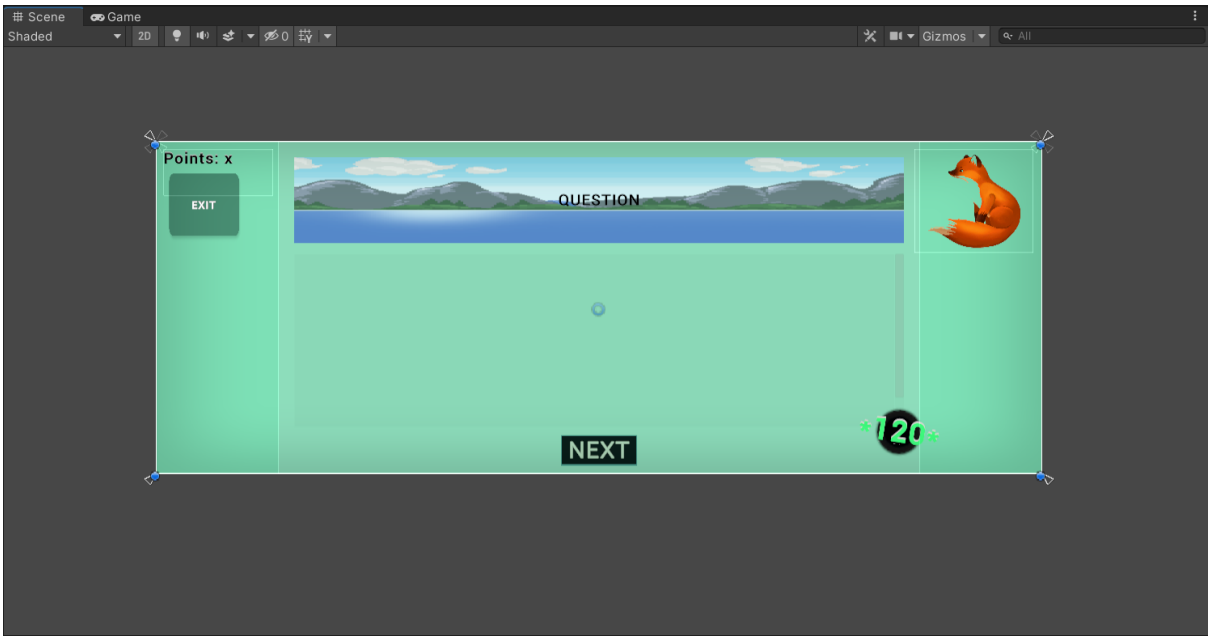


Figure 101 - UI Creation

Then, an object was created for the ‘Game Manager’ and the ‘UI Manager’ scripts to be attached to called ‘Manager’. These objects allowed all the references for objects and UI to be managed and set.
Image of manager object with scripts attached:

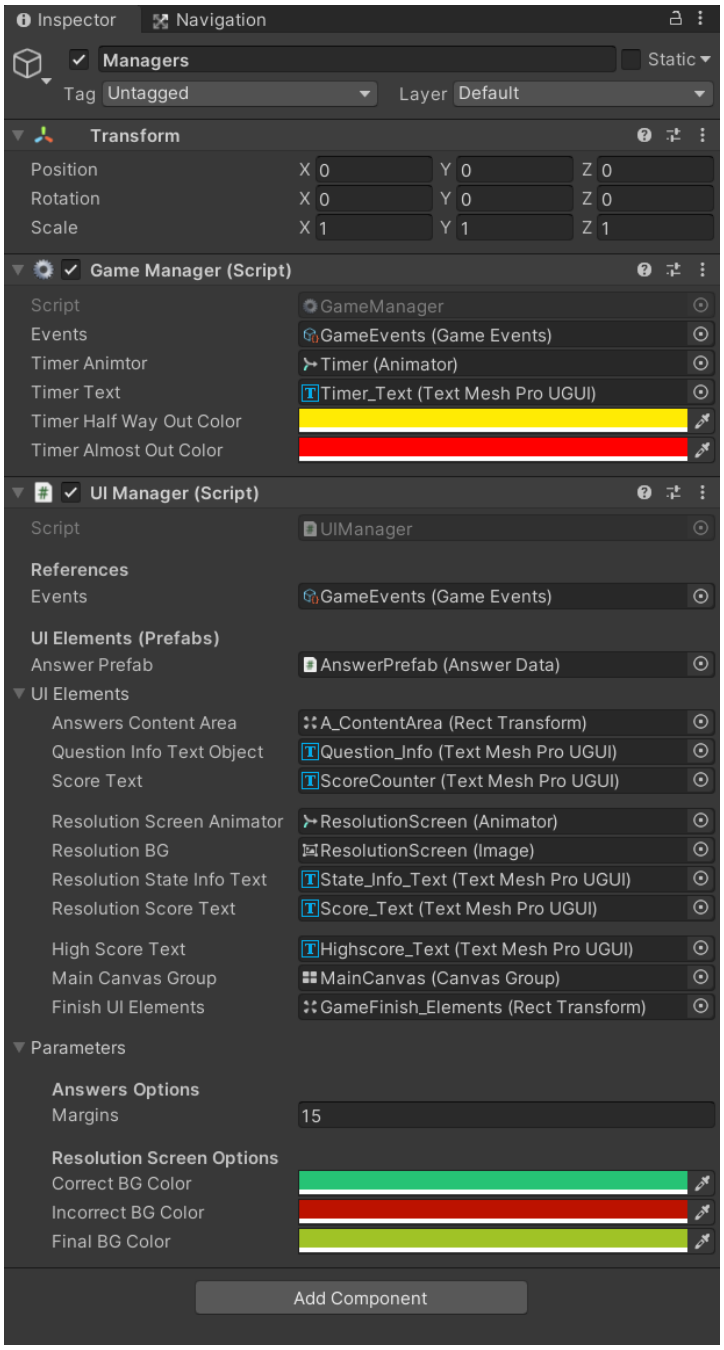


Figure 102 - Managers Object Inspector

In short, the ‘Game Manager’ script basically managed the quiz which loaded the questions, updated the answers, getting random questions, checking if the answers are correct or incorrect, starting the timer of the quiz, comparing answers, restarting the quiz, setting the high score, updating the score, and just overall managing the quiz. This script was extremely long, 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.

```
public class GameManager : MonoBehaviour {




    #region Variables

    private             Question[]          _questions              = null;
    public              Question[]          Questions               { get { return _questions; } }

    [SerializeField]    GameEvents          events                  = null;

    [SerializeField]    Animator            timerAnimtor            = null;
    [SerializeField]    TextMeshProUGUI     timerText               = null;
    [SerializeField]    Color               timerHalfWayOutColor    = Color.yellow;
    [SerializeField]    Color               timerAlmostOutColor     = Color.red;
    private             Color               timerDefaultColor       = Color.white;

    private             List<AnswerData>    PickedAnswers           = new List<AnswerData>();
    private             List<int>           FinishedQuestions       = new List<int>();
    private             int                 currentQuestion         = 0;

    private             int                 timerStateParaHash      = 0;

    private             IEnumerator         IE_WaitTillNextRound    = null;
    private             IEnumerator         IE_StartTimer           = null;

    private             bool                IsFinished
    {
        get
        {
            return (FinishedQuestions.Count < Questions.Length) ? false : true;
        }
    }
```

Figure 103 - Snippet of Game Manager Variables

The ‘UI manager’ scripts which managed the UI of the quiz to decide what ui was to be displayed when using the quiz uses a struct which is basically a data type that defines a grouped list of variables under one name in a block of memory which allows the different variables to be accessed via a single pointer.
This script also uses a series of functions to do things like update the question UI, display resolution screens with the score counter, switch the timer, calculate the score, create, and erase answers, and update the score UI.
This script was also very long so below is a snippet of the struct followed by a snippet of some of the methods used like updating the questions, displaying the resolutions screens, and creating answers and calculating the score.

```
[Serializable()]
public struct UIElements
{
    [SerializeField] RectTransform answersContentArea;
    public RectTransform AnswersContentArea { get { return answersContentArea; } }

    [SerializeField] TextMeshProUGUI questionInfoTextObject;
    public TextMeshProUGUI QuestionInfoTextObject { get { return questionInfoTextObject; } }

    [SerializeField] TextMeshProUGUI scoreText;
    public TextMeshProUGUI ScoreText { get { return scoreText; } }

    [Space]

    [SerializeField] Animator resolutionScreenAnimator;
    public Animator ResolutionScreenAnimator { get { return resolutionScreenAnimator; } }

    [SerializeField] Image resolutionBG;
    public Image ResolutionBG { get { return resolutionBG; } }

    [SerializeField] TextMeshProUGUI resolutionStateInfoText;
    public TextMeshProUGUI ResolutionStateInfoText { get { return resolutionStateInfoText; } }

    [SerializeField] TextMeshProUGUI resolutionScoreText;
    public TextMeshProUGUI ResolutionScoreText { get { return resolutionScoreText; } }

    [Space]

    [SerializeField] TextMeshProUGUI highScoreText;
    public TextMeshProUGUI HighScoreText { get { return highScoreText; } }

    [SerializeField] CanvasGroup mainCanvasGroup;
    public CanvasGroup MainCanvasGroup { get { return mainCanvasGroup; } }

    [SerializeField] RectTransform finishUIElements;
    public RectTransform FinishUIElements { get { return finishUIElements; } }
}
```

Figure 104 - Structs Snippet

Below are the functions for updating, displaying, and timing the Questions:

```
/// </summary>
void UpdateQuestionUI(Question question)
{
    uIElements.QuestionInfoTextObject.text = question.Info;
    CreateAnswers(question);
}
/// <summary>
/// Function that is used to display resolution screen.
/// </summary>
void DisplayResolution(ResolutionScreenType type, int score)
{
    UpdateResUI(type, score);
    uIElements.ResolutionScreenAnimator.SetInteger(resStateParaHash, 2);
    uIElements.MainCanvasGroup.blocksRaycasts = false;

    if (type != ResolutionScreenType.Finish)
    {
        if (IE_DisplayTimedResolution != null)
        {
            StopCoroutine(IE_DisplayTimedResolution);
        }
        IE_DisplayTimedResolution = DisplayTimedResolution();
        StartCoroutine(IE_DisplayTimedResolution);
    }
}
IEnumerator DisplayTimedResolution()
{
    yield return new WaitForSeconds(GameUtility.ResolutionDelayTime);
    uIElements.ResolutionScreenAnimator.SetInteger(resStateParaHash, 1);
    uIElements.MainCanvasGroup.blocksRaycasts = true;
}
```

Figure 105 - Updating and Display Snippets

More functions are shown below for calculating the score, create answers and erasing answers.

```
IEnumerator CalculateScore()
{
    var scoreValue = 0;
    while (scoreValue < events.CurrentFinalScore)
    {
        scoreValue++;
        uIElements.ResolutionScoreText.text = scoreValue.ToString();

        yield return null;
    }
}

/// <summary>
/// Function that is used to create new question answers.
/// </summary>
void CreateAnswers(Question question)
{
    EraseAnswers();

    float offset = 0 - parameters.Margins;
    for (int i = 0; i < question.Answers.Length; i++)
    {
        AnswerData newAnswer = (AnswerData)Instantiate(answerPrefab, uIElements.AnswersContentArea);
        newAnswer.UpdateData(question.Answers[i].Info, i);

        newAnswer.Rect.anchoredPosition = new Vector2(0, offset);

        offset -= (newAnswer.Rect.sizeDelta.y + parameters.Margins);
        uIElements.AnswersContentArea.sizeDelta = new Vector2(uIElements.AnswersContentArea.sizeDelta.x, offset * -1);

        currentAnswers.Add(newAnswer);
    }
}
/// <summary>
/// Function that is used to erase current created answers.
/// </summary>
void EraseAnswers()
{
    foreach (var answer in currentAnswers)
    {
        Destroy(answer.gameObject);
    }
    currentAnswers.Clear();
}
```

Figure 106 - Functions for score, questions, and answers snippet

Once these scripts were written they were attached to the managers game object and all the references 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 to create and edit questions within unity in the inspector 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.

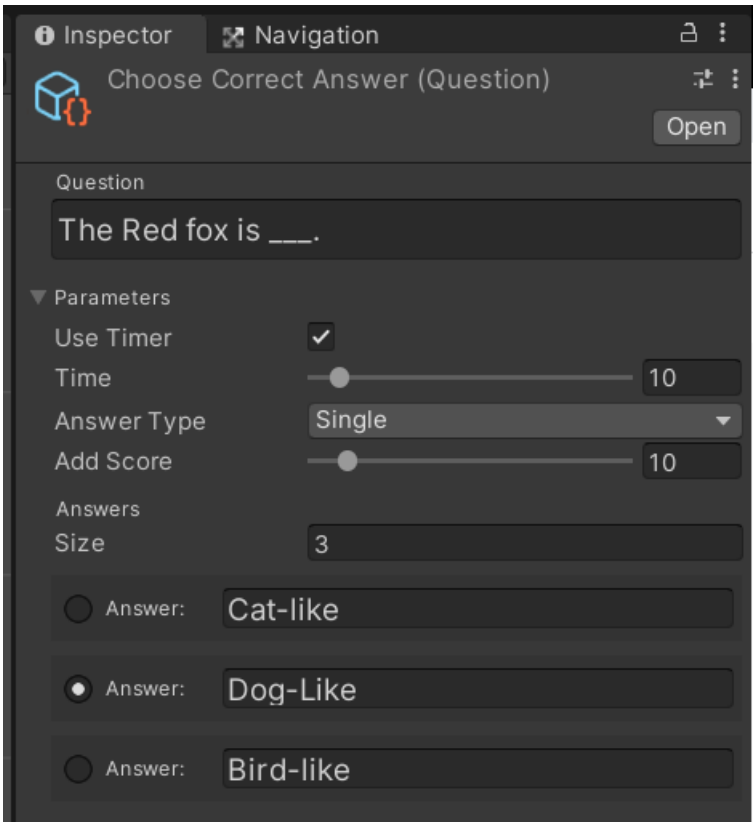


Figure 107 - inspector where questions were edited

Snippet of Function from 'question editor' script 'Draw Answers()' function. allows the answers to be checked if correct or not.

```
void DrawAnswers ()
{
    EditorGUILayout.BeginVertical();

    EditorGUILayout.PropertyField(arraySizeProp);
    GUILayout.Space(5);

    EditorGUI.indentLevel++;
    for (int i = 0; i < arraySizeProp.intValue; i++)
    {
        EditorGUI.BeginChangeCheck();
        EditorGUILayout.PropertyField(answersProp.GetArrayElementAtIndex(i));
        if (EditorGUI.EndChangeCheck())
        {
            if (answerTypeProp.enumValueIndex == (int)Question.AnswerType.Single)
            {
                SerializedProperty isCorrectProp = answersProp.GetArrayElementAtIndex(i).FindPropertyRelative("_isCorrect");

                if (isCorrectProp.boolValue)
                {
                    UncheckCorrectAnswers();
                    answersProp.GetArrayElementAtIndex(i).FindPropertyRelative("_isCorrect").boolValue = true;

                    serializedObject.ApplyModifiedProperties();
                }
            }
        }
        GUILayout.Space(5);
    }

    EditorGUILayout.EndVertical();
    EditorGUI.indentLevel--;
}
```

Figure 108 - Draw Answers Function

‘Answer Data’:

This script 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 script is referenced in the ‘UI Managers’ Script.

```
public class AnswerData : MonoBehaviour {

    Variables

    /// <summary>
    /// Function that is called to update the answer data.
    /// </summary>
    public void UpdateData (string info, int index)
    {
        infoTextObject.text = info;
        _answerIndex = index;
    }
    /// <summary>
    /// Function that is called to reset values back to default.
    /// </summary>
    public void Reset ()
    {
        Checked = false;
        UpdateUI();
    }
    /// <summary>
    /// Function that is called to switch the state.
    /// </summary>
    public void SwitchState ()
    {
        Checked = !Checked;
        UpdateUI();

        if (events.UpdateQuestionAnswer != null)
        {
            events.UpdateQuestionAnswer(this);
        }
    }
    /// <summary>
    /// Function that is called to update UI.
    /// </summary>
    void UpdateUI ()
    {
        if (toggle == null) return;

        toggle.sprite = (Checked) ? checkedToggle : uncheckedToggle;
    }
}
```

Figure 109 - Storing Answer Data

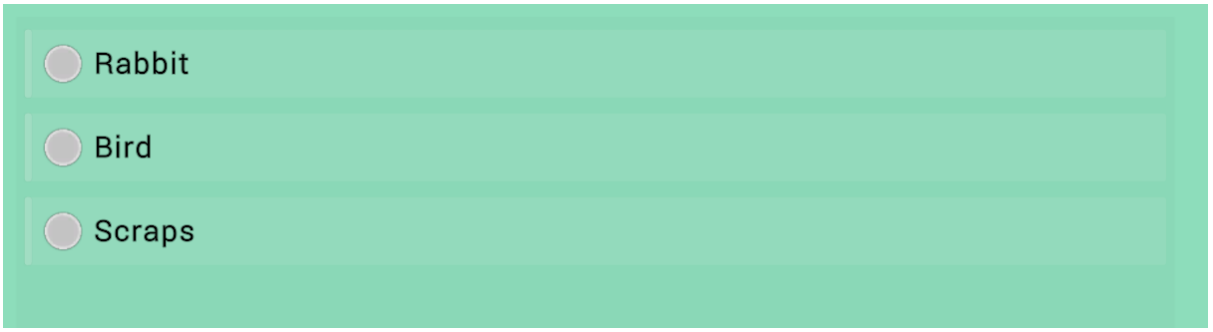
‘Answer Drawer’:

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

```
[CustomPropertyDrawer(typeof(Answer))]
public class Answer_Drawer : PropertyDrawer {

    public override float GetPropertyHeight(SerializedProperty property, GUIContent label)
    {
        return base.GetPropertyHeight(property, label) + 15;
    }
    public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
    {
        GUI.Box(position, GUIContent.none);

        SerializedProperty isCorrectProp = property.FindPropertyRelative("_isCorrect");
        Rect isCorrectRect = new Rect(position.x + 5, (position.y + position.height / 2) - 7.5f, 15, 15);
        isCorrectProp.boolValue = EditorGUI.Toggle(isCorrectRect, isCorrectProp.boolValue, EditorStyles.radioButton);

        Rect labelRect = new Rect(position.x + isCorrectRect.width + 10, position.y, 45, position.height);
        GUIStyle labelStyle = new GUIStyle(EditorStyles.miniLabel)
        {
            alignment = TextAnchor.MiddleLeft
        };
        GUI.Label(labelRect, new GUIContent("Answer:", "Define an Answer."), labelStyle);

        SerializedProperty infoProp = property.FindPropertyRelative("_info");
        Rect infoRect = new Rect(labelRect.x + labelRect.width, (position.y + position.height / 2) - 10, position.width - 75, 20);
        GUIStyle textAreaStyle = new GUIStyle(EditorStyles.textArea)
        {
            fontSize = 15,
            fixedHeight = 20
        };
        infoProp.stringValue = EditorGUI.TextArea(infoRect, infoProp.stringValue, textAreaStyle);
    }
}
```

Figure 110 - Layout manager for quiz

Image of UI for answers:



‘Question’:

The question script basically allowed the user to 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 quizzed on. It also allowed the quiz to get and return the correct answers.
Image of creating new question with right click in the Unity editor, which was enabled by script:

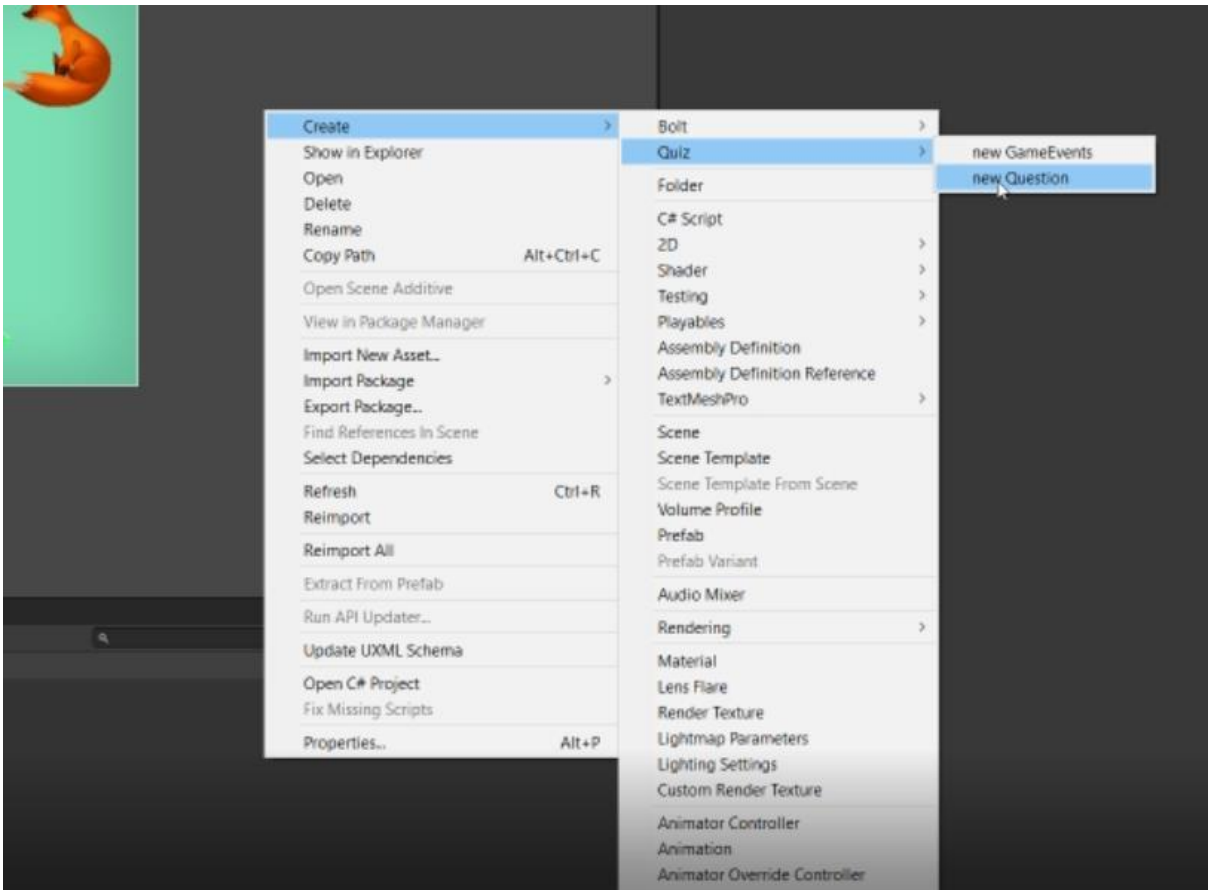


Figure 111 - script allowing Creating quiz questions from inspector

Question Script to make scriptable objects, this script allowed us to create new questions within Unity.

```
[Serializable()]
public struct Answer
{
    [SerializeField] private string _info;
    public string Info { get { return _info; } }

    [SerializeField] private bool _isCorrect;
    public bool IsCorrect { get { return _isCorrect; } }
}
[CreateAssetMenu(fileName = "New Question", menuName = "Quiz/new Question")]
public class Question : ScriptableObject {

    public enum                 AnswerType                  { Multi, Single }

    [SerializeField] private    String      _info           = String.Empty;
    public                      String      Info            { get { return _info; } }

    [SerializeField]            Answer[]    _answers        = null;
    public                      Answer[]    Answers         { get { return _answers; } }

    //Parameters

    [SerializeField] private    bool        _useTimer       = false;
    public                      bool        UseTimer        { get { return _useTimer; } }

    [SerializeField] private    int         _timer          = 0;
    public                      int         Timer           { get { return _timer; } }

    [SerializeField] private    AnswerType  _answerType     = AnswerType.Multi;
    public                      AnswerType  GetAnswerType   { get { return _answerType; } }

    [SerializeField] private    int         _addScore       = 10;
    public                      int         AddScore        { get { return _addScore; } }

    /// <summary>
    /// Function that is called to collect and return correct answers indexes.
    /// </summary>
    public List<int> GetCorrectAnswers ()
    {
```

Figure 112 - New Question Filename Script

‘Game Events’:

This just managed all the different call backs for the quiz and scripts which basically managed all the game events.

```
[CreateAssetMenu(fileName = "GameEvents", menuName = "Quiz/new GameEvents")]
public class GameEvents : ScriptableObject {

    public delegate void    UpdateQuestionUICallback            (Question question);
    public                  UpdateQuestionUICallback            UpdateQuestionUI                = null;

    public delegate void    UpdateQuestionAnswerCallback        (AnswerData pickedAnswer);
    public                  UpdateQuestionAnswerCallback        UpdateQuestionAnswer            = null;

    public delegate void    DisplayResolutionScreenCallback     (UIManager.ResolutionScreenType type, int score);
    public                  DisplayResolutionScreenCallback     DisplayResolutionScreen         = null;

    public delegate void    ScoreUpdatedCallback();
    public                  ScoreUpdatedCallback                ScoreUpdated                    = null;

    [HideInInspector]
    public                  int                                 CurrentFinalScore               = 0;
    [HideInInspector]
    public                  int                                 StartupHighscore                = 0;
}
```

Figure 113 - Game Events Script Snippet

‘Audio Manager’ object and script:

Finally, the audio manager object was created which had a script attached to it called ‘Audio Manager’. 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 in the inspector as shown below:

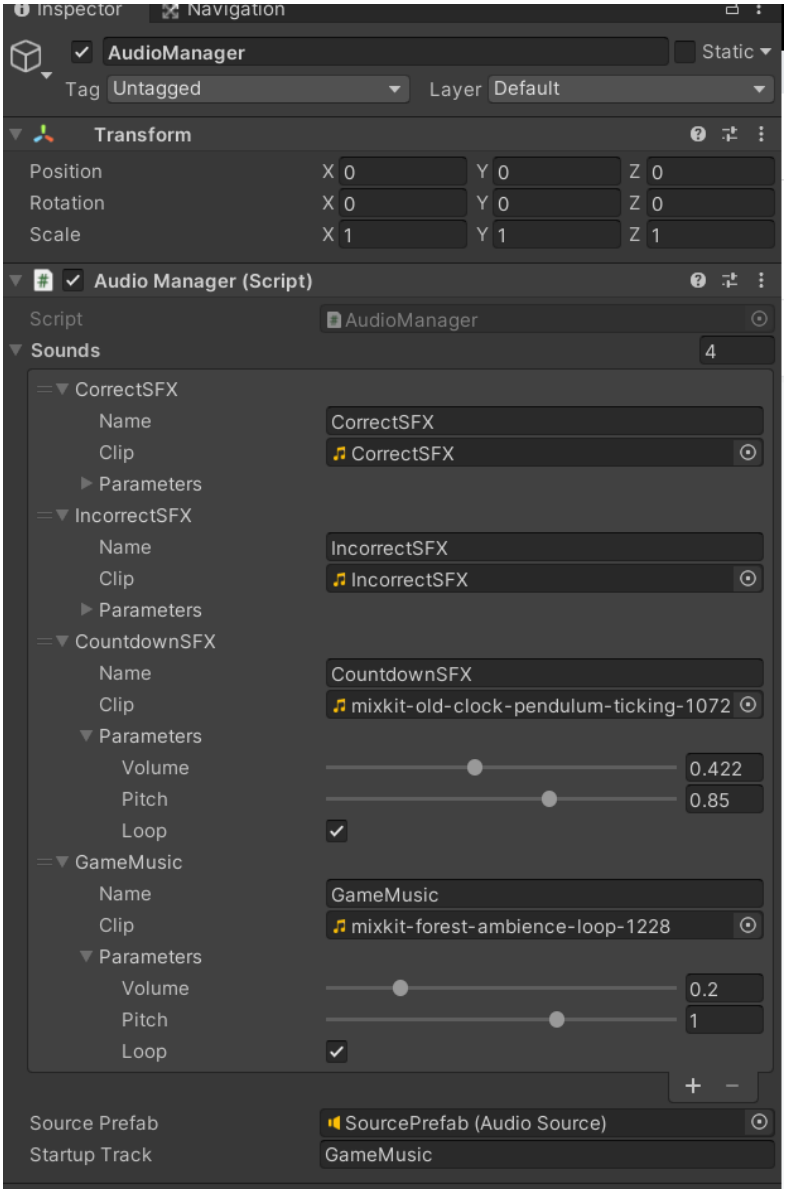


Figure 114 - Audio Manager Object

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:

```
/// <summary>
/// Function that is called to play a sound.
/// </summary>
public void PlaySound(string name)
{
    var sound = GetSound(name);
    if (sound != null)
    {
        sound.Play();
    }
    else
    {
        Debug.LogWarning("Sound by the name " + name + " is not found! Issues occured at AudioManager.PlaySound()");
    }
}
/// <summary>
/// Function that is called to stop a playing sound.
/// </summary>
public void StopSound(string name)
{
    var sound = GetSound(name);
    if (sound != null)
    {
        sound.Stop();
    }
    else
    {
        Debug.LogWarning("Sound by the name " + name + " is not found! Issues occured at AudioManager.StopSound()");
    }
}

Getters
```

Figure 115 - Audio scripts snippets

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 changed to match the wildlife the user is being tested on.
Below is an example of the Fox quiz:

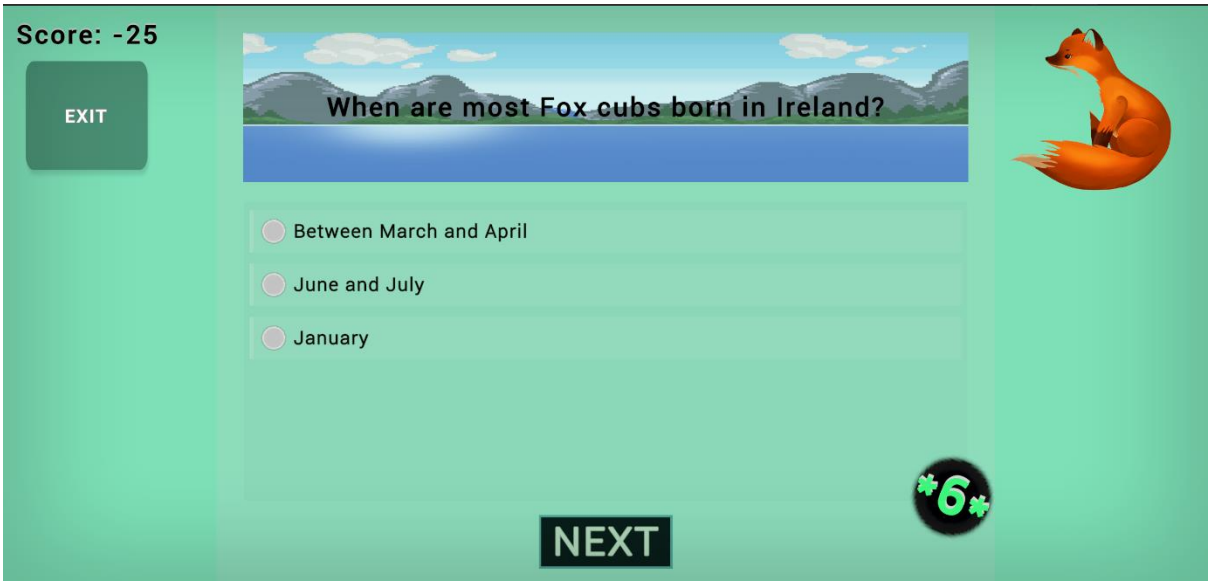


Figure 116 - Fox Quiz

Correct Screen:

Figure 117 - Correct Screen

Wrong Screen:

Figure 118 - Wrong Screen

Final Score Scene:

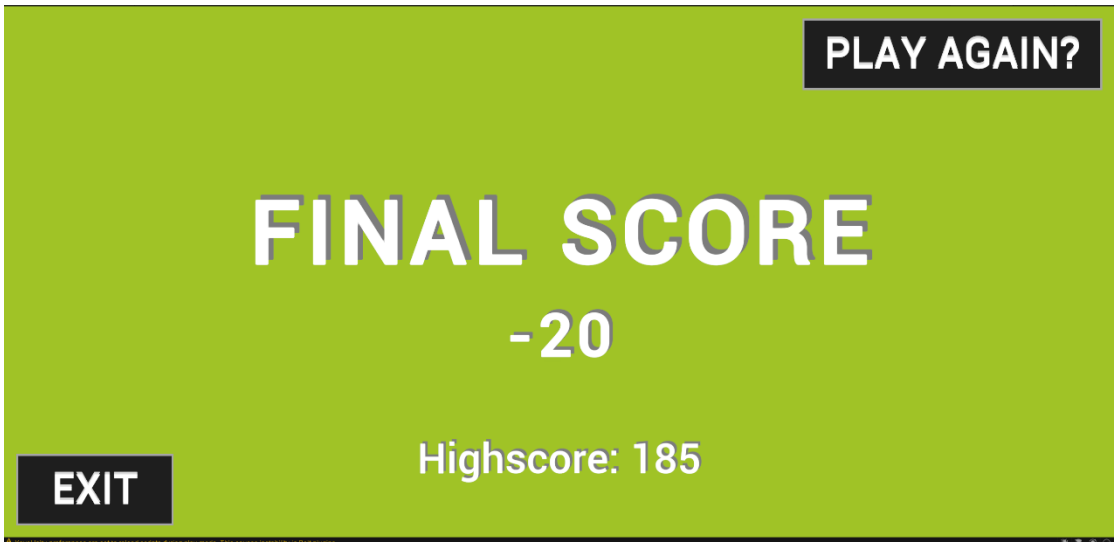


Figure 119 - Resolution Screen

4.3.5. Models

Animals

To make the animals, Blender was used to create these models. To do this a reference image was imported into Blender and the shape of the animal was drawn and turned into a mesh. Once this was done points were added to the mesh and stretched, resized, and positioned to make the animal or tree look more accurate compared to the reference image.
An example of the process of the badger animal being made is described below.

First the reference image was imported.

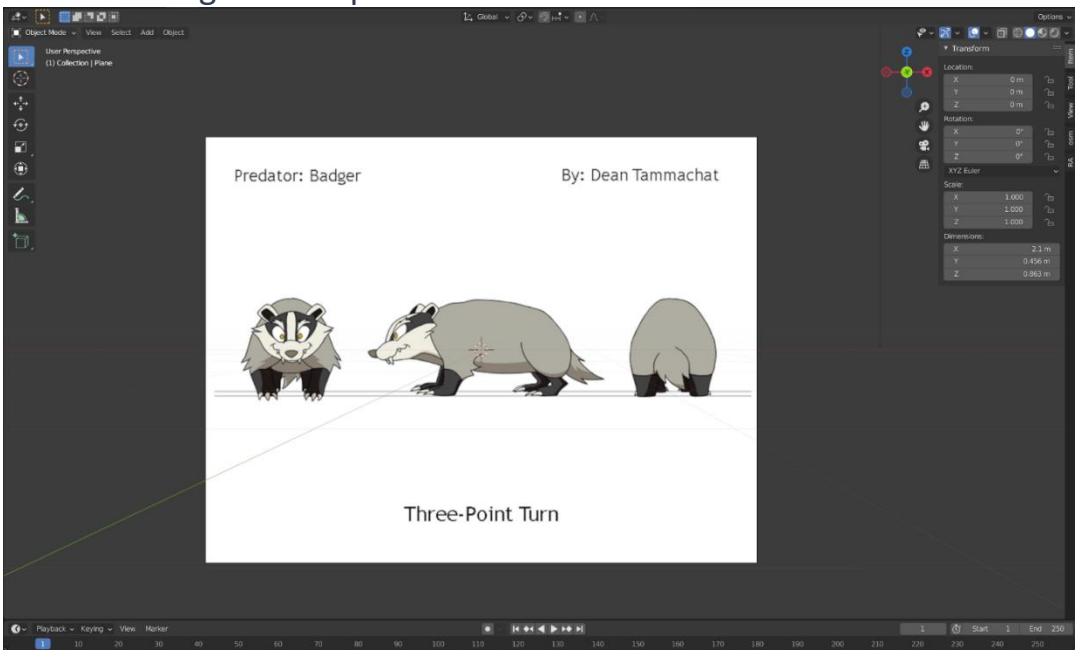


Figure 120 - Badger Reference image in Blender

Then all the points were drawn around the shape of the image to get a realistic shape. They were resized, scaled, and positioned correctly.

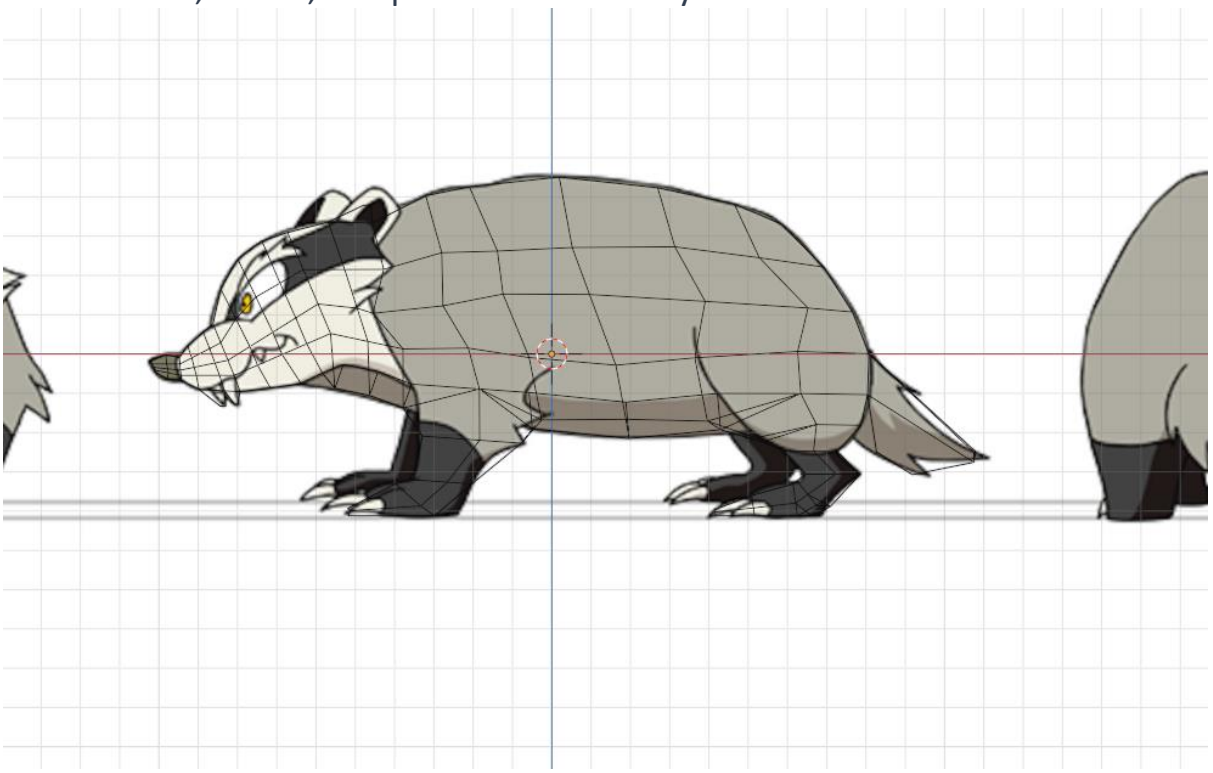


Figure 121 - Mesh drawn around shape

Then, all the points and shapes were extruded and fitted to make a 3D model of the badger.

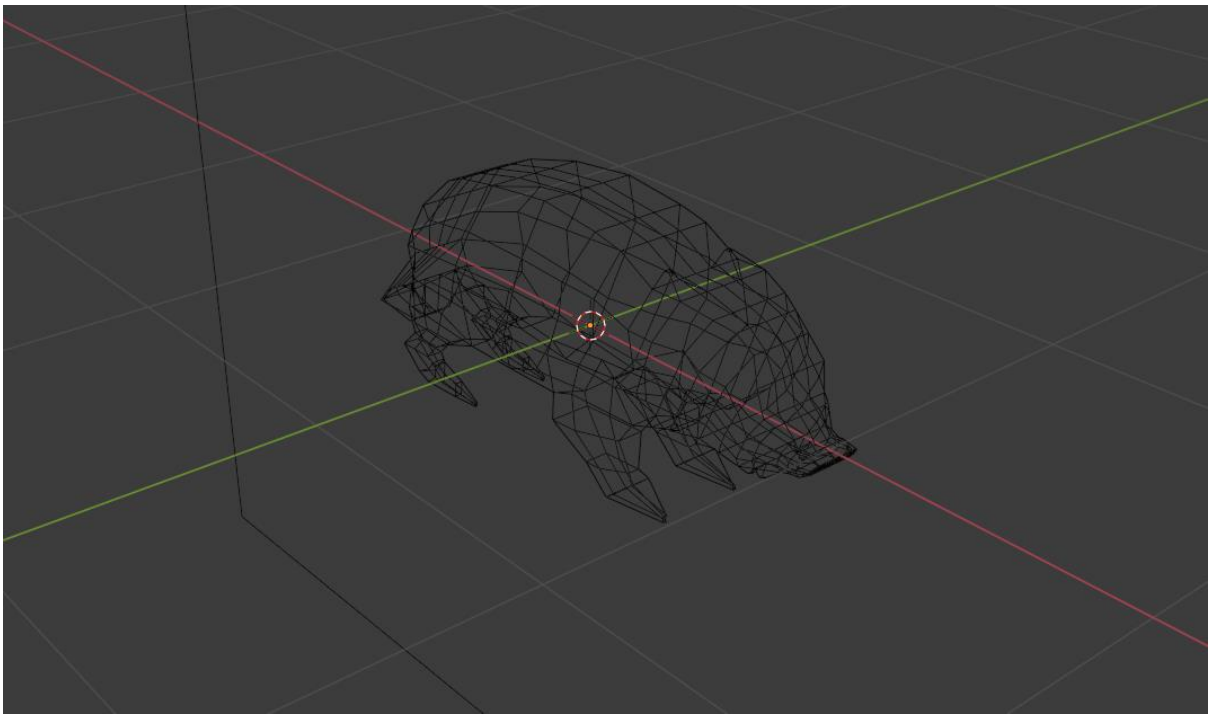


Figure 122 - 3d model extruded

Finally, the model is made into a solid object and shaded so the edges and faces are flat. The model is then exported as an FBX file to Unity.

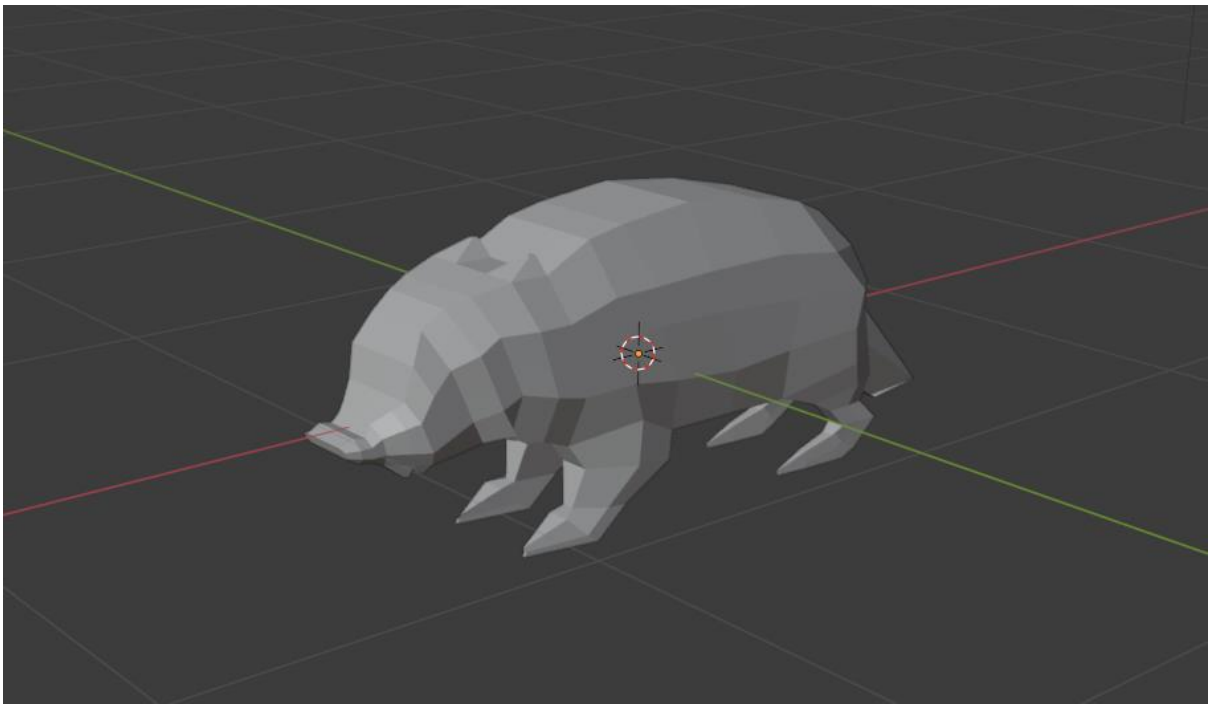


Figure 123 - Finished model

This process was done for all the animal models. In total there were five different models.

Trees

Next the tree modelling process will be described as it is a slightly different process of modelling.

When modelling the tree in blender there are basic tree model pre-sets that are built on to Blender, so it generates a basic tree model without leave and just a trunk. Once the basic model pre-set is created then branches and leaves were added to make into the desired tree. The amount, size, scale, and positioning of the branches were created, and the shape and textures of the leaves were added as well.
There were six trees in total created in blender, an example of one was the birch tree.

First the trunk of the tree was created and resized to match the profile of a birch tree. Branches were added and customized to be similar of a birch.

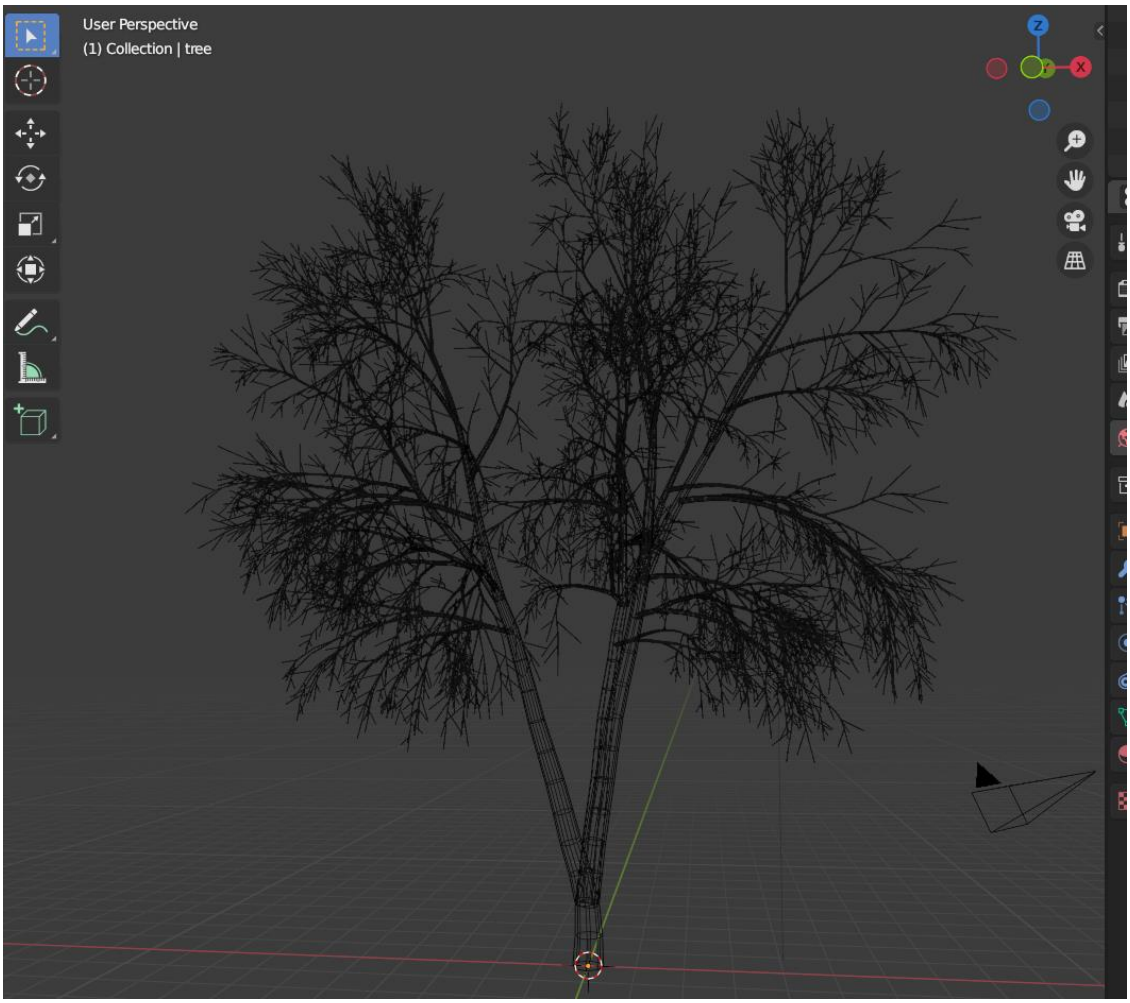


Figure 124 - Tree mesh

Next the leaves were created. a reference image of the leaves of a birch tree was imported into Blender and the shape of the leaf was drawn to match. The leaves were created and added.

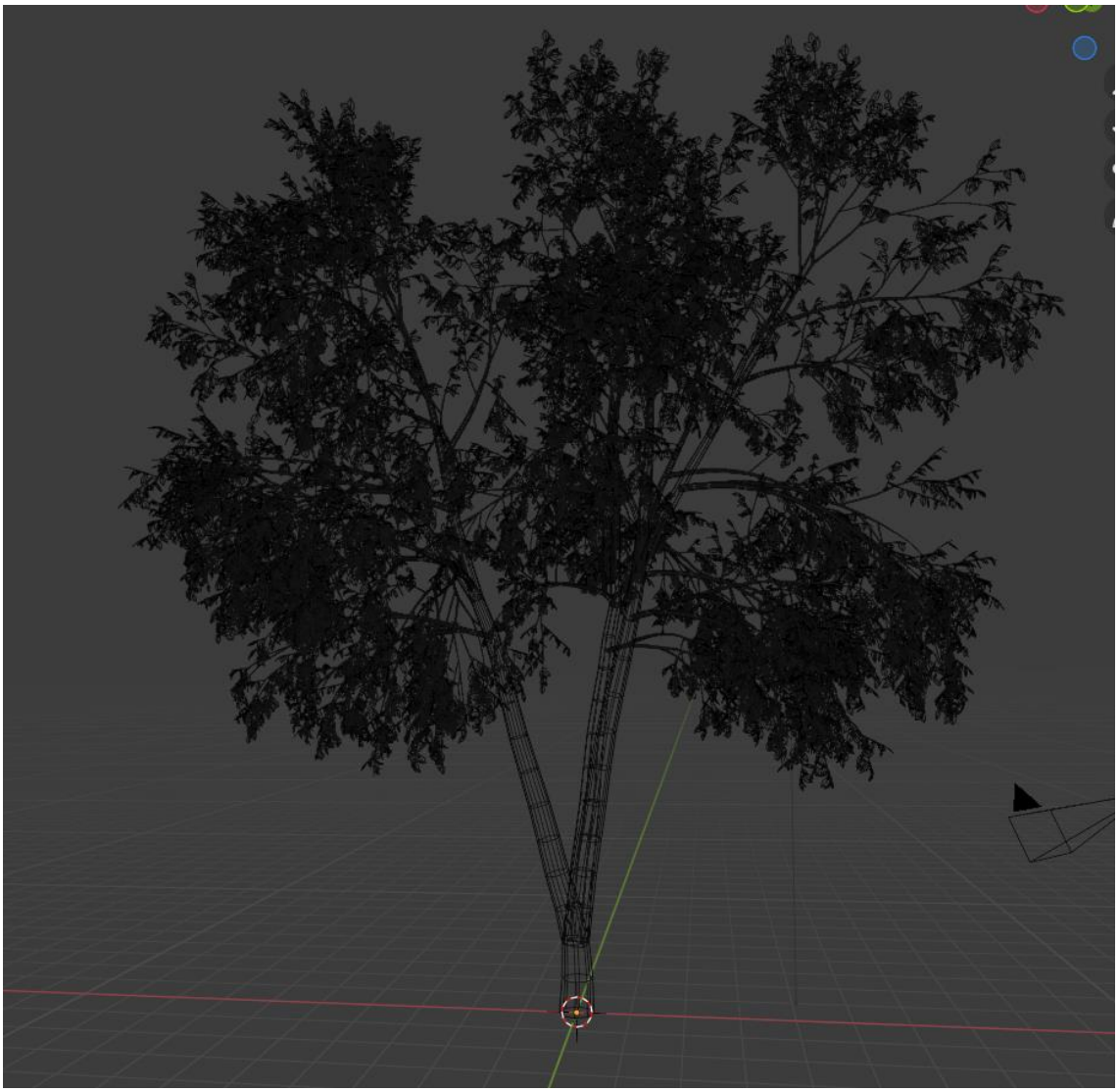


Figure 125 - Tree with leaves created

Once this was done a texture for the birch tree's bark and the leaves were created using images from google and added to the tree to create a realistic looking model of the birch tree.

Reference image for bark texture:

Figure 126 - Bark Refence image

Reference image for Leaf:

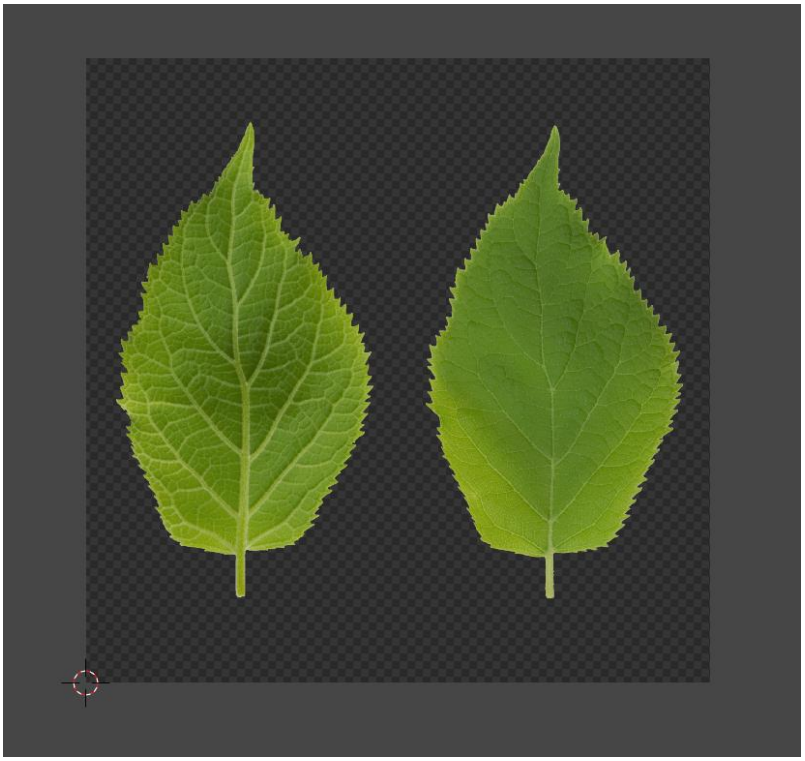


Figure 127 - Leaf Reference image

Result of Birch Tree Created:

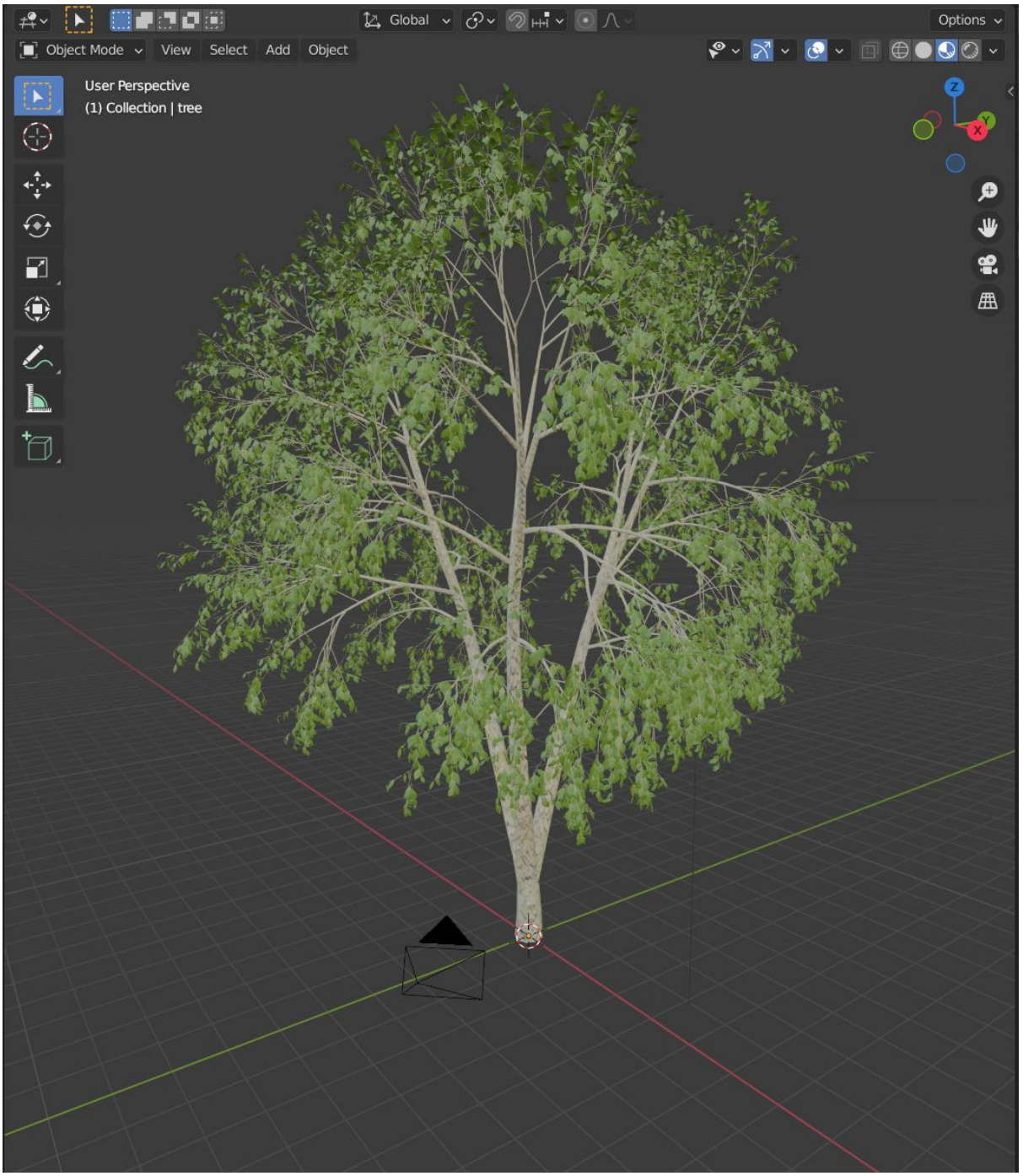


Figure 128 - Model finished with textures added

Once this tree was finished it was exported as an FBX file into Unity and used in the game. This process was repeated for all trees.

Rocks

To make the rocks that were used in the game, Blender was used. In Blender a cube was created. It was then modified to have subdivision surface, this made it look closer to a rock shape. After this, a texture was added to the model. This texture was called ‘Voroni’,

and it was then scaled and modified to look like a rocks surface. Once this was done, a model had now been created of a rock which is shown below.

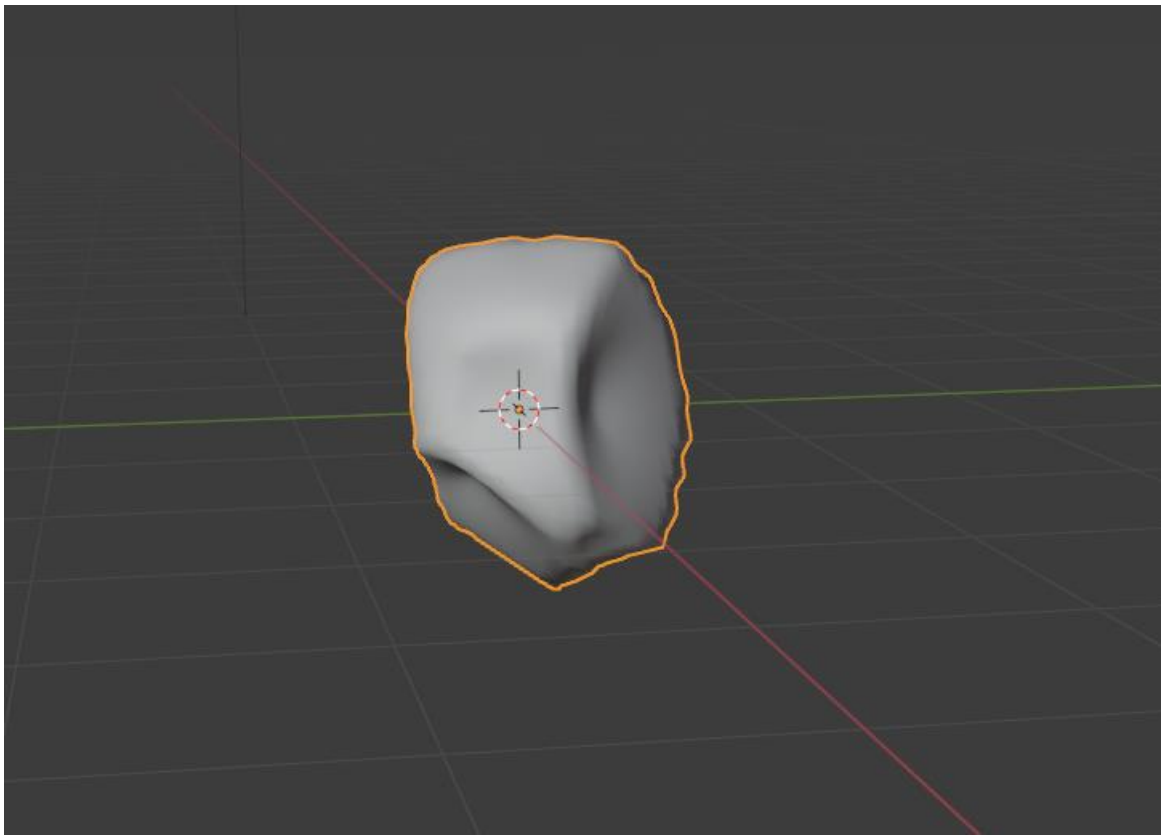


Figure 129 - Rock Model

This was exported to Unity to be later used in the game.

Grass

To create the grass, a plane was created in blender and resized and shaped to look like a blade of grass, it was made to look thin and pointy. A bend was also added. Once the blade was added, this was duplicated many times to create a clump of grass and joined together.

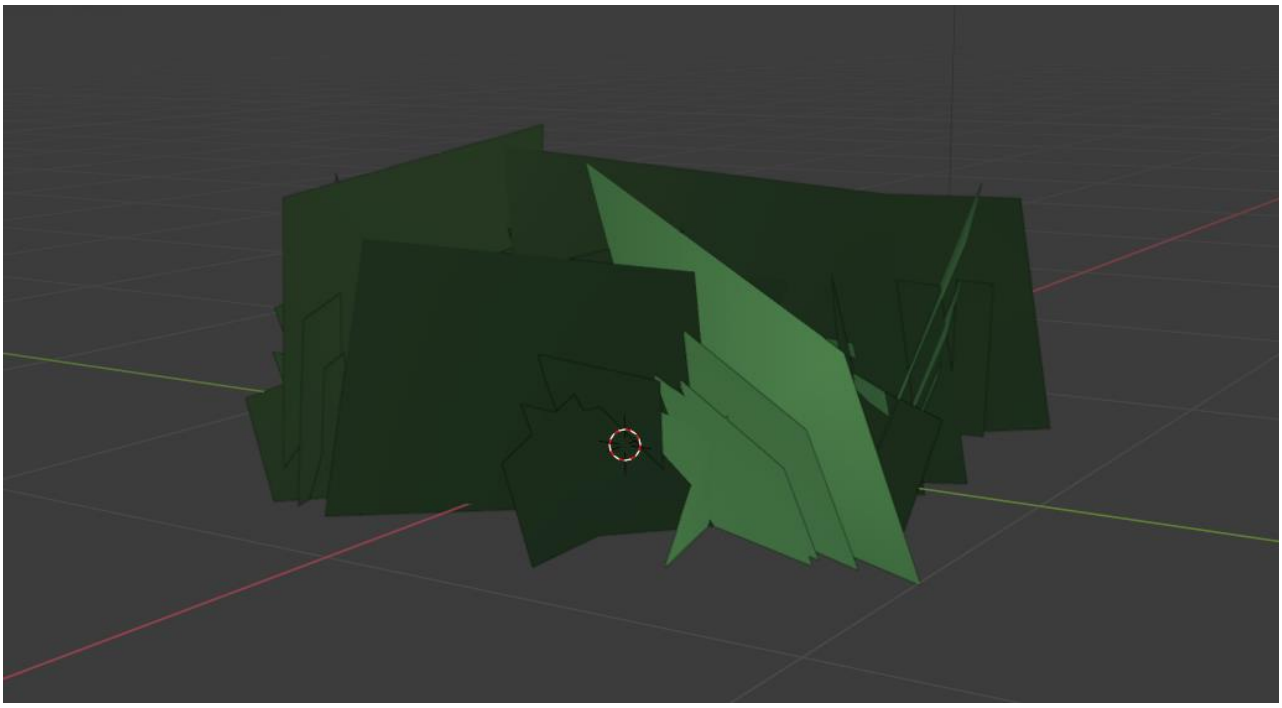


Figure 130 - Grass model in Blender

Once this grass object was created, it was exported into unity where a sway animation and a grass material were added. Below is an image of the final grass model which has the material to make it look like grass.

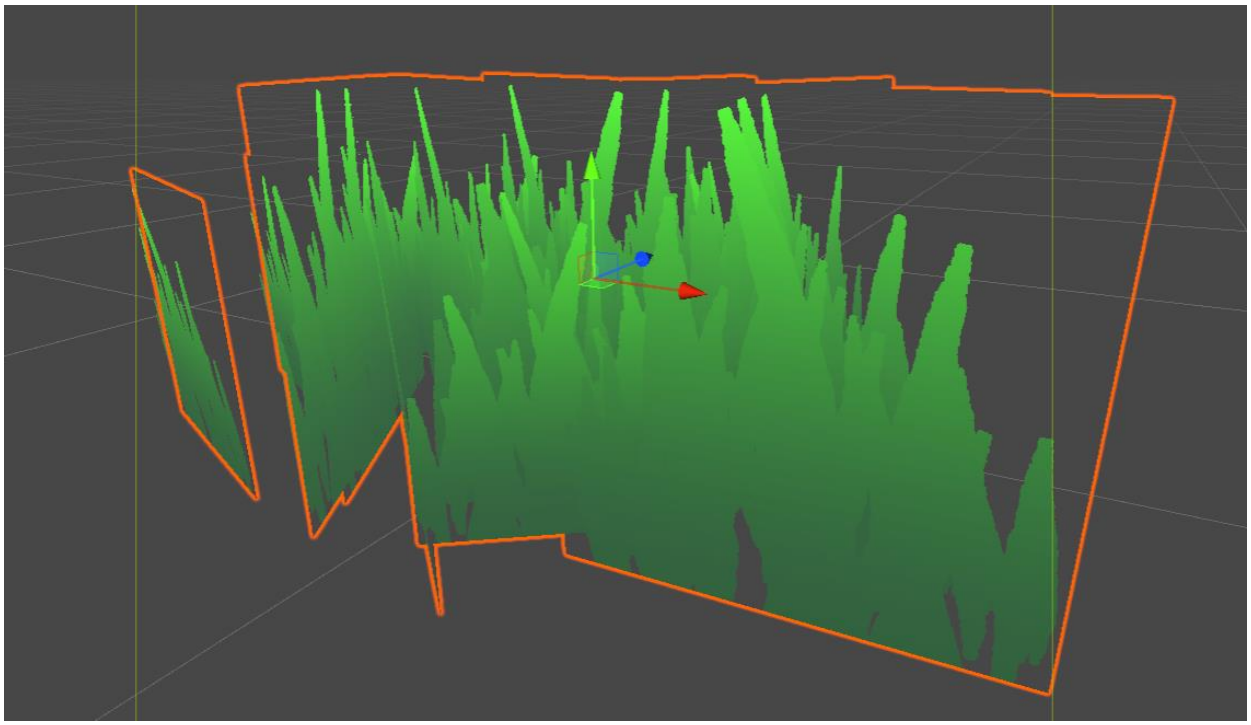


Figure 131 - Grass Finished

Info Cubes

Throughout the environment info cubes were created. These were object that allowed the user to access information about the trees. They were created in Unity. The tree models were shrunk down and attached to a cube which then used a C# script to rotate. These info cubes also used the Ray casting scripts to be inspected.

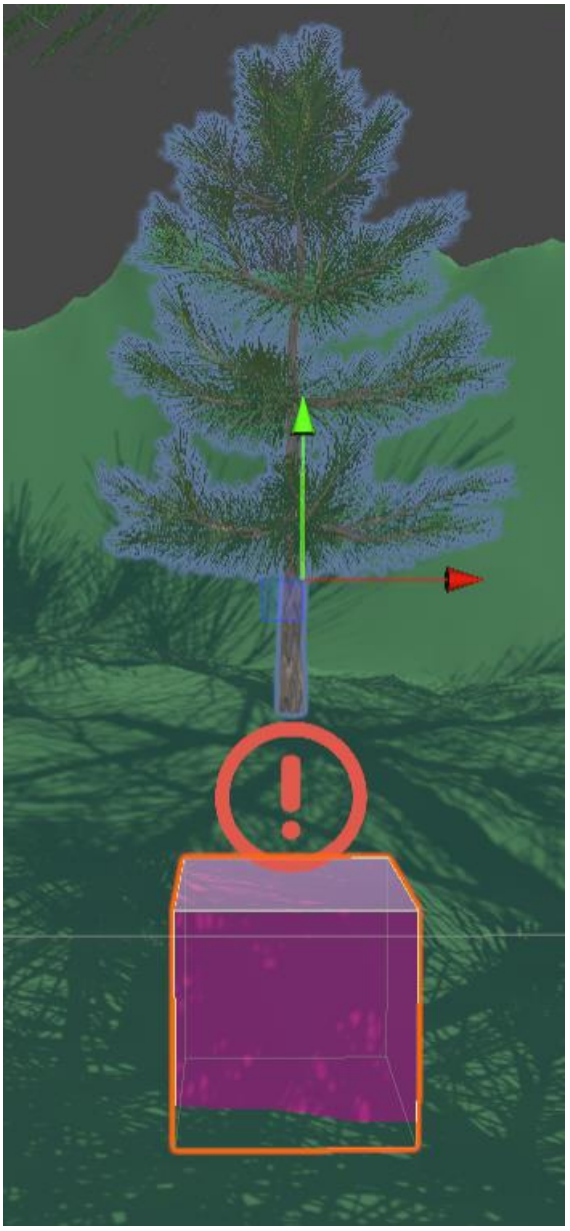


Figure 132 - Info Cube Model

4.3.6. Menu Creation

When creating the menu system for this game there were four different menus created. A start menu, an options menu, a Pause menu, and a quiz menu. Various scripts were used to navigate throughout the scenes in the game and to display loading screens.
To create these menus, canvas objects were created and UI elements such as buttons, background images, and text were added to the canvas. For example, the main menu has three buttons, one to play the game, one for the options menu, and one to leave the game.
The process in creating the start menu involved creating a new scene in Unity, placing models such as the trees, grass, and the frog model in the scene as the background.

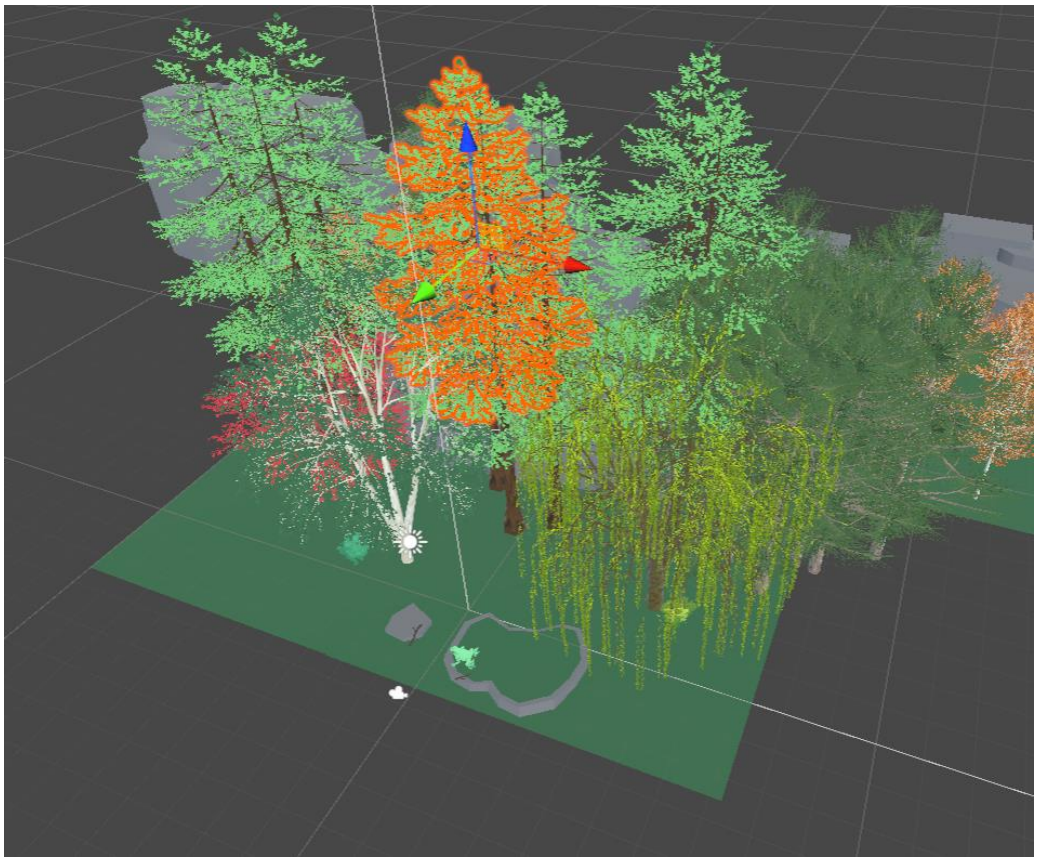


Figure 133 - Menu Scene allocation

A canvas object was then created and added to the scene. Buttons, images, and text were added to this canvas. A Play button was added which starts the game. An options button was added which brings you to the options menu, and a leave game button which quits the game.

Figure 134 - menu buttons UI creation

For these buttons to work a script was created called ‘Level Loader’ which allowed the scenes to be changed and to give the buttons functionality when clicked. This script also allowed a loading screen to be displayed when loading in and out of the game.

Below is an image of the level loader script to manage the scenes:

```
using System.Collections;
using UnityEngine.SceneManagement;
using UnityEngine;
using UnityEngine.UI;

public class LevelLoader : MonoBehaviour
{

    public GameObject loadingScreen;
    public Slider slider;
    public Text progressText;


    public void LoadLevel(int sceneIndex)
    {

        StartCoroutine(LoadAsynchronously(sceneIndex));

    }

    IEnumerator LoadAsynchronously(int sceneIndex)
    {
        AsyncOperation operation = SceneManager.LoadSceneAsync(sceneIndex);

        loadingScreen.SetActive(true);

        while (!operation.isDone)
        {
            float progress = Mathf.Clamp01(operation.progress / .9f);

            slider.value = progress;
            progressText.text = progress * 100f + "%";

            yield return null;
        }
    }
}

```

Figure 135 - Coroutine for loading level

Below is the final product of the Main menu in game:

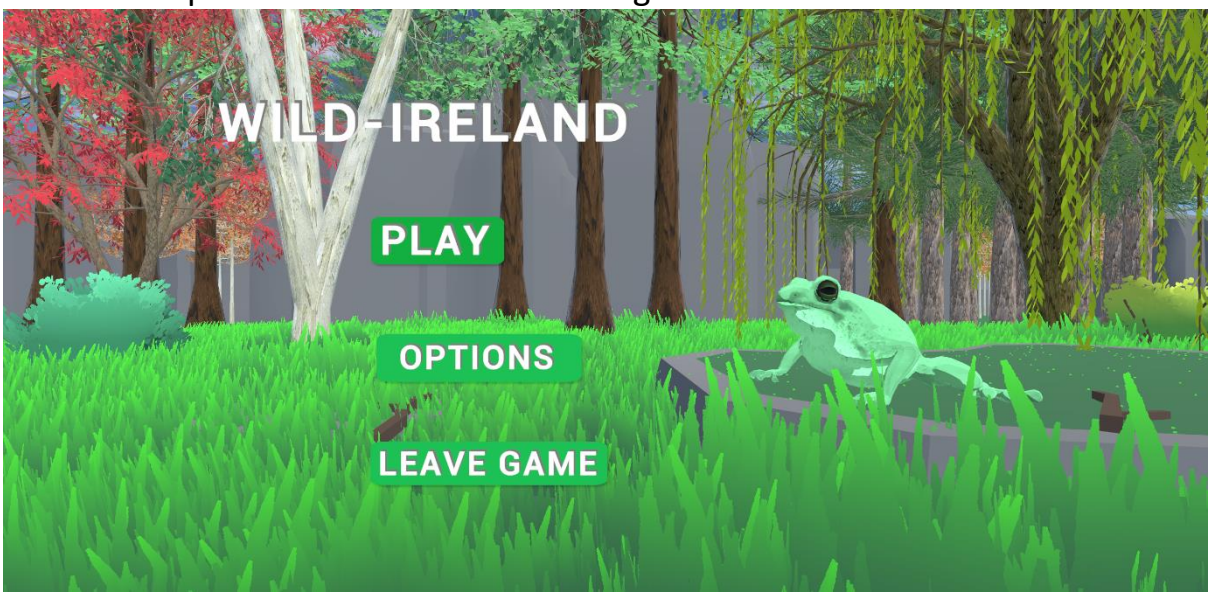


Figure 136 - Final Main Menu

The options menu followed the same process a s the main menu and so did the quiz menu

below, they used the same script and technique of creation. Shown below are images of the options and quiz menu.

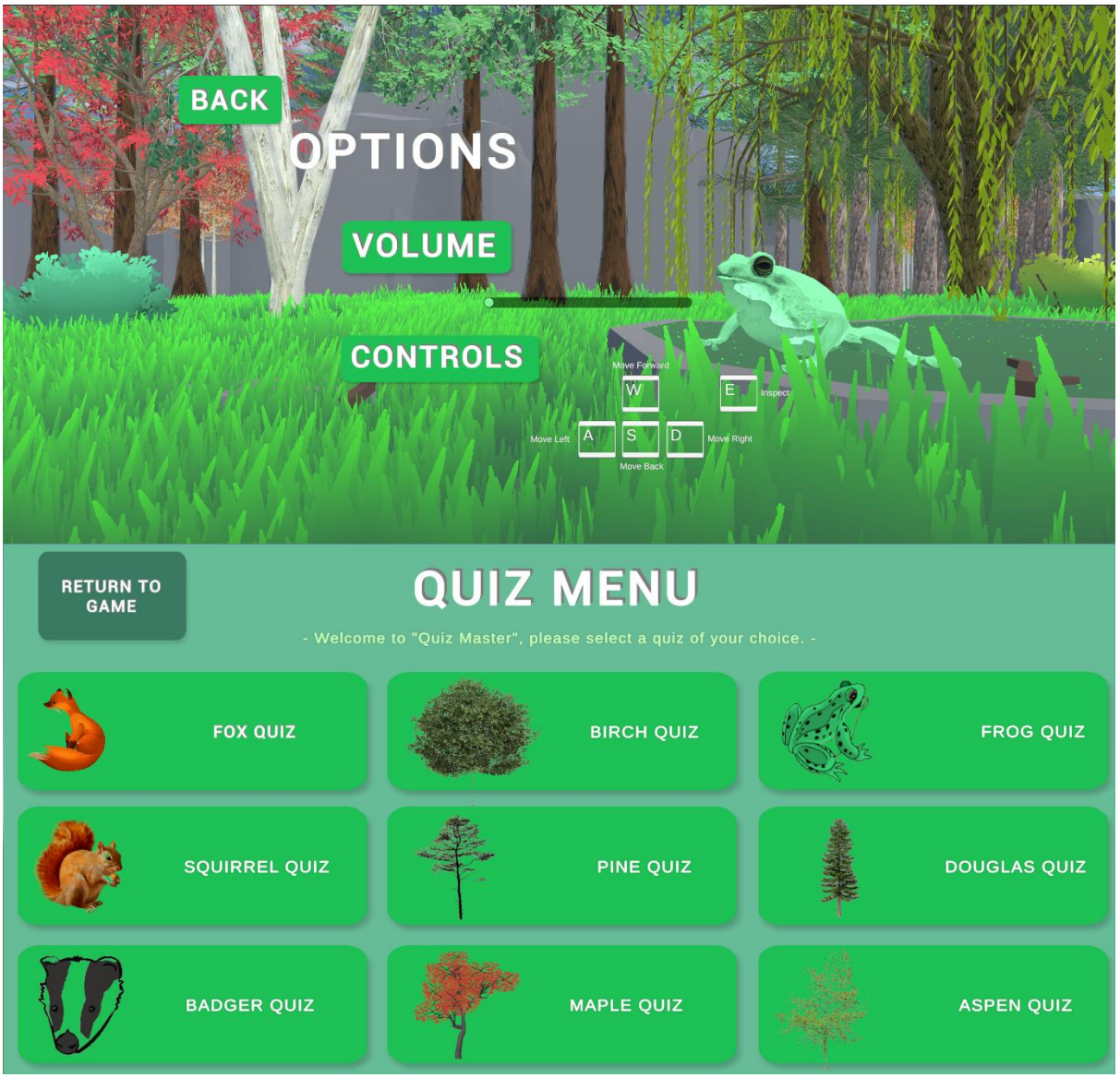


Figure 137 - Quiz menu and Options menu

When it comes to the pause menu this was created in a slightly different way as it uses a different script. While the UI elements and design followed a similar pattern as the other menus, the scripts were different.
It used a script called 'Pause Menu'. Which allowed the menu to be accessed whilst the main scene was running. When the escape button is pressed the game is frozen and the pause menu appears which allows the user to either resume, access settings or return to the main menu.

Below is the script used:

```
public class pauseMenu : MonoBehaviour
{
    public static bool GameIsPaused = false;

    public GameObject PauseMenuUI;

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.None;
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        PauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }

    void Pause()
    {
        PauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }

    public void LoadSettings()
    {
        Debug.Log("Loding settings");
    }

    public void ReturntoMenu()
    {
        SceneManager.LoadScene("Menu");
    }
}
```

Figure 138 - Pause menu script

Below is another image of the pause menu being used in game. As you can see the game is frozen and the user has three options to choose from.

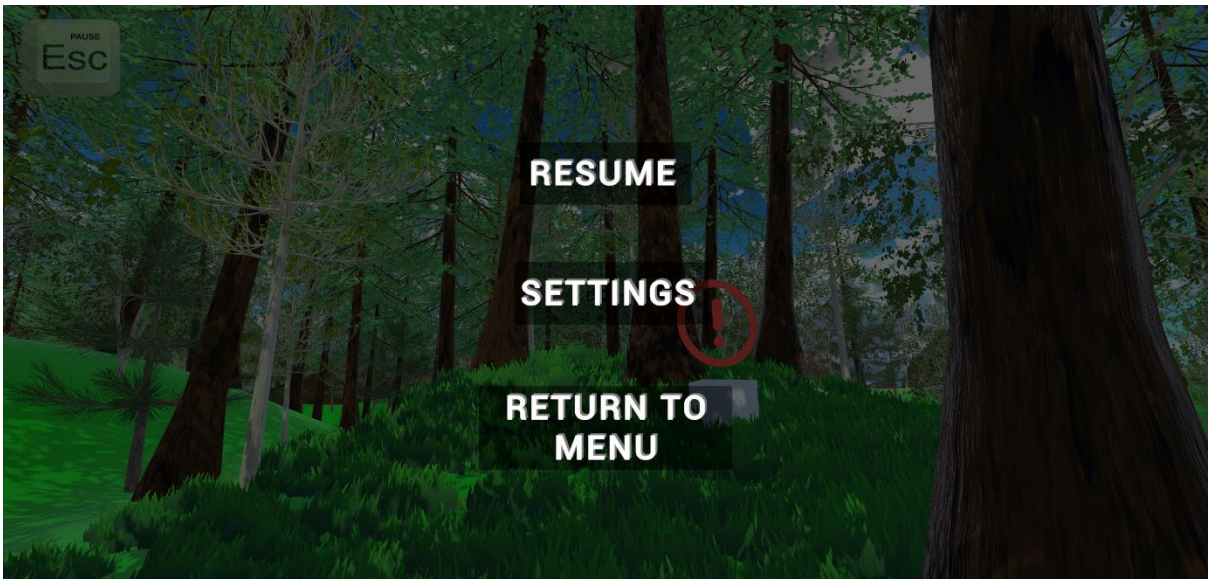


Figure 139 - Pause menu working

4.3.7. Audio

The next part that will be discussed is the Audio for the Game. In every scene, an audio game object was created so an audio source could be attached to it.
In the main menu, nature sounds were set to play on a loop. When buttons were clicked, click sounds were set to play every time they are clicked. Frog audio was attached to the frog object in the menu scene as well.

For the cutscene that plays once the user clicks play in the menu, a recording of the narrator talking was attached to an audio source object and set to play once the game is accessed from the start menu. Music was also attached to this object allowing music to play simultaneously.

In the main world scene, nature sounds, animal sounds, and sounds that play when the start menu buttons are accessed were also attached to game object to allow the audio sources to play.

As discussed previously in the quiz section, an audio manager script managed the audio of the quiz allowing sounds like correct, incorrect, music and countdown sounds to be played at the right times. The music audio used for this was also used in the quiz menu.

4.3.8. Cutscene

In this section the development of how the cutscene was made will be discussed. The cutscene consisted of a narrator introducing the user to the game once the user starts the game from the main menu. Which gives the user some information on how the game works.

Cutscene playing image:

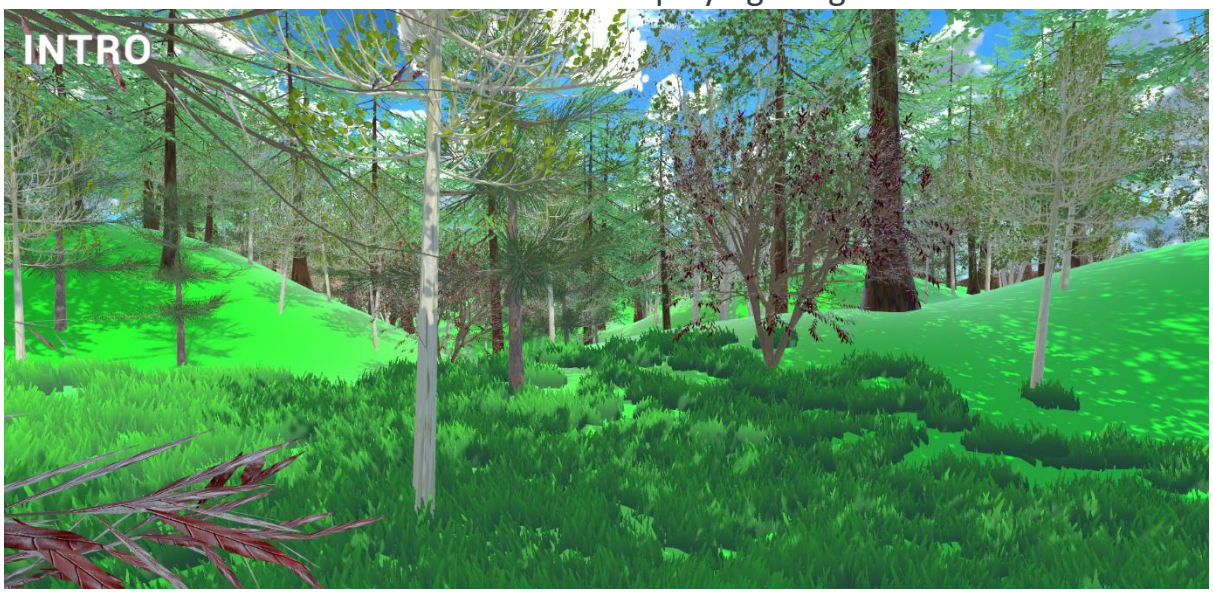


Figure 140 - Cutscene

This was done using a script written in C# and creating a series of cameras and animations to create a cinematic cutscene of the environment. A series of cameras were created, then animations of the cameras rotating and moving throughout the terrain were created in the animation window of the Unity editor. These animations were attached to the cameras. After this, the script was written and attached to a game object in the scene which allowed the script to work. The script allowed the cameras to switch between each other and play the animations at the desired times using a coroutine.

Below is a snippet of the script showing the coroutine used in the script:

```
IEnumerator TheSequence()
{
    yield return new WaitForSeconds(11);

    Cam2.SetActive(true);
    Cam1.SetActive(false);
    yield return new WaitForSeconds(5);
    Cam3.SetActive(true);
    Cam2.SetActive(false);
    yield return new WaitForSeconds(5);
    Cam5.SetActive(true);
    Cam3.SetActive(false);
    yield return new WaitForSeconds(8);
    Cam4.SetActive(true);
    Cam5.SetActive(false);
    yield return new WaitForSeconds(8);
    Cam1.SetActive(true);
    Cam4.SetActive(false);


    yield return new WaitForSeconds(17);
    Cam1.SetActive(false);
    player.SetActive(true);
    UI.SetActive(true);
    Symbol.SetActive(true);
    Title.SetActive(false);
    Music.SetActive(false);
    MiniMap.SetActive(true);

}
```

Figure 141 - Cutscene sequence

4.3.9. Mini Map

This part will describe how the mini map in the corner of the screen to show the players movement and show the map from a bird's eye view was made. To make this, a new camera object was created, and a short script was attached to this object which allowed the camera to display at the same time as the main camera but this time it was displayed on top of an image in the right-hand corner of the screen, giving a bird's eye view of the player and the map around it.

Below is the mini map that was created:

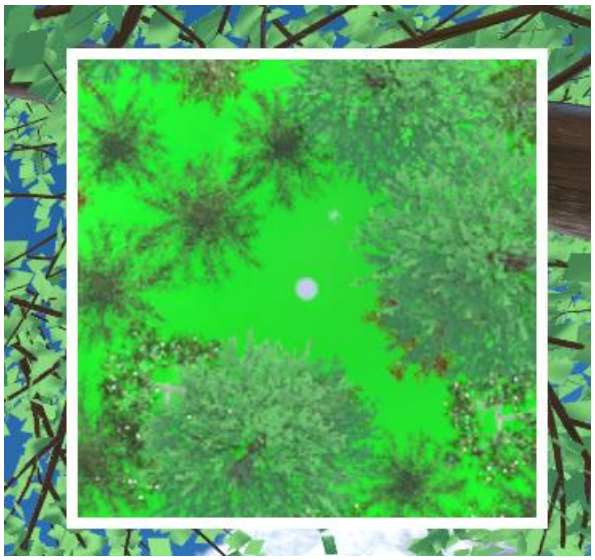


Figure 142 - Mini Map

4.3.10. UI

All the UI elements that have not already been discussed will be talked about in this section, including the loading screen and the HUD elements. In Unity, most UI elements are created by creating a canvas object and then adding elements like buttons, backgrounds, and text as discussed in the design section.

The loading screen was implemented and used to be put in between loading into scenes in the game. This was created by adding a progress bar to a canvas object and then creating an object with a script attached to it called ‘level loader’ that allowed the loading screen to be played every time the user had to wait for the next scene to be loaded, and to allow the progress bar to show a percentage of how much had been loaded.

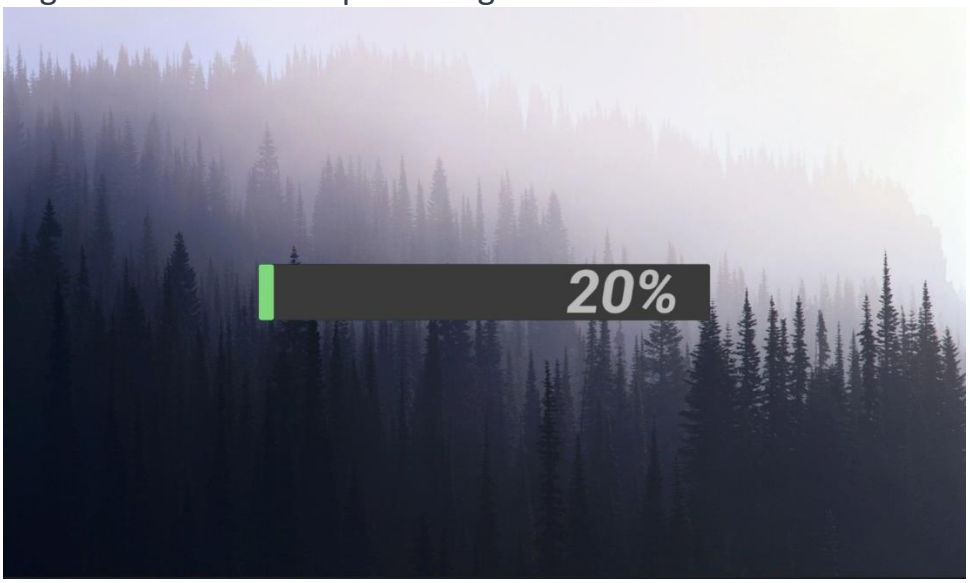


Figure 143 - Loading Screen

4.3.11. Rigging and Animation

Once the models had been created, it was time to rig and animate the models, this was done using Blender. As there are a few different animals, only the process of one of the animals being rigged and animated will be discussed as they all follow the same process. The only difference is the model is different. The rigging and animation of the badger animal will be shown.

Rig

First the rig was created. To do this each bone must be created for the skeleton which is also known as the armature. Each bone was created and attached to each other to create the armature. These bones were then put up against the model of the badger and resize, scaled, and move to the same shape as the badger and to fit inside it to simulate a real animal skeleton. Once this was done and the armature was complete, it was made to be a child of the model so that the model and the armature were all in one. The next step is to generate the rig. In Blender this is simple if you have the armature and the model created and are set as one object. All you must do is click 'Generate rig' and the rig will be created. This allows the limbs to be moved around as if a real skeleton was inside the animal's body.

Below is an image of the process of creating the armature and rigging the animal model of the badger. It also shows the limbs moving with the skeleton.

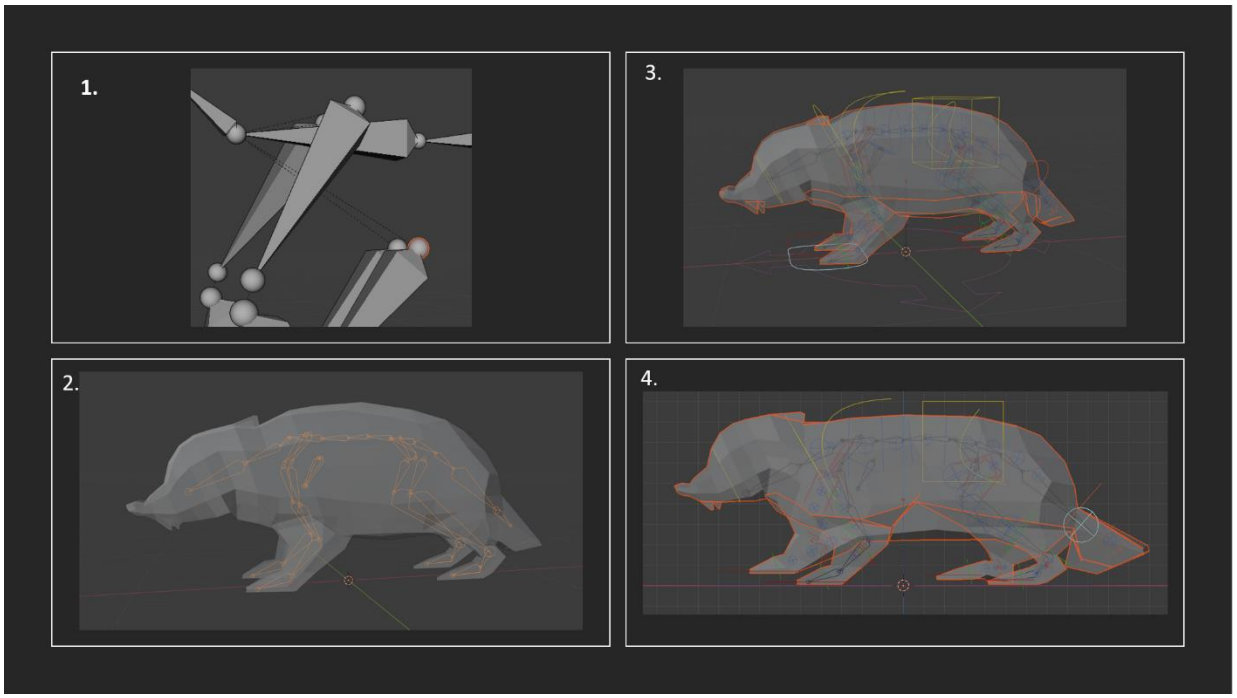


Figure 144 - Badger being rigged and Armature created

Animation

The next part was creating the animations for the animals. Again, this was done using Blender.
When animating the animals in Blender, the models were animated using keyframe animation. This consisted of the animations being recorded and the rigs which were attached to the models being moved and set in different positions to capture the different frames to produce animations like walking.
Below is an image of the walking animation of the fox being captured in the action editor using keyframe animation.

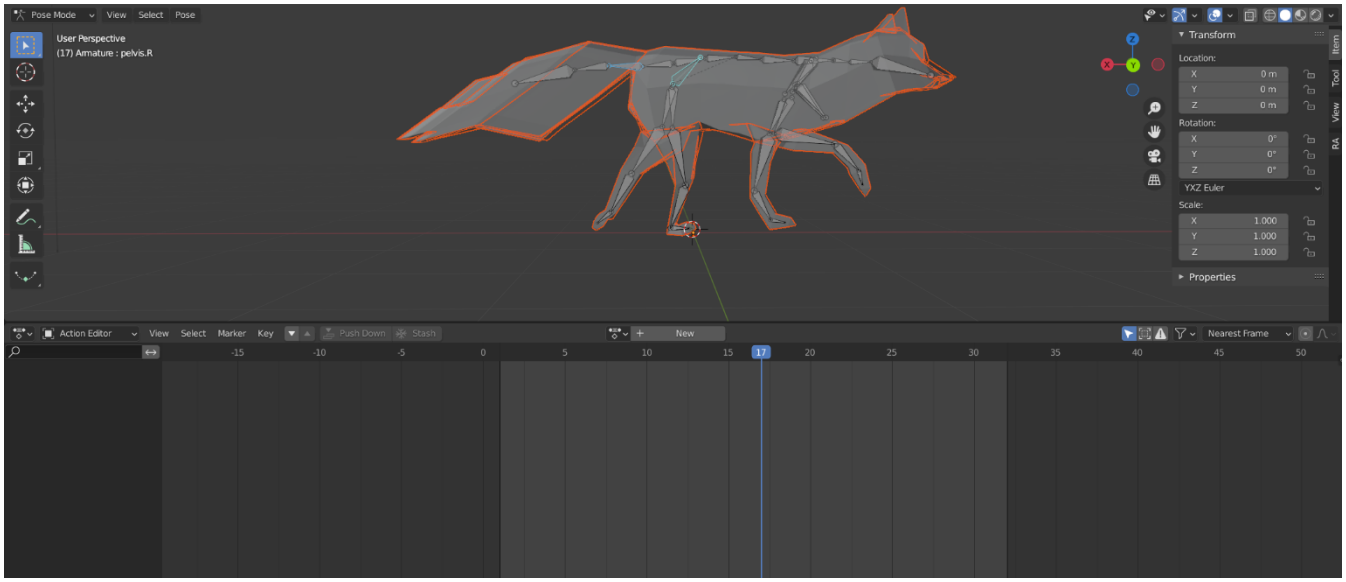


Figure 145 - fox animated using keyframes

Once the animation was captured and recorded it had to be saved in Blender and then exported with the model to Unity as an FBX file. When the model was imported into unity along with the animations, an animator controller was created in unity and attached to the animal models. The animation was then placed inside the animator component which then allowed the animal to be animated when walking.

Animator component with animation attached:

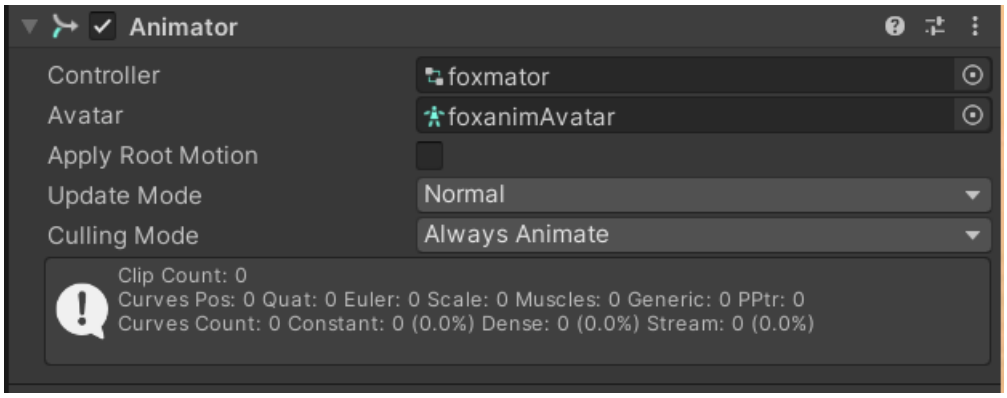


Figure 146 - Fox animation assigned to fox

4.3.12. Player Movement

To allow the player to move and look around, player movement needed to be created and attached to the player.

To do this a player object was created with a camera attached to it. Script written in C# was then attached to the player to allow the player to move around using the keys on their keyboard. For example, if the user pressed the ‘w’ key the scripts would tell the player that it needs to move forward.

A camera movement script was attached to the camera to allow the user to be able to look around using their mouse. One the player object was created, and these scripts were attached to the camera and player, the player was able to move forward, back side to side, jump, and look around in any direction.

Image of the player inspector showing the player movement script attached:

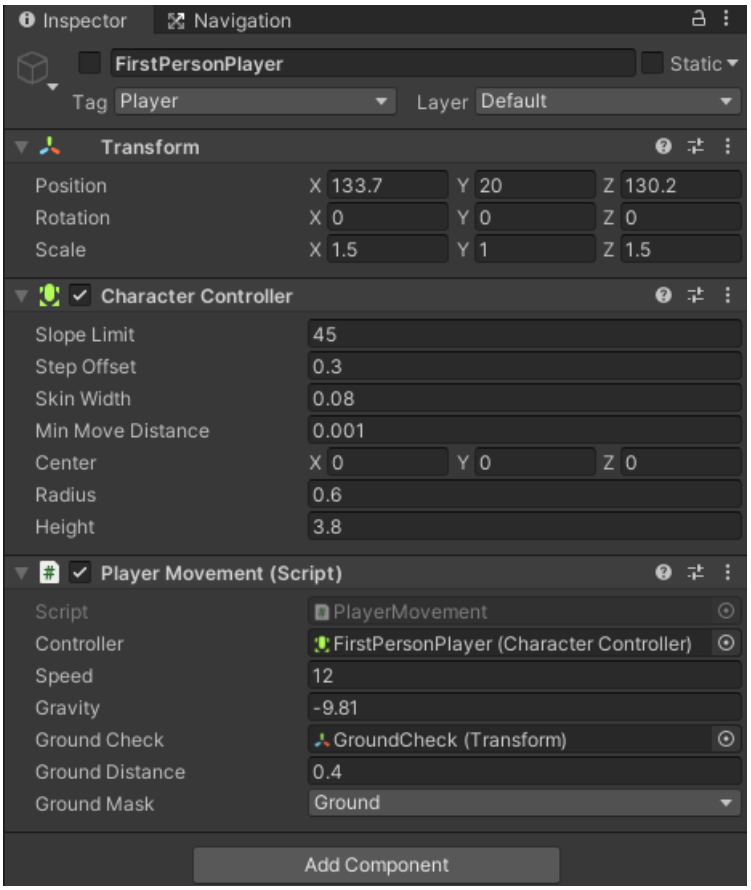


Figure 147 - Player Object with script and controller attached

4.3.13. Release

When the project was finished the project needed to be built so it could be distributed to users to gather feedback and to turn the project into an actual working game. To do this, the project was built from within Unity. The settings were adjusted to the right specifications, and it was set to run on PC standalone, Windows, Mac, and Linux. Once the game was built it produced a file which is used to run the game. This file was distributed to 3rd party users for testing.

For the game to be run you must go inside the project folder and click into the ‘game builds’ folder. From there, the ‘Wild-Ireland.exe’ file should be run.

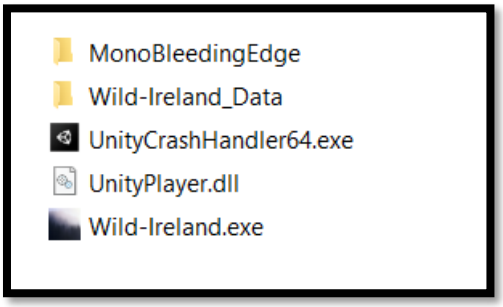


Figure 148 - Build File

4.4. Conclusions

In this section the first prototype of the main menu and the options menu were created in unity. A script was written in C# to allow interaction between scenes, screens, and buttons. Models were created in blender of trees and exported into unity to be placed throughout a scene. A frog model was added to the scene. The two screens are displayed at the end.

An in-depth discussion pf the system development was discussed. All aspect of the system were shown including procedural generation, AI, UI, quiz, modelling, animation, and all the important and features created in this system.

5. Testing and Evaluation

5.1.Introduction

This section will outline the testing methods used in this project during its development. GitHub Version control will be discussed along with a White and Black Box testing plan. Tests groups will be made for users to try out the system and find errors. The final part of this section will discuss the evaluation of the project, outlining the issues and feedback.

5.2. Testing

For the duration of this project, tests were run frequently. This was done through Unity to make sure the game is functioning properly. As discussed in the Design section, the use of GitHub helped with testing so that if there were errors with the system, having backups would be useful so that the system can be returned to its original state before the errors were encountered.
White box testing and black box testing were performed as discussed below. However, these were not the only ways of testing. There were also other methods of testing in Unity, in scripts, the models, and AI testing.

White Box Testing

White box testing allows the tester to examine the structure, design, and coding of a chosen system. This method was used to improve the overall usability experience for the user and increase the design strength, whilst finding smaller errors throughout the application as well.

This project involved a lot of trial and error regarding the development process. Each feature was tested until they were suitable to be shown to a third-party user such as someone outside the project or a testing group. When errors arose, they were recorded and solved. (25)

Next a test plan table was created to be used as a checklist for error checking to see if the tests passed or failed.

Test Plan Table

Test No.	Test Description	Expected Outcome	Pass? -
1	Does the game successfully download?	The game will download to the user's computer.	Yes
2	Does the game launch when selected?	The game will start successfully.	Yes
3	Does the loading screen appear?	When game is launch or loading terrain loading screen will appear	Yes
4	Does the Game take long to load?	Will take less than a minute?	Yes
5	Does the Start menu appear?	Menu appears.	Yes
6	Is the music playing?	Music playing on loop.	Yes
7	Does the settings button work?	Bring you to options.	Yes
8	Can you adjust the volume?	Volume slider adjusts game volume.	No
9	Can you View the controls?	Controls are visible to user.	Yes
10	Does the back button work?	Back button brings you back	Yes
		to previous screen.	Yes
11	Does the quit button work?	Quit button exits the application.	Yes
12	Does the start game button work	Play button loads in the terrain.	Yes
13	Does the game load quickly?	Should take less than 1 minute.	No
14	Does the Game work when loaded?	User able to navigate through menus and move throughout terrain.	Yes
15	Is the User allowed to maneuver?	User can use keyboard to	Yes

		maneuver throughout terrain.	
16	Does the Controls match the default?	The controls match the controls in the options menu.	Yes
17	Are the objects spawned in?	The terrain, forest and animals are spawned in.	Yes
18	Does the game look as intend?	The game looks as is intended. Lighting is correct.	Yes
19	Can you interact with animals?	animals react to the environment and player	Yes
20	Can you interact with wildlife?	The wildlife approaches or runs away from player.	Yes
21	Does the AI react to user?	Certain species approach player, certain species	Yes
		don't, some run away.	Yes
22	Does the AI work?	The AI performs rules set in scripts.	Yes
23	Is the terrain Generated?	The terrain is generated, and the user can walk on it.	Yes
24	Is The environment generated?	Trees and grass are generated.	Yes
25	Does the inventory work?	Inventory is functional.	No
26	Does the inventory work when animal added?	Animal is shown in inventory.	No
27	Does the Quiz work?	Quiz is functional.	Yes

28	Do all the Action buttons work?	Buttons in quiz create an action.	Yes
29	Does the settings menu work?	Options menu is functional.	Yes
30	Do the buttons in the menu Allow you to adjust volume, view controls and quit?	Functional.	No
31	Does the save feature work?	The user can save their progress. They can also load their progress.	No
32	Does the Resume button work?	The user can resume game.	Yes
33	Are there any glitches?	No.	Yes
34	Are there any performance issues?	No.	Yes
35	Is the game running slow?	No. depends on user's computer.	Yes
36	Are the error messages displayed?	Yes.	No
37	Does the information display in information screen	Correct information is displayed.	Yes
38	Does the video Play in inventory?	Yes, video plays.	No
39	Can you exit the inventory?	Pressing exit exits the inventory.	No
40	Does the inspect prompt show for wildlife?	Prompt appears when near wildlife. Or looking at it.	Yes
41	Is the in-game music playing?	Music is playing in background.	Yes
42	Is the sound effect/Audio working?	Sound effects functional.	Yes
43	Are all animations being executed?	Animals animated successfully.	No

Black Box Testing

Black-box testing allows for scenarios or use cases to be tested where the system or software has errors. An example of these scenario errors would be a user inputting something like a username with the wrong syntax and the application won't output an error message. This will be classified as an error for black box testing. An example in this project would be if the user tried to 92 inspect an animal and the wrong animal is shown in the user's notebook, or the animal isn't shown at all.

For this system, user testing was the most efficient way to outline the potential faults in the game. Group testing was organized where possible end users used the application and looked for potential errors. Any issues identified in group testing and testing were recorded either on a computer or by video if the test was conducted via an online call. Forms or surveys were created and used to record feedback as well. The final testing phase will consist of a beta version of the game made available to play and be tested. This will identify the final error so that the finishing touches can be applied. (25)

The main testing phase of this application consisted of a beta version of the game being sent to a variety of people such as friends, family, and online forums such as reddit and discord. This allowed the game to be black box tested and allowed the test table to be filled in which brought to light any bugs or errors in the system.

Bugs and Recommendations Table:

Bug Type	Answer	Version	Fixed
Recommendation	Weather System	Beta	No
Bug / Error	Mountains generated in terrain sometimes	Beta	No
Bug / Error	Grass didn't render properly	Alpha	Yes
Bug / Error	Menu items overlapping for quiz feature.	Beta	Yes
Bug / Error	Lighting issue when switching scenes	Beta	Yes
Bug / Error	No sound for animals	Beta	Yes
Bug / Error	Trees sometimes generated inside other objects.	Beta	Yes
Bug / Error	Cutscene played every time started game.	Alpha	Yes
Bug / Error	Frog jumped at weird angles sometimes.	Beta	Yes
Recommendation	Add Loading Screens.	Alpha	Yes
Bug / Error	Countdown audio played even though quiz wasn't being used.	Beta	Yes
Bug / Error	Badger generated at strange heights sometimes.	Beta	Yes
Bug / Error	Sometime animals spawned to many of one type.	Beta	Yes
Bug / Error	Trees had no Colliders.	Beta	No
Bug / Error	No animations on most animals.	Beta	Yes
Recommendation	Sky didn't match the terrain.	Beta	Yes
Recommendation	More behaviors need to be added.	Beta	Yes
Bug / Error	Fell through the map sometimes.	Beta	Yes
Recommendation	No footsteps for player.	Beta	Yes
Bug / Error	Player couldn't jump.	Beta	Yes

Bug / Error	Lighting issues on some of the rocks.	Alpha	Yes
Bug / Error	Mountains blocked out the light sometimes.	Beta	Yes
Bug / Error	Player didn't spawn back in the same place.	Beta	No

Once this was done and the feedback was received, changes were made to the system to fix any bugs and errors, and to also take into consideration any feedback given to make the system better. Finally, when this was complete, the game was finished, and evaluation forms were created and sent out to 3rd party users to evaluate the system.

Unity Testing

As the project got bigger, there was a lot more classes, scripts, and objects in Unity. This made it more difficult to change parts of the project without getting an error. To bypass this issue, automated testing was used to help ensure all the parts of the project were working as expected by using the Unity Test Framework package. This saved time as it didn't rely on manual testing, and it enables you to edit and test your project in both play mode and edit mode. (26)

Scripts testing

When testing code and scripts, using a debugger enables you to inspect the source code whilst the game or application is running. This was accomplished by using visual studio as a debugger to debug and check for errors in the C# code.

Model Testing

When testing the models, they needed to be tested for accuracy against real life wildlife. To do this, pictures were examined and compared to the models being created. This involved a lot of trial and error. The models were be shown to third party users in the final testing stages to make sure they looked accurate.

AI Testing

When testing the AI, the game was played by both the developer and the third-party users in a variety of different scenarios. This meant that there was a lot of trial and error to get the AI to work without errors. For example, the user would've had to approach an animal AI Agent multiple times to see how it reacted to check for errors or to check for the right response.

5.3. Evaluation

In this project the evaluation was high priority, this is because the user experience when using this system is extremely important. The application should be easy to use, and not require the user to be wondering how to use it or not know what the application is about. When this system is evaluated by a variety of end-users, the developer and possible industry professionals, it will mean that the system has been evaluated in many ways so the standard of usability will be at its strongest. This section will outline what approach was taken to evaluate the system and the metrics used as well.

For the evaluation, a feedback form was created using google forms. This provided a lot of helpful and interesting feedback for the project. In this part of the evaluation, graphs to represent the data, answers and the feedback received from the form will discussed and shown.

The feedback form provided lots of valuable information, the first question that was asked is shown below:

Figure 149 - 1st question

Once the forms were submitted the responses were looked at and the graphs were examined. The graph below represents what percentage of people who rated the game at a certain rating. Most people seemed to give it a high rating.

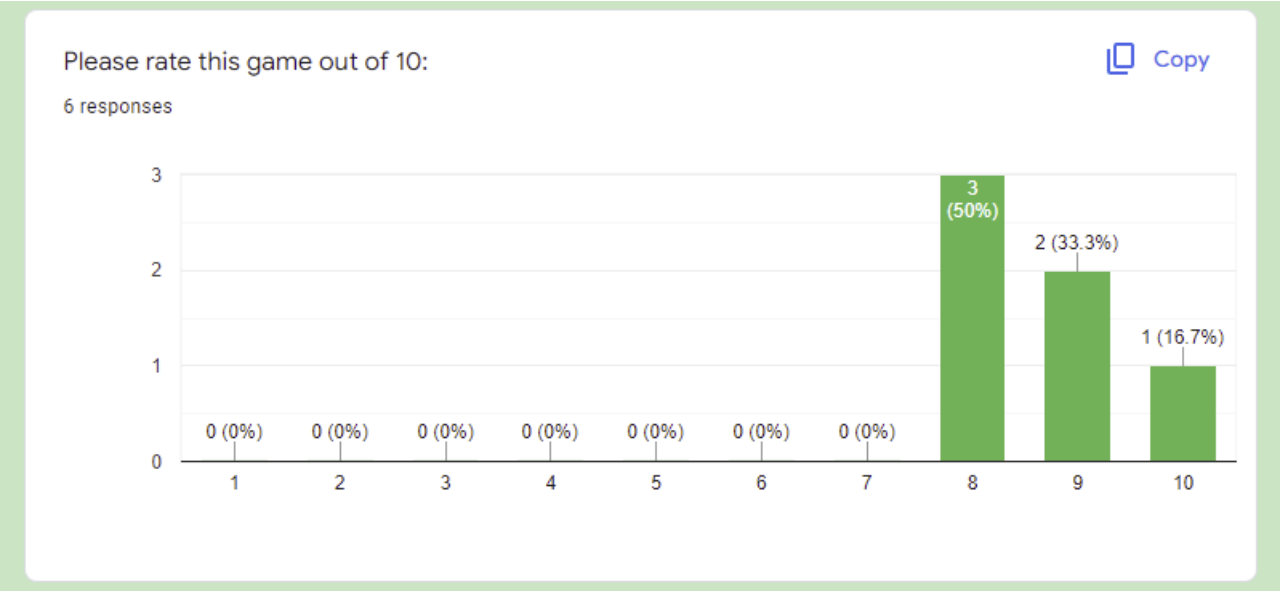


Figure 150 – 1st question and feedback

Question 2:

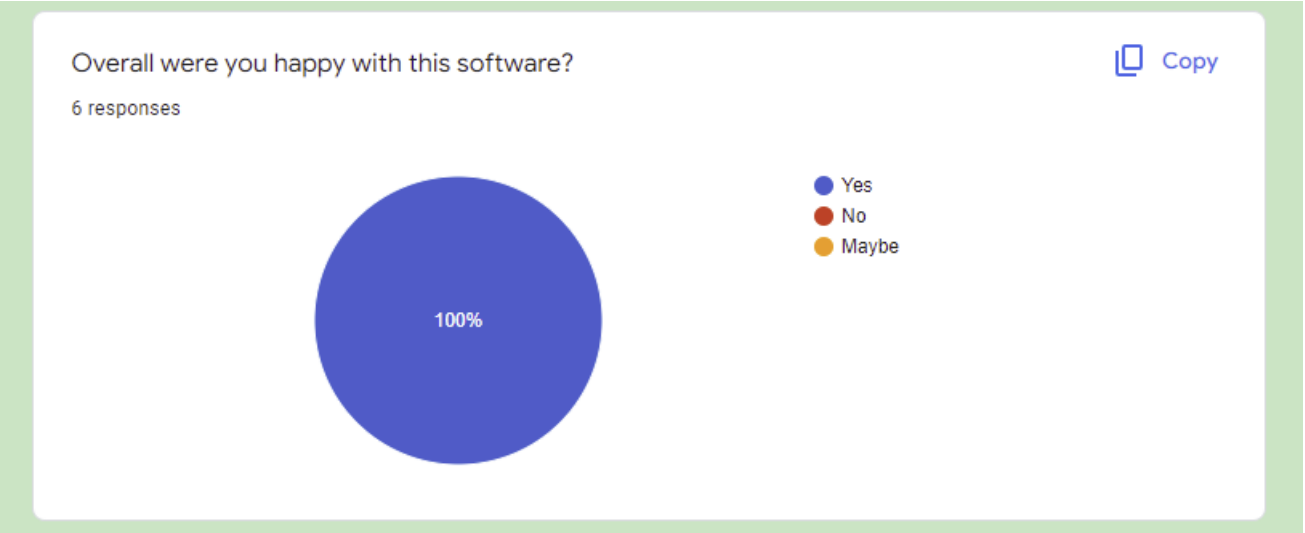


Figure 151 - Question 2 and feedback

For this question, people were asked if they were happy with the software. Everyone who submitted an answer said yes. This shows that overall, everyone was happy with it.

Question 3:

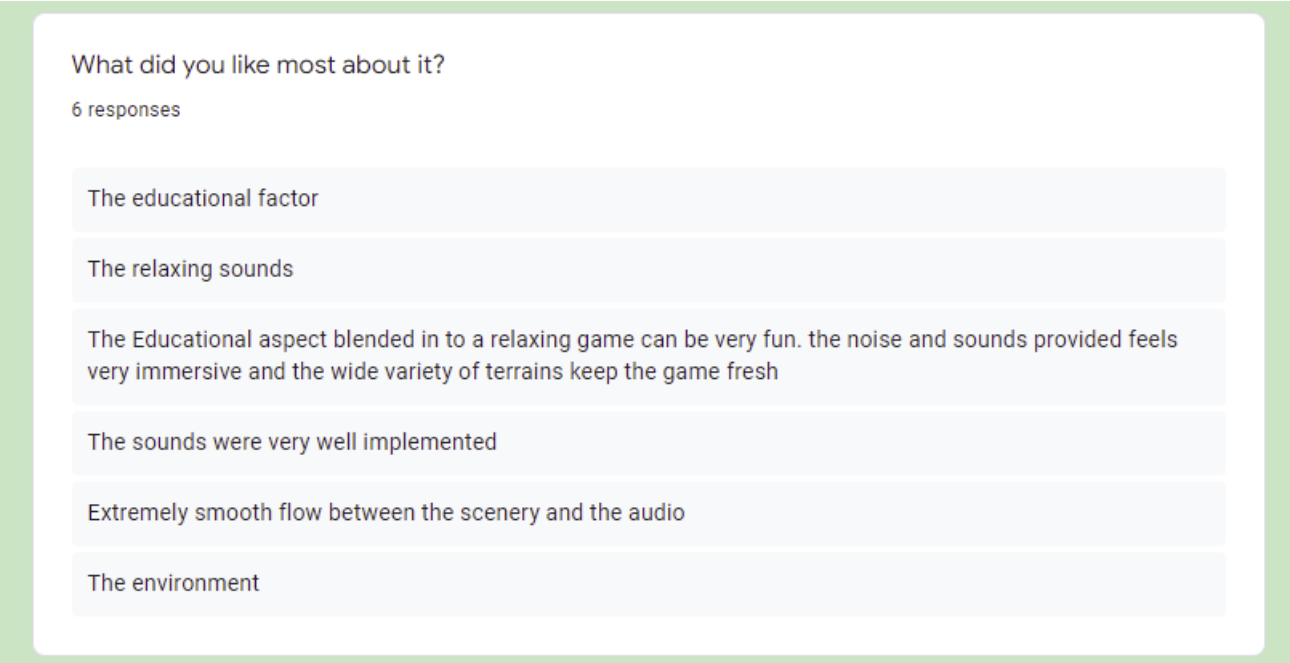


Figure 152 - Question3 and Feedback

For this one, they were asked what their favourite aspect of the game was. Most people said they liked the educational aspect and the environment of the game.

Question 4:

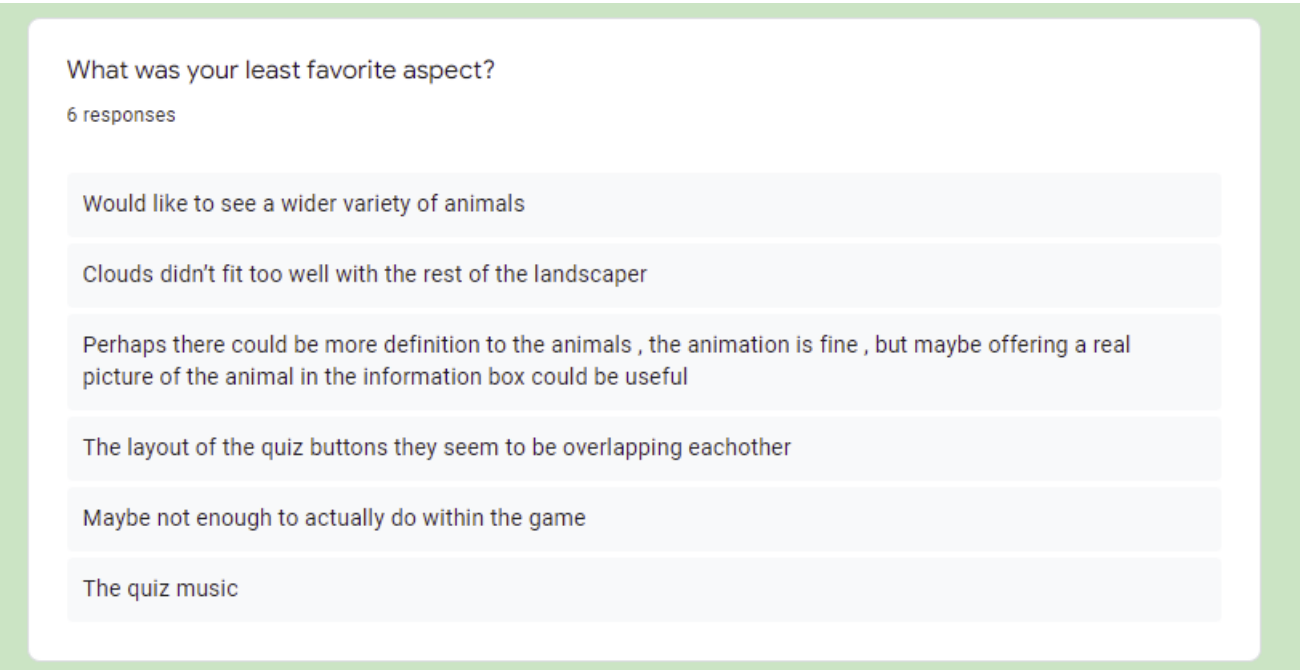


Figure 153 - Question 4 and Feedback

For this question, most people wanted to see more done with animal models and a wider variety of animals.

Question 5:

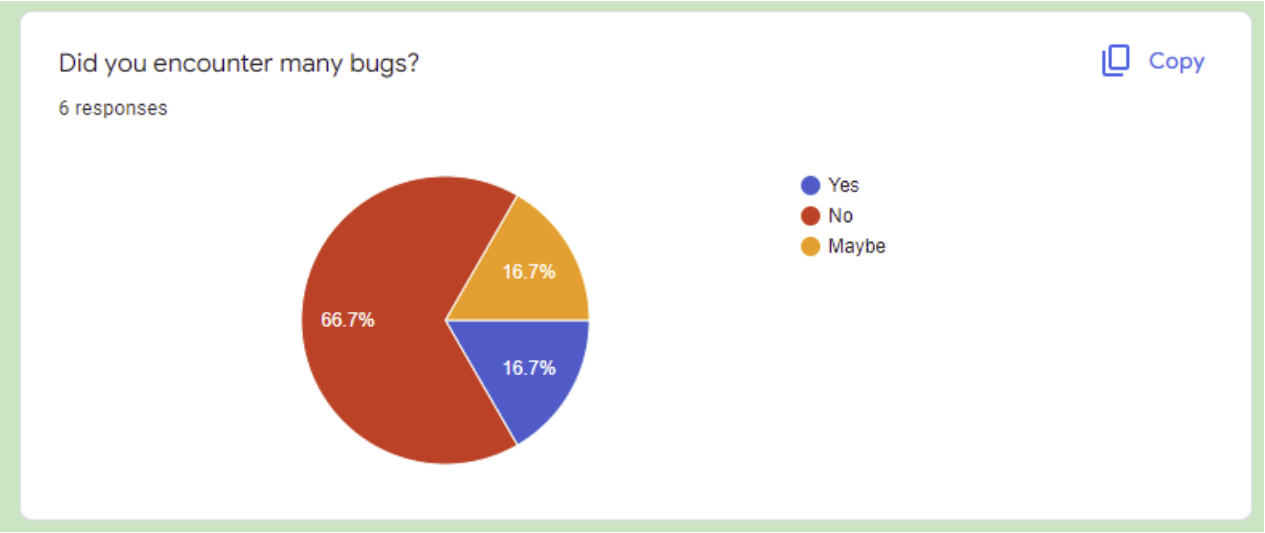


Figure 154 - Question 5 and Feedback

This question was asking if people encountered bugs, most people said non but there was still a large portion of people who said yes.

Question 6:

Figure 155 - Question 6 and Feedback

Most of bugs were with the grass generation, there were also bugs with seeing parts of the game you shouldn't be able to see.

Question 7:

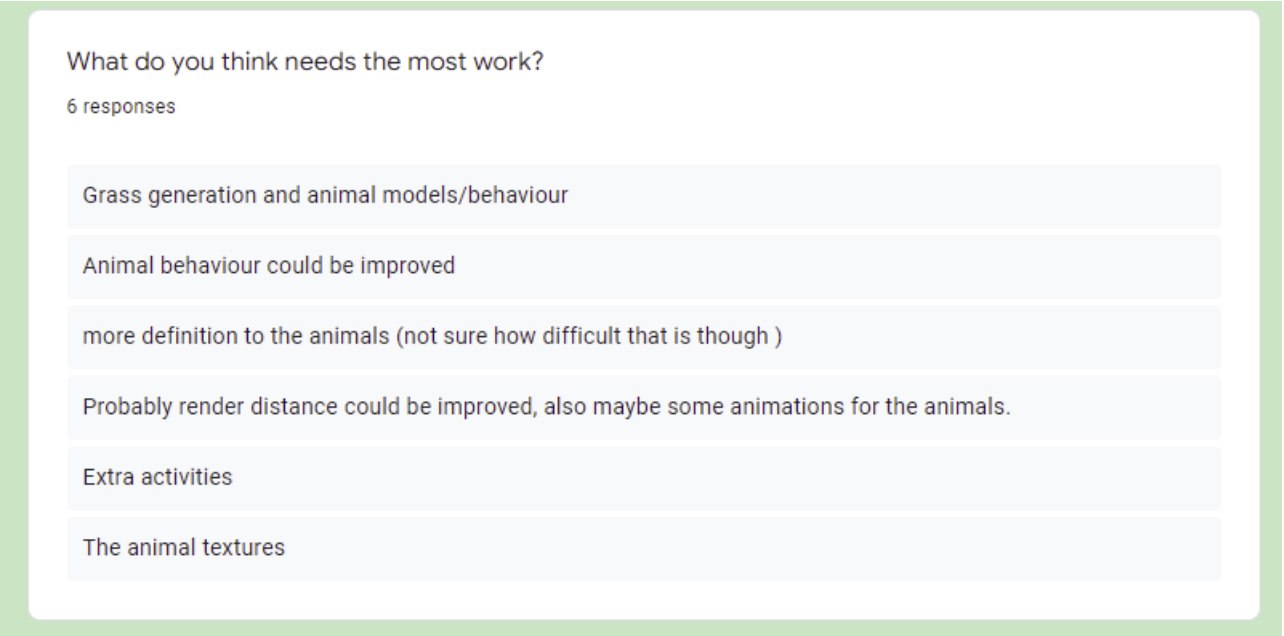


Figure 156 - Question 7 and Feedback

Most people thought that the grass generation needed the most work and they would've liked to see extra activities.

Question 8:

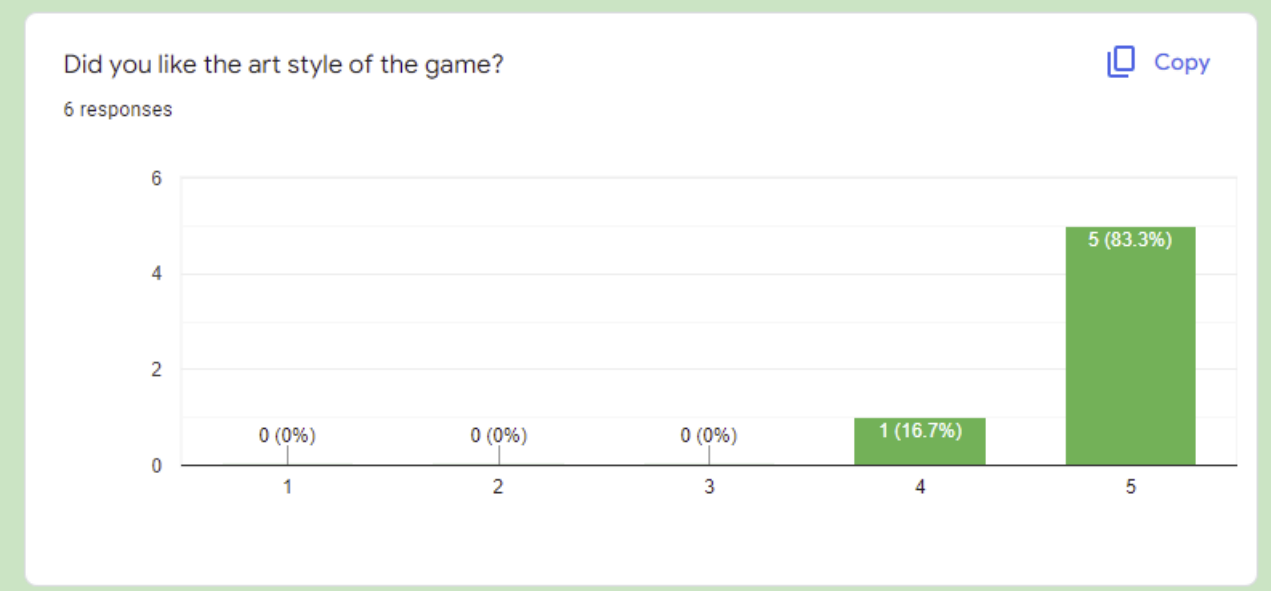


Figure 157 - Question 8 and Feedback

When asked about the art style of the game everyone seemed to like it.

Question 9:

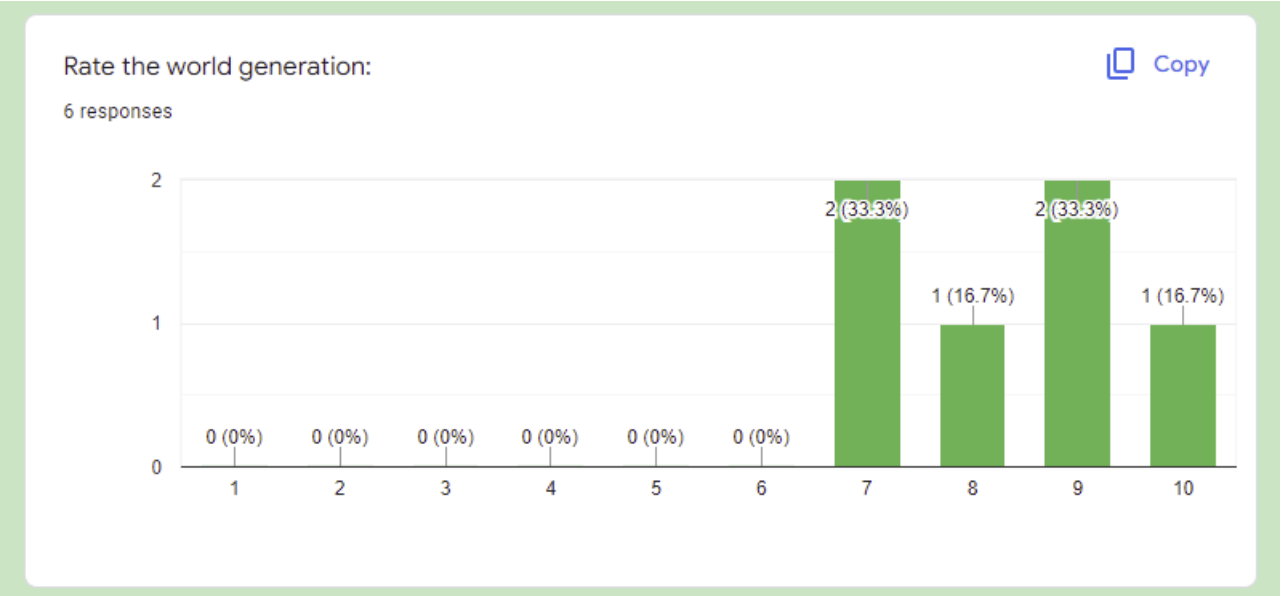


Figure 158 - Question 9 and Feedback

When it came to the world generation, the opinions were more varied compared to the other results. However, all the ratings were still high.

Question 10:

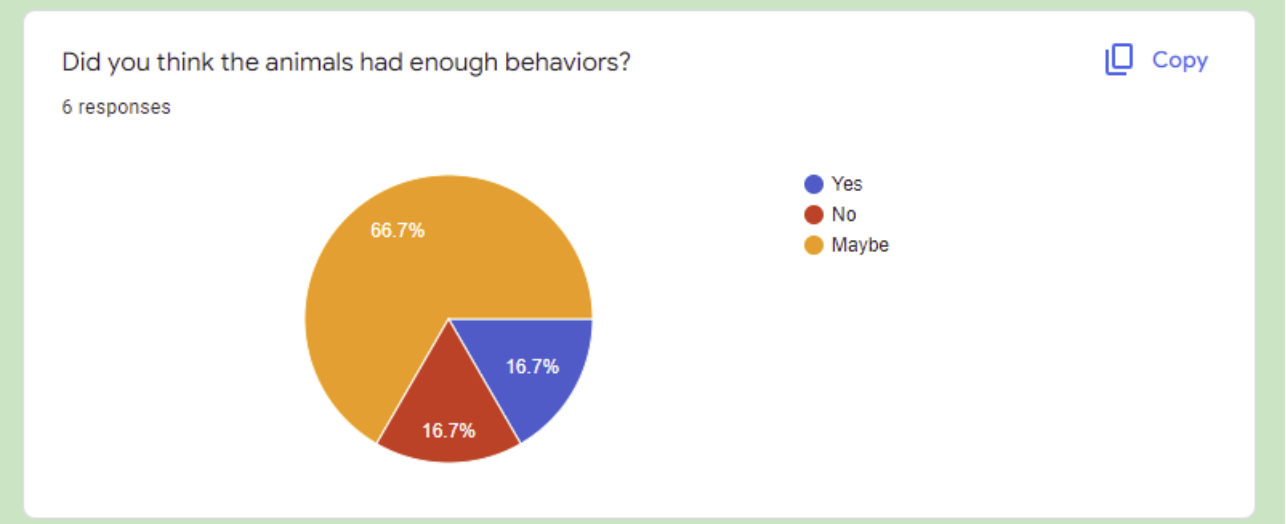


Figure 159 - Question 10 and feedback

When it came to the animal behaviours, most people said there wasn't enough. I think this is because the version that they tested didn't showcase enough of the behaviours. However, more behaviours should be added.

Question 11:

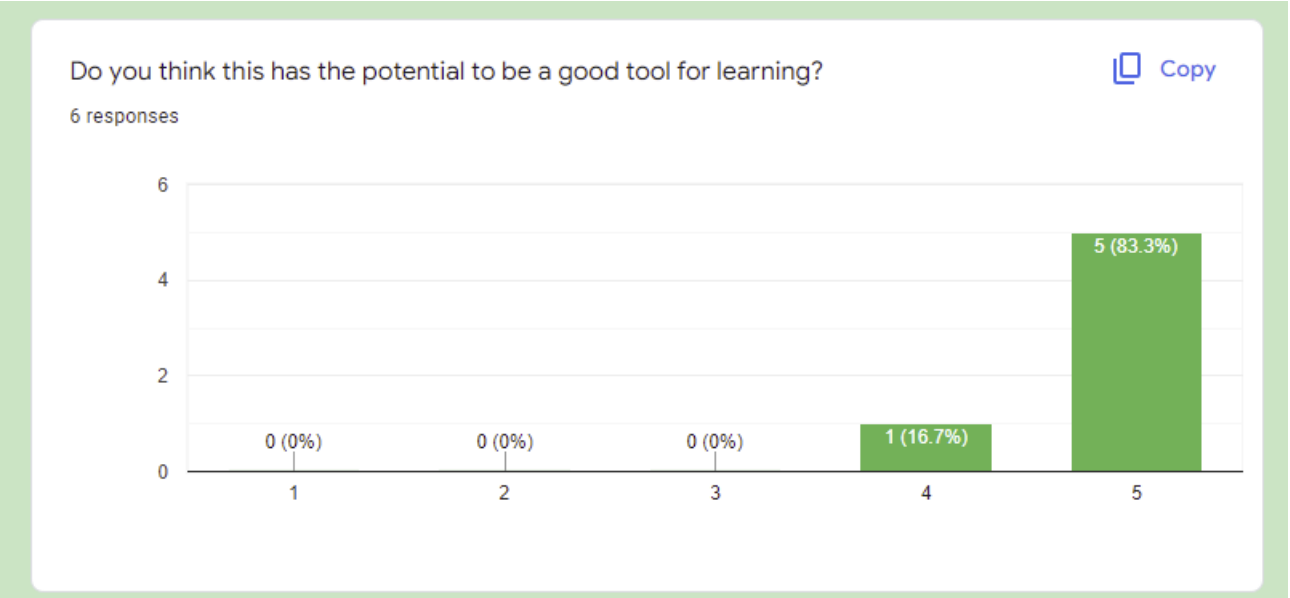


Figure 160 - Question 11 and Feedback

When asked if they saw potential for this game to be a good learning, everyone gave a high rating. This is good information to know, if the game was to be worked on further it could be quite successful.

Question 12:

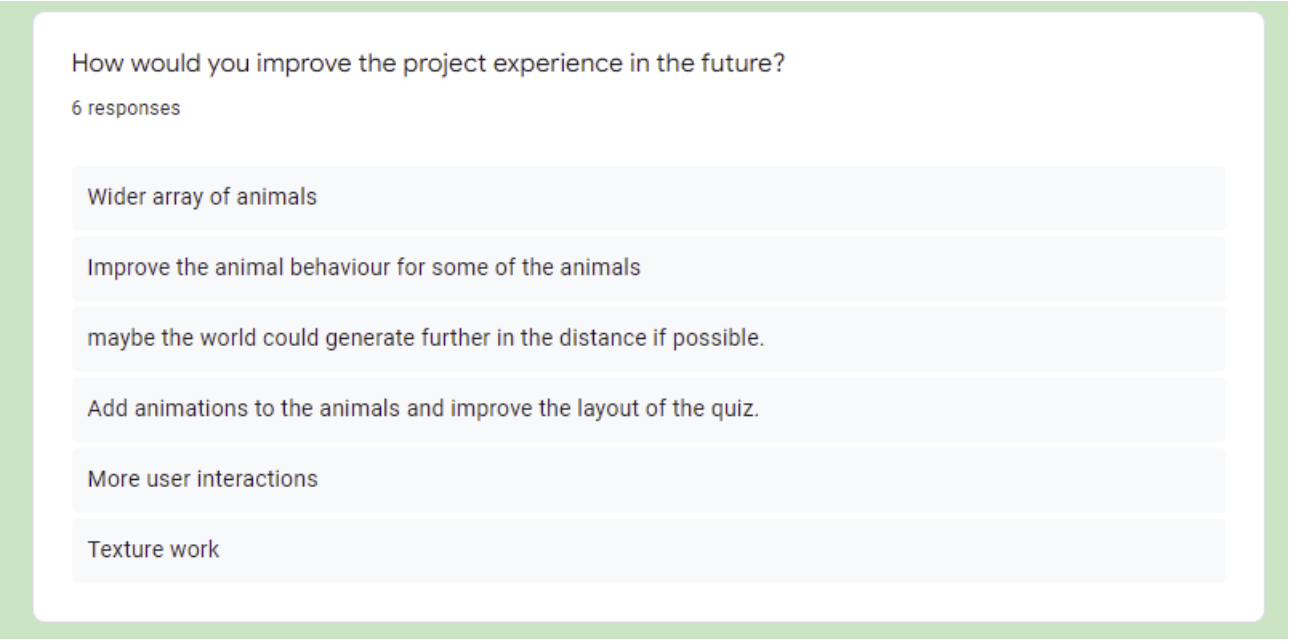


Figure 161 - Question 12 and Feedback

Valuable information for improvements for the game were received. Again, the animals were recommended to be improved.

Question 13:

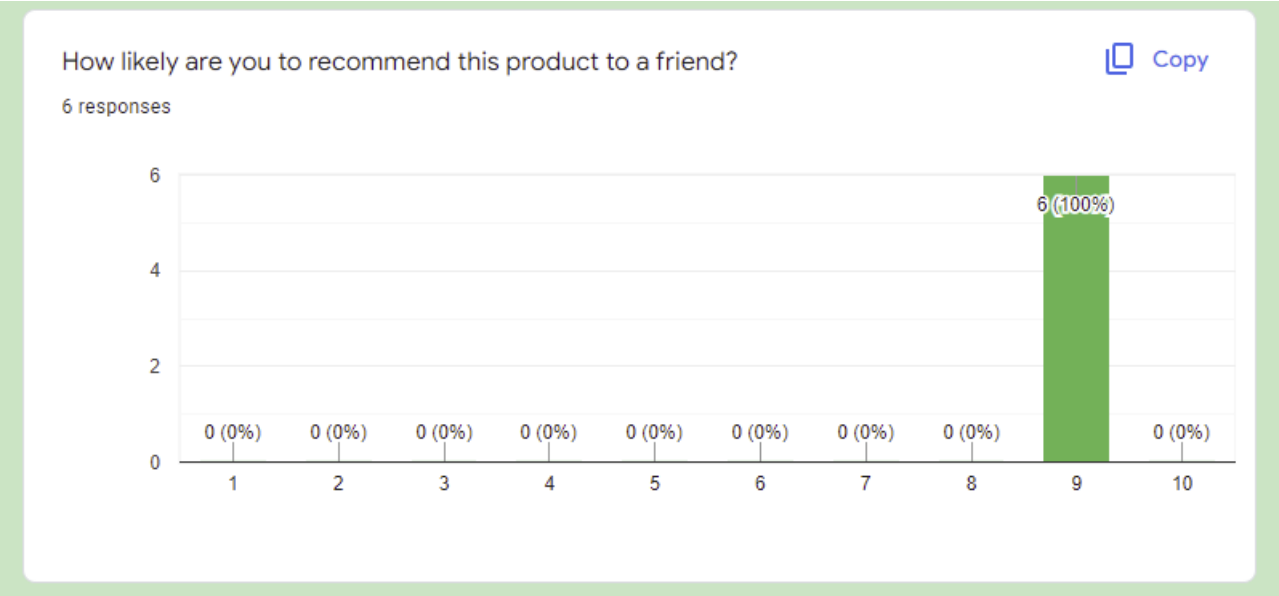


Figure 162 - Question 13 and Feedback

Question 14:

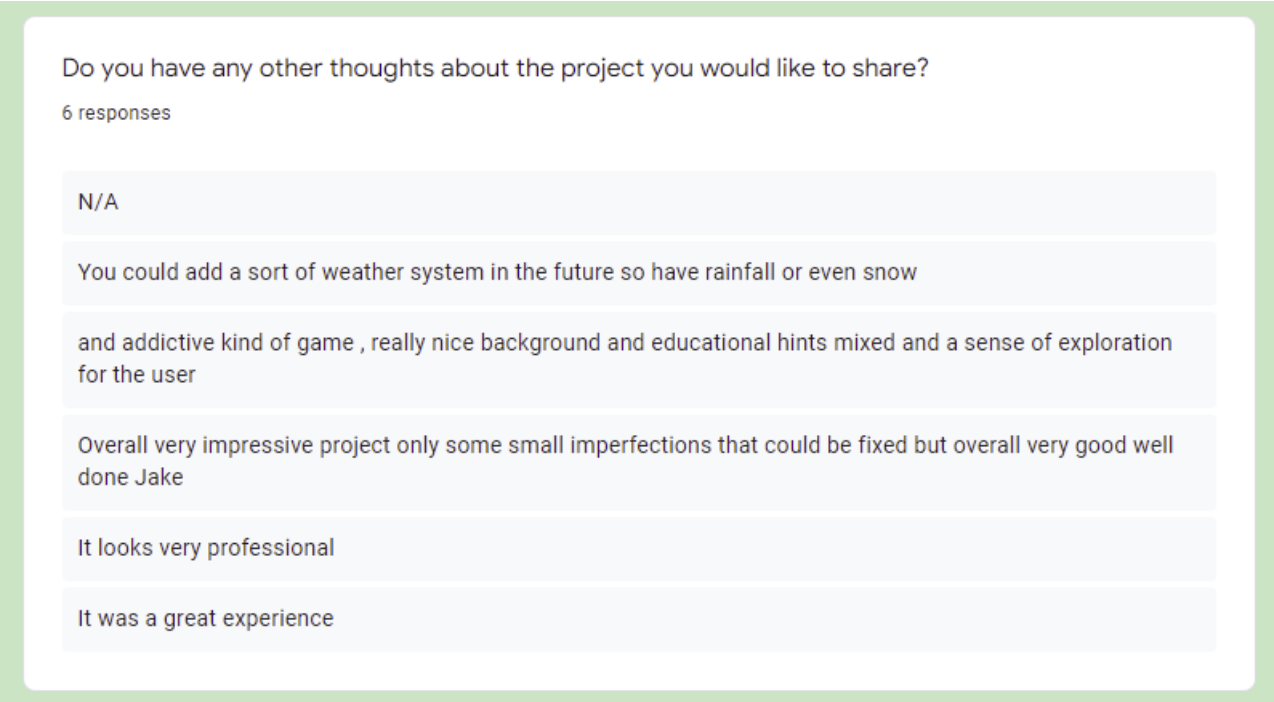


Figure 163 - Question 14 and Feedback

For the last question for final comments, most people gave positive and informative feedback. They overall seemed to have a good experience.

Most the users were impressed with application. There were a few recommendations for the application, and a few minor bugs. Overall, I was happy with the feedback, it was quite valuable and gave me ideas for future work.

5.4. Conclusions

This section discussed the testing and evaluation of this project. The testing discussed was white box and black box testing alongside using GitHub for backups and end-user testing. The evaluation consists of feedback from lots of potential end-users which was considered and used to change and successfully complete the system development.

6. Issues and Future Work

6.1. Introduction

This Final section of the project will reflect on the entire system. It will discuss possible future work for the project and then conclude the project by talking about its key points.

6.2. Issues and Risks

In this project there a several challenges or issues that are not yet resolved.
The issues and risks are in table format below:

Issue and Risks	Explanation	Solution	Solved? Y / N
Risk	Low amounts of knowledge in Rigging in Blender	Complete research and watch tutorials	Yes
Issue	handling the models might cause problems for example if you created a tree model in Blender, there could be thousands of leaves on that tree.	Make sure models had low number of leaves and objects.	Yes
Risk	The textures and materials might not be exported correctly.	Make sure texture types of match Unity's preferences	Yes
Risk	Models might cause unity to run slower	Make sure models are exported correctly.	Yes
Risk	Lack of Experience with coding animal AI.	Do research into tutorials and give lots of time to complete.	Yes
Issue	Only able to work on project on PC mostly from home as laptop	Make sure to do report work on laptop as well.	Yes

	isn’t very powerful.		
Issue	Workload could be too much, as system is feature heavy.	Get rid of unnecessary features.	Yes
Risk	using too much memory or space in the project	Using procedural generation allows the game to only take up memory whilst running.	Yes
Issue	The animal scripts for the AI will vary in functionality which will greatly increase the workload.	Re use some techniques used in the scripts.	Yes
Issue	Assets like materials and textures take up lots of memory.	Only save textures that are needed.	Yes
Issue	There are lots of minor errors with the NavMesh system when navigating around obstacles.	Use the navmeshcomponents package for better functionality.	Yes
Risk	Prototype not fully functioning for Demo.	Make sure there are no bugs.	Yes
Risk	Not meeting required deadline.	Make sure application is finished at least 2 weeks before deadline.	Yes
Risk	Research links might not be reliable sources of information.	Spend time checking if they are reliable.	Yes

6.3. Plans and Future Work

For Future work, there was a few different scenarios that were considered. This game would work very well as an educational game in the future.

Models

There were a few different issues with the game and lots of suggestions were made to improve it for a later version. The first was the models that were used in the game. Although the models of the trees were detailed and got a lot of good feedback, the animal models didn't receive the same praise. They were not as complex or detailed as the trees and could use some work. If they were to be improved in the future, they would be made to look more realistic and accurate to what they look like in real life.

AI

Suggestions for the animal AI in the system were also made. Many of the user thought that there weren't enough behaviours for the animals. If a later version of the game was made, a lot more behaviours would be added to the animals to give a variety of actions. This would make the game more interesting for the user.

Information

In the game, there was information given to the user about the wildlife. Users mentioned that it would be better to make the information more interesting by carefully selecting what information is used and to make it more visually appealing. This would make it more attractive to learn and use.

Procedural generation

A lot of procedural generation was used in this game. For future work, it would be better to add to the procedural generation by making it more complex and making it a lot bigger and colourful. This would make the game more diverse in that the user would enjoy a more unique experience when playing the game. Maybe more detailed objects in the forest would be generated, the mountains would be made to look better, and the animals would be generated with more variety.

Weather system

A good idea that was suggested was to incorporate a weather system, seen as though it's about nature and wildlife it would be good to see a simulation of the weather.

Inventory and save system

In the research section there are subsections about an inventory system and a save system. It was decided that for this project they were not needed. However, if a future version of this project that was more complex was made, these features could be added to increase the value of the user's experience.

All these ideas would certainly make the project better and more successful for the future, adding things like better aesthetics, and functionality could make this game into something very attractive to students for educational purposes.

6.4. Conclusions

The final part of this report will outline the conclusions of all the chapters. It will also briefly discuss some key points of the chapters.

6.4.1 literature Review

- Educational wildlife games have been created and are competent in what they are trying to achieve, but they don't have many features to allow the user to be tested on the knowledge and information.
- Other techniques apart from textbooks and lectures are a good way to educate students on topics, methods such as AI, 3D modelling, and procedural generation can be used to provide a more interesting approach to education.

6.4.2 Design

- ADDIE methodology is a great design tool for creating educational software. This provides a basic approach but is quite effective.
- Inspection, animation, quiz, UI, and level design is different from the normal take on an educational application. This requires the design to be carefully thought out. Scenarios need to be created to facilitate with this process.

6.4.3 Development

- Procedural generation is extremely useful to be used to great landscapes and environments in game to make the game more efficient and complex whilst providing the user with a unique and enjoyable experience. This makes the development easier to customize without having to build new models all the time.
- UI in Unity is a great method for presenting information to the user when creating a game. It allows the application to be seen in a way that's creative and easy for the user to look at.
- AI pathfinding is a great way to design and implement AI. This can set specific behaviours for different animals to act in a certain way and can be re used with other AI characters.
- Quizzes are a great tool for allowing the user to be tested on what they have learnt and is a great feature to be developed in an educational application.
- Ray casting is a good technique to allow players to interact with objects in a game to give a wide variety of options to choose from to learn about wildlife and access

other features through this method.

6.4.4 Testing and Evaluation

- The feedback and evaluation showed that lots of 3rd party users thought that it was a great tool. However, it did need some changes and needed to be improved for the future when looking at the animals, ai and size of the game.
- The Art style was appreciated by a lot of the users and the procedural generation worked very well when the application was being used. It allowed big environments to be created which made unique scenarios each time the game was played.

6.4.5 Issues and future work

- Even though the game received a lot of pf positive feedback. There were some issues that stood out. The animal AI didn't have enough variety in behaviours, and the animal models didn't represent closely enough to real life.
- Suggestions for additional features were taken into consideration such as, a weather system, improved AI, more interesting information.
- Most of the feedback was for aspects that could improve rather than aspects they should be fixed as most of the core features were working. Some of the features only needed further work or improvement.

6.4.6 Final Reflections

- Overall, this projected reached most of its goals and aims. These aims were to create a fun, interactive and unique learning experience application to allow users to learn about wildlife in an improved way.
- In terms of learning, this project has given me a huge insight into what goes into developing an application from scratch and a big look into the game development side of things. Research skills, design, and development skills have been gained by embarking on the journey of creating this project. Lots of valuable skills and knowledge have been gained along the way.,

- Lots of challenges were faced when developing this game. However, with lots of research and work through different technologies, the core functionalities of the project were able to be completed in the end.

- Planning was a big part in the process of this project. It helped with prioritizing and scheduling the different sections which resulted in a faster and more organized experience of developing.

6.5. GANTT CHART

For this project a GANTT chart was created. This chart was very long and couldn't be screenshotted and put in the report as the visibility wasn't very good. The chart was made into a table and shown below. This chart was used to plan out the deadlines and development of the project.

	Task Name	Duration	Start	Finish
1	Proposal	1 month	09/20/21	10/18/21
2	Introduction	2d	10/19/21	10/20/21
3	Project background	2d	10/20/21	10/21/21
4	Project description	1d	10/22/21	10/22/21
5	Project Scope	2d	10/23/21	10/25/21
6	Literatue Review	3d	10/26/21	10/28/21
7	Technologies to resea	1d	10/29/21	10/29/21
8	Design	3d	11/01/21	11/03/21
9	Software methodology	2d	11/04/21	11/05/21
0	requirements	1d	11/06/21	11/06/21
1	Front-end	3d	11/07/21	11/09/21
2	Prototypes	3d	11/10/21	11/12/21
3	Medium Fidelity	2d	11/14/21	11/15/21
4	Use case Diagrams	2d	11/15/21	11/16/21
5	Use case Scenarios	1d	11/16/21	11/16/21
6	System Design	1d	11/17/21	11/17/21
7	Art	1d	11/18/21	11/18/21
8	Models	1d	11/19/21	11/19/21
9	AI	1d	11/20/21	11/20/21
0	Map Design	1d	11/21/21	11/21/21
1	Audio	1d	11/21/21	11/21/21
2	Development	4d	11/22/21	11/25/21
3	Testing and evaluatior	1d	11/25/21	11/25/21
4	Issues and Future Wo	1d	11/26/21	11/26/21
5	Interim Report	25d	12/01/21	01/04/22
6	interim Demo	1d	01/22/22	01/22/22
7	Terrain Genration	1d	01/23/22	01/23/22
8	3d Modelling	1d	01/24/22	01/24/22
9	UI development	3d	01/27/22	01/31/22
0	Prototype	8d	02/09/22	02/18/22
1	Audio development	1d	02/19/22	02/19/22
2	AI Development	1d	02/21/22	02/21/22
3	Quiz development	3d	02/23/22	02/25/22
4	Inventory developmen	6d	02/26/22	03/04/22
5	Database Use	1d	03/03/22	03/03/22
6	Testing phase	3d	03/02/22	03/04/22
7	Export Phase	1d	04/01/22	04/01/22
8	Final Report Due	1d	04/04/22	04/04/22
9	Final Demo Due	1d	04/25/22	04/25/22

Figure 164 - GANTT Table

Bibliography

1. Beyond Blue – epicgames.com [internet]. [Cited 2021 November 1]. Available From: https://www.epicgames.com/store/en-US/p/beyond-blue
2. The hunter: Call of the Wild – callofthewild.thehunter.com [internet]. Cited 2021 November 1]. Available from: http://callofthewild.thehunter.com/
3. Zoo Tycoon – steampowered.com [internet]. Cited 2021 November 1]. Available from: https://store.steampowered.com/app/613880/Zoo_Tycoon_Ultimate_Animal_Collection/
4. Unreal Engine - unrealengine.com [internet]. Cited 2021 November 1]. Available from: https://www.unrealengine.com/en-US/
5. CryEngine – cryengine.com [internet]. Cited 2021 November 1]. Available from: https://www.cryengine.com/
6. Armory – armory3d.org [internet]. Cited 2021 November 1]. Available from: https://armory3d.org/
7. Unity – unity.com [internet]. Cited 2021 November 1]. Available from: https://unity.com/
8. Programming languages – makeuseof.com [internet]. [Cited 2021 November 7]. Available from: https://www.makeuseof.com/tag/unity-game-development-languages/
9. Blender – Trustradius.com [internet]. [Cited 2021 November 7]. Available from: https://www.trustradius.com/products/blender/reviews?qs=pros-and-cons
10. Cutscene – medium.com [internet]. [Cited 2021 November 7]. Available from: https://medium.com/geekculture/creating-a-cutscene-in-unity-aaec5042aaab
11. Inventory System – youtube.com [internet]. [Cited 2021 November 7]. Available from: https://www.youtube.com/watch?v=SGz3sbZkfkg
12. Inspection System – youtube.com [internet]. [Cited 2021 November 8]. Available From: https://www.youtube.com/watch?v=au3GrDuJaEw&t=557s
13. Procedural generation – adrianb.io [internet]. [Cited 2021 November 8]. Available From: https://adrianb.io/2014/08/09/perlinnoise.html

14. NavMesh – unity3d.com [internet]. [Cited 2021 November 8]. Available From: https://docs.unity3d.com/ScriptReference/AI.NavMesh.html
15. Procedural animation – youtube.com [internet]. [Cited 2021 November 15]. Available From: https://www.youtube.com/watch?v=acMK93A-FSY&t=115s
16. Project no 1 -tudublin.ie [internet]. [Cited 2021 November 15]. Available From: https://library-cc.tudublin.ie/search/?searchtype=X&searcharg=dt211c+or+DT228+or+DT282+or+DT255&sortdropdown=-&SORT=DZ&extended=0&SUBMIT=Search&searchlimits=&searchorigarg=Xdt211c+or+DT228+or+DT282%26SORT%3DDZ
17. Project no 2 -tudublin.ie [internet]. [Cited 2021 November 15]. Available From: https://library-cc.tudublin.ie/search/?searchtype=X&searcharg=dt211c+or+DT228+or+DT282+or+DT255&sortdropdown=-&SORT=DZ&extended=0&SUBMIT=Search&searchlimits=&searchorigarg=Xdt211c+or+DT228+or+DT282%26SORT%3DDZ
18. Design methodology – ceur-ws.org [internet]. [Cited 2021 November 15]. Available From: http://ceur-ws.org/Vol-1394/paper_9.pdf
19. ADDIE Model – learnupon.com [internet]. [Cited 2021 November 15]. Available From: https://www.learnupon.com/blog/addie-5-steps/
20. Unity System architecture – google.com [internet]. [Cited 2021 November 15]. Available From: https://www.google.com/search?q=inty+system+architecture&rlz=1C1JJTC_enIE927IE927&sxsrf=AOaemvLjQf-W0UbMWvs2ln-nGv4YvuGppA:1641213337427&source=lnms&tbm=isch&sa=X&ved=2ahUKEwi6gOmPzJX1AhV4QkEAHbo7CZEQ_AUoAXoECAEQAw&biw=2133&bih=1041&dpr=0.9
21. Art – youtube.com [internet]. [Cited 2021 November 15]. Available From: https://www.youtube.com/watch?v=QTM2Yr_EibU
22. Models – vincentwildlife.ie [internet]. [Cited 2021 December 16]. Available From: https://www.vincentwildlife.ie/species
23. Audio – unity3d.com [internet]. [Cited 2021 December 16]. Available From: https://docs.unity3d.com/Manual/class-AudioSource.html

24. Tree Creation Blender – youtube.com [internet]. [Cited 2021 December 16]. Available From: https://www.youtube.com/watch?v=jMDRc4hJwvA&t=507s
25. Testing White and Black – geeksforgeeks.org [internet]. [Cited 2021 December 16]. Available From: https://www.geeksforgeeks.org/differences-between-black-box- testing-vs-white-box-testing/
26. Unity testing – unity3d.com [internet]. [Cited 2021 December 16]. Available From: https://docs.unity3d.com/Manual/testing-editortestsrunner.html
27. Scripts testing – unit3d.com [internet]. [Cited 2021 December 28]. Available From: https://docs.unity3d.com/Manual/ManagedCodeDebugging.html
28. Gantt chart – smartsheet.com com [internet]. [Cited 2021 December 28]. Available From: https://app.smartsheet.com/sheets/48hphGP8p6Prfm3Rhgc3cp383Q 54rj78W8rxMqQ1?view=gantt
29. Design – The Art of Game Design [Book]. [Cited 2021 January 1]. Available From: The Art of Game Design by Jess Schell
30. https://github.com/jakebolger/Wild-Ireland-FYP [Cited 2021 January 1]. Available From: GitHub
31. https://www.edutopia.org/article/5-tips-designing-multiple-choice-quizzes [Cited 2021 April 1]. Available From: Edutopia.org