OCR GCE A COMPUTER SCIENCE PROJECT

Full name	**James Keywood**
Candidate number	7060
Centre name	Kingston Grammar School
Centre number	14415
Project title	Gardening game
Qualification code	H446-03
Date	11-05-22

Contents

Initial ideas

Project proposals

Gardening game

To create a 3D game on Unity in which the user can roam around collecting plants and is able to bring them back to a greenhouse. I would like to include procedural generation to randomly create the environment around the greenhouse that the user can roam around in to add variety to the game. It would be generated every time the user leaves the greenhouse. Perhaps the end goal is to stop the random land generation. The success criteria are as follows: to include procedural generation, random plant and grass placement, a greenhouse that the user can come back to and plant things, and an end goal such as stopping the random land generation.

Maze game

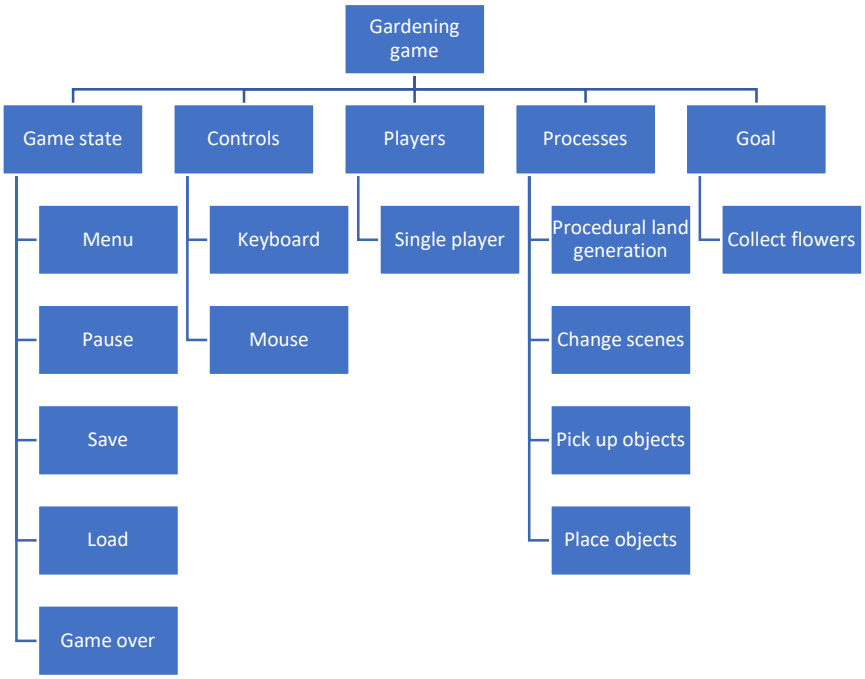
A 3D maze game on Unity in which the user has to escape a randomly generated maze while being chased by enemies. I would use Unity's build in navigation system for the enemies. Harder levels could be added as progression or faster enemies. The success criteria are as follows: to include randomly generated mazes, enemies with pathfinding, and some sort of progression like harder levels.

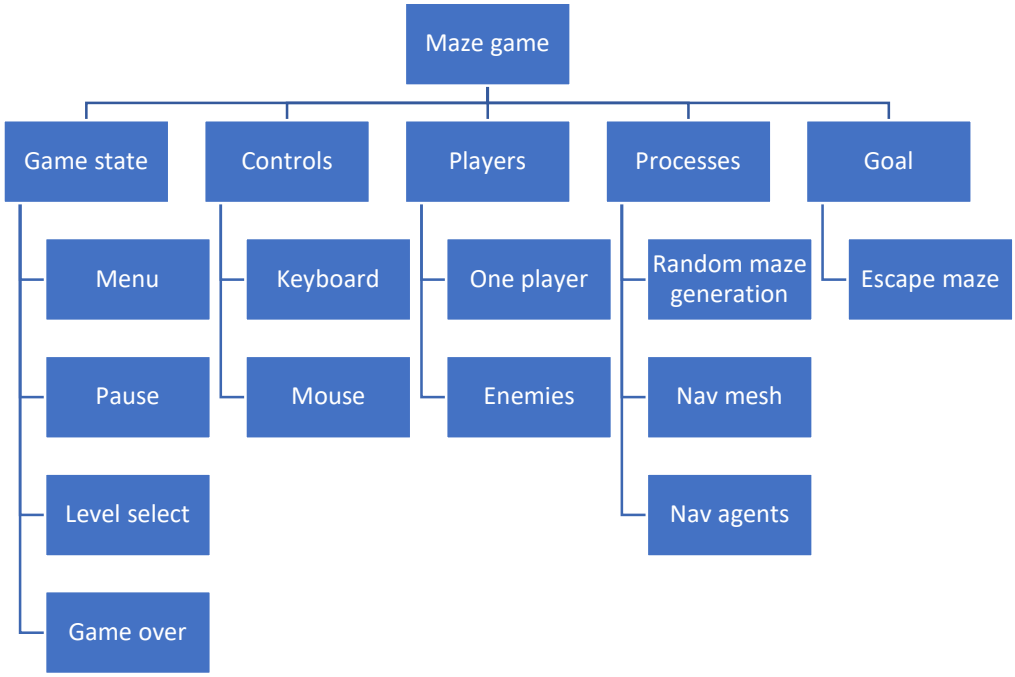
Jumping game

A 3D game on Unity in which the user has to jump and run through levels of obstacles. The game could keep track of the time taken. There could be a variation of difficulty in the levels along with powerups or boosts that the user can use. The success criteria are as follows: to include different levels and variety of difficulty, movement mechanics such as momentum and jumping, and powerups and boosts.

I decided that the jumping game would not include enough algorithms or complex computational problems and narrowed it down to the gardening and maze game.

Systems diagrams





Similar programs

Gardening game

Outer Wilds

In the game Outer Wilds, the player has 22 minutes to launch into space before the sun goes into a supernova. The objective of the game isn't to avoid this fate, since no one knows it's coming. Instead, the goal is to just explore the solar system. After the 22 minutes is up, everything ends and you start all over, retaining the memory of your past life.[1]

I love the time loop mechanic introduced in this game and think it would apply nicely to my project, pairing with the idea of procedural generation.

Maze game

Soul Knight

This is a game that allows the user to select characters and explore a randomly generated, 2D dungeon. The method of level generation in this game could be applied to this project.

Entombed

This is an Atari 2600 game designed by Tom Sloper and programmed by Steven Sidley[2]. The game involves the player moving through a maze whilst avoiding enemies. The mazes are generated in a sequence and uses an algorithm to generate[3]. This description matches my goal perfectly.

I could use this approach to design my algorithm for randomly generating mazes.

Decision

I concluded that the gardening game would include more interesting and complex computational problems for me to overcome and therefore chose this for my project.

[1] (Rivera 2019)
[2] (Entombed (Atari 2600) n.d.)
[3] (Baraniuk 2019)

Analysis

Project definition

Gardening or farming games have been around for a couple decades now, to the point where users can play games that completely simulate farm life or managing a garden. For example, Farming Simulator 19 sold over two million copies[4] and managed to earn $143 million in one year of release.[5]

For my project, I plan to make a gardening game in which the user can explore a random, procedurally generated landscape and collect plants to bring back to their greenhouse. I would like the game to be of a low poly art style for multiple reasons; I think that it is an appealing style for a gardening game to have, but also since it reduces the number of vertices in the game, increasing performance for the user. It would be a single player game in which the user's goal is to collect different types of plants. I like the idea of a time loop, similar to in the game Outer Wilds, in which the user will get lost if they are exploring for too long. After a cycle, the landscape would change with procedural generation, adding variation to the experience.

For this all to work together, the game will require: an algorithm to randomly and procedurally generate landscape; a function to change scenes between outdoors and the greenhouse; a system that allows the user to pick up plants and choose where they should be planted; an indicator to let the user know how long they have been exploring; an inventory to show what the user has collected; easy keyboard and mouse input; a start-up screen with new game or load; a pause menu to save or continue; a game over screen for when the user gets lost.

To implement all of this, I am going to analyse similar games such as Outer Wilds, Animal Crossing and Stardew Valley. There are aspects and mechanics in each of these games that I wish to implement into my own.

Identifying suitable stakeholders

The low poly style of the game, paired with the lack of speaking, complex mechanics and a relaxing game loop make this game suitable for a wide range of ages. The game will be played with a mouse and keyboard on a PC, making it easy and simple to play. However, a simple touchscreen joystick could replace the keyboard and the user could tap the screen instead of using a mouse, making it quite easy to adapt the game to be played on a mobile, touchscreen device. This could make it even more accessible, lowering the target audience age even further.

With this in mind, I think that the game should be designed with a target audience of age 7 and up. Due to the calming nature of the project outline, I think the target audience will include a lot of users who experience stress and anxiety.

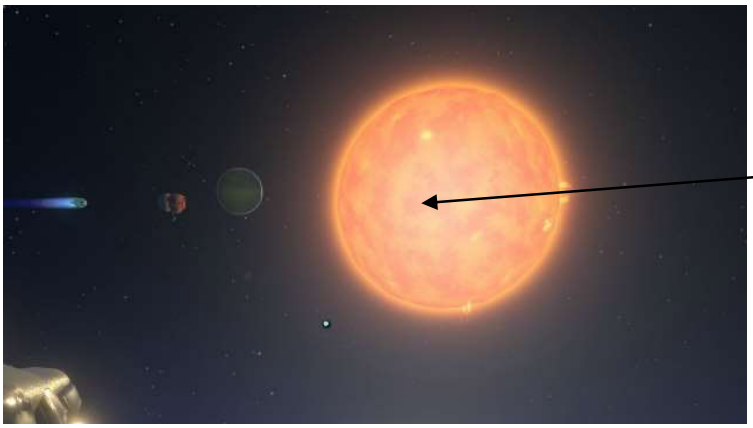
To act as a representative of this target audience, I have chosen Dan Hepburn. He is a seventeen-year-old student studying at my school. He likes to play video games in his spare time, making him great for feedback since he can compare the game to others he has played. Furthermore, I am able to keep regular contact with Dan since I share many school classes with him. On top of this, Dan experiences a lot of stress about his school work, so will be a great candidate for testing whether the game produced helps him feel less anxious. He has mentioned that he has been looking for a calming experience when playing games, so this will be a focus for me during development.

[4] (Macgregor 2019)
[5] (Macgregor 2019)

Research into existing games

Outer Wilds

In the game Outer Wilds, released in May 2019, the user plays as an unnamed, alien space explorer. However, the user may quickly realise that they are trapped within a time loop.



The sun that falls into supernova after 22 minutes of play time, causing the time loop for the user. Progress also resets if the player dies at any point.

[6]

Simple and straight forward game over screen

[7]

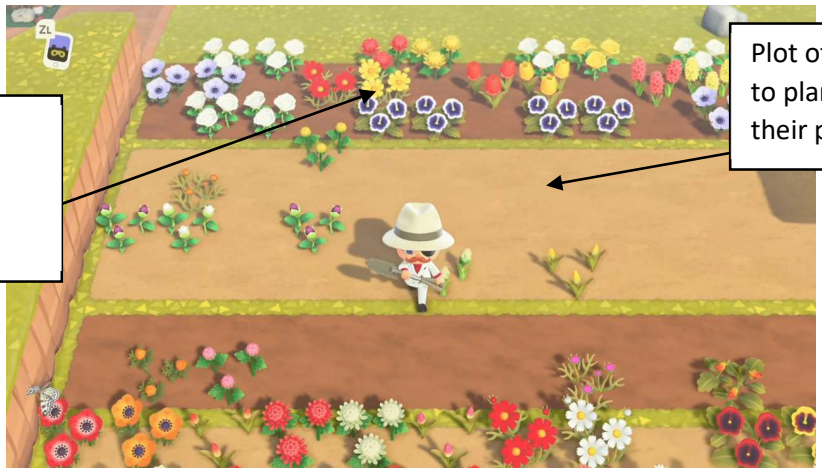
This concept of a time loop is really intriguing to me, and I would like to incorporate it into my game. Instead of the game resetting every 22 minutes, I think that I would introduce a mechanic in which the user gets lost after 10 minutes or so. For the user to know when this will occur, I may introduce a timer on screen or some kind of sound that implies they will get lost soon and lose all the plants they had collected on that journey. The game over screen in Outer Wilds is also really clean and simple, and I may implement something like it to tell the user when they are lost.

[6] (Prahlow 2019)
[7] (Prahlow 2019)

Animal Crossing

In the game Animal Crossing, the user moves to their very own deserted island, on which they can perform tasks or develop the island to their liking.

[8]



[9]

The concept of picking randomly coloured flowers is a mechanic that I will certainly use in my game. The crossbreeding concept in Animal Crossing is also very interesting and adds purpose to collecting different types of flowers. Whilst this would be fun to add to my game, I think it would be very time consuming and difficult to implement, so I won't include it. The use of trees scattered around and a simple grass texture adds a lot of atmosphere to the game whilst being easy to implement. Furthermore, I like the idea of a plot of land for the user, but I think I will use a greenhouse instead of an open, outside area.

[8] (Totaka 2020)
[9] (Totaka 2020)

Stardew Valley

This game is a farming simulator in which the player can create a new character and will then receive a plot of land and a small house. It allows the player to grow crops, raise livestock, mine, forage and more.

Aesthetic background

Opening screen for Stardew Valley

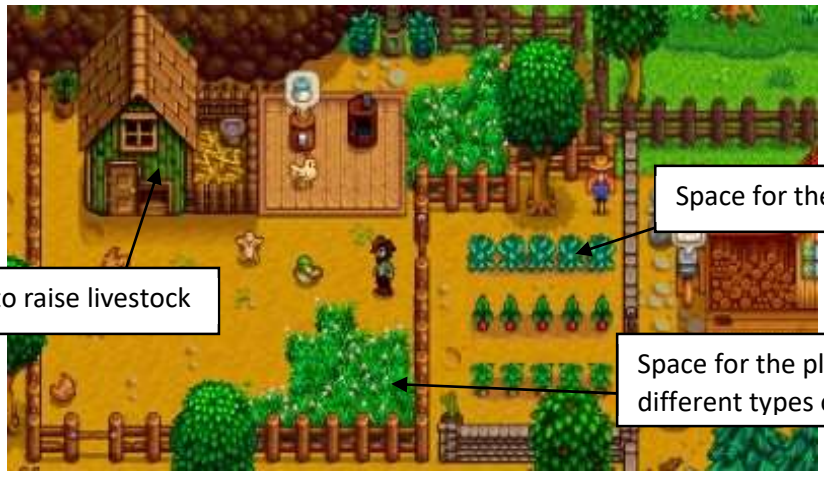
Simple and aesthetic buttons for the user to make a new game, load a game, open co-op or exit.

[10]

Trees scattered around landscape

House for the player to live in

[11]



Space for the player to grow crops

Housing for the player to raise livestock

Space for the player to grow different types of crops

[10] (Barone 2016)
[11] (Barone 2016)

[12]

[13]

Overall, this game has proven to be very helpful at this stage of development. The opening screen aesthetic is very similar to how I would like to implement my own, simple and clean. Furthermore, the house that the player is given when the game starts is a similar scale and size to how large I would like the player greenhouse to be in my game. The way trees are scattered around the landscape in Stardew Valley is also a good balance between cramped and overly spacious. Whilst being able to catch and raise livestock in my game would add to the content, I think it would be slightly out of my scope and be very difficult to implement. However, the use of fences and dividers in the plot of land for differnet types of plants is nice and I would like to do something similar, Finally, I really like the design of the player's map in this game and it would be great if I could implement something similar for the randomly generated landscape in my game.

Stakeholder interview 07/06/22

An interview with my stakeholder Dan Hepburn to determine basic mechanics, visuals and sounds that should be implemented into the game.

Game mechanics

Should I include a score system for collecting plants?

Yes, it adds some depth to the game and allows the user to compete with their past scores if high scores were saved.

How long should the time loop be?

Around 10 minutes would be a good compromise since it gives the user enough time to explore and collect plants, but also forces the user to return to their greenhouse in a reasonable amount of time.

How many plants should the greenhouse store?

Around 60 plants would be a good number since it ensures that the user has to think about what plants to keep but also does not feel too claustrophobic.

Should I include different types of plants?

[12] (Barone 2016)
[13] (Barone 2016)

Yes, definitely since adding variety is a good idea and benefits the user experience.

How realistic should I attempt to make the game?

I don't think any aspects such as hunger, thirst or fatigue should be implemented since it does not fit with the context.

Should I include wild animals that roam around the landscape?

I think that is a good mechanic to add since it adds a level of immersion to this simulator game. If it doesn't negatively impact the user experience, then it should definitely be included.

Is single player sufficient or should I implement a multiplayer mode?

Since this game is a very calming and relaxing game, multiplayer would negatively impact the user experience since it might make the game stressful or too complex.

Game visuals

What game style should I use?

A simple game style, such as low poly, would add to the peaceful and simple nature of the game.

What kind of colours should I include in a game like this?

Bright but not obnoxious colour schemes.

Should the game generate unique plants?

Yes, since it helps to increase variety in the user experience.

Sound effects

Would sound effects add to the user experience?

Yes definitely, such as footsteps, animals making noises, trees rustling, background music, interaction sounds.

What kind of music would benefit this game?

Tranquil and calm music, similar to the music style in Minecraft or Firewatch, helps to add to the atmosphere and nature of the game.

Overall feedback

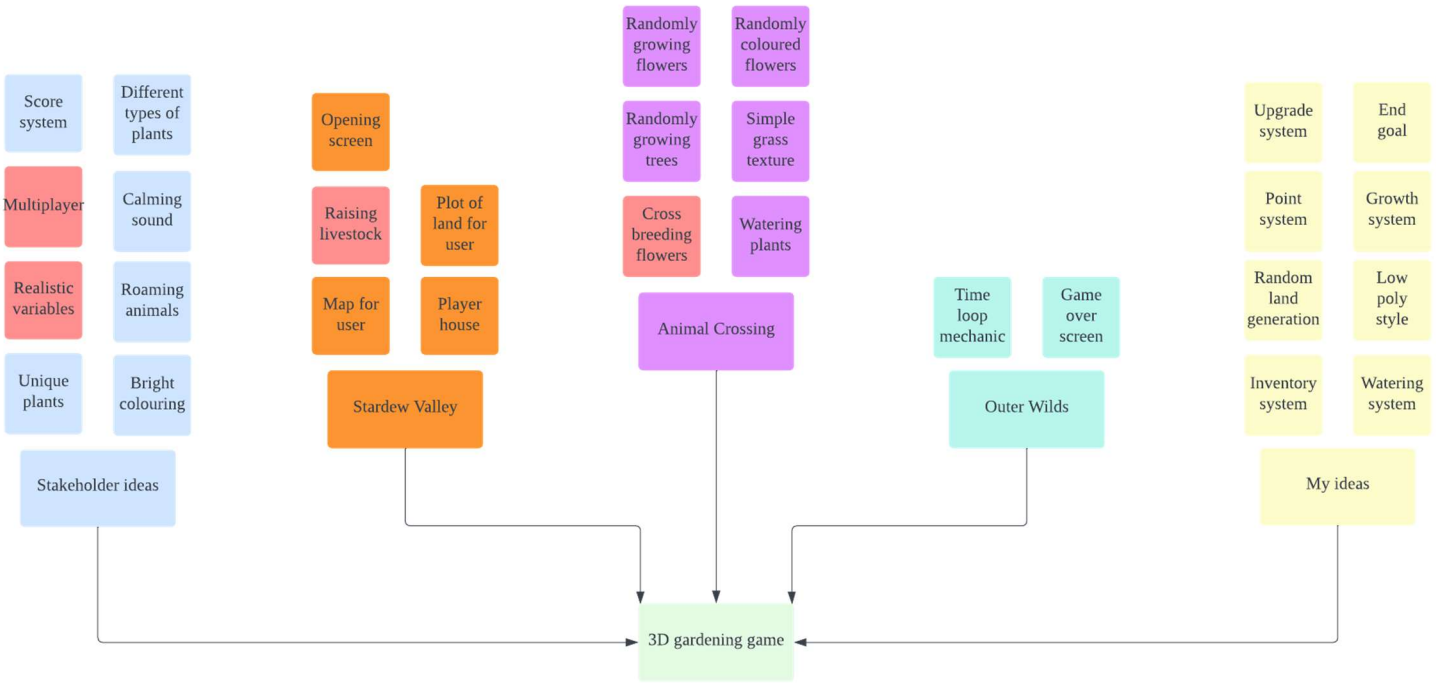
This interview proved to be very helpful in understanding the basic details that I want to include in this project. I should include a scoring system, perhaps with a high score so that there is a competitive aspect to the game. Furthermore, the time loop should be around 10 minutes. The user should be able to collect unique and different plants every time they leave the greenhouse. I will use a low poly game style with bright colouring. If I have time, I will include wild animals that roam the landscape to add to the. It will be single player due to the peaceful nature of this project. Finally, I will include game sounds and a calming backing track since it adds to the atmosphere of the game.

Identifying essential features

Source	Feature	Justification
My ideas	Low poly style	A low poly style is good for multiple reasons. It reduces the complexity of shapes in my game world, increasing performance. This also makes it easier for me to make parts of my game world. On the other hand, it is a very appealing art style and will be used for my game.
	Random land generation	Random land generation will certainly be a complex challenge, but I think it is very important for game variance and the user's experience. I will use Perlin noise to make this possible and will be implementing it into my game.
	Inventory system	An inventory system will definitely be useful for all sorts of reasons; so that the user can view what they have collected whilst exploring, to see what they are taking out exploring, and more. This will be included.
	Watering system	A watering system will add a time-based pressure to the user. If they do not collect water whilst exploring, they will not be able to water the plants in their greenhouse and their plants will die. I will include a feature like this.
	Growth system	A growth system allows users to see a result of their in-game work and I think it is an essential feature for a game like this. I will include it in my game.
	Point system	A point system allows the user to convert their grown plants into points, which can be used to upgrade their tools.
	Upgrade system	An upgrade system allows the users to upgrade their tools so that they work quicker.
	End goal	An end goal adds incentive to the game and could require lots of points.
Outer Wilds	Time loop mechanic	This feature will be included in my game. Instead of using 22 minutes, I will use a loop of 10 minutes. If the timer ends while the user is exploring, they will get lost and lose any plants they had collected on that specific journey. This adds time pressure to the game and makes it more interesting for the user.
	Game over screen	This simple game over screen featured in Outer Wilds is clean and straight forward, and I will include something similar in my game. It makes it easy for

		the user to understand what happened without confusing them.
Animal Crossing	Randomly growing flowers	This feature will definitely be included in my game. I may include different types of plants and flowers for the user to collect. Randomly placed plants adds variation to the user experience.
	Randomly coloured flowers	The implementation of randomly coloured flowers with different types of flowers is a nice feature in this game and I would like to include it. It adds variation to the experience which is fun for the player.
	Randomly growing trees	The randomly growing trees in Animal Crossing adds atmosphere to the game world. For this reason, I will include something similar to my game.
	Simple grass texture	The grass texture in Animal Crossing is simple yet effective and is not resource heavy. I would like to include a pattern like this for my grass texture in my game since it adds to the atmosphere of the low poly art style.
	Crossbreeding flowers	Whilst being able to crossbreed plants would add content to the game, I think that it would be too difficult to implement and would be very time consuming. I will not implement it in my game.
	Watering plants	The idea of watering plants would be very effective in my game. It would add time pressure on the user since if they don't water their plants they will die. The user will have to find water on their journey out the greenhouse.
Stardew Valley	Opening screen	The simple and aesthetic opening screen in Stardew Valley is very effective and I would like to include something similar in my game. This entices the user into playing the game if it has a good opening screen.
	Player house	The house that the player is given in Stardew Valley is a good size and aesthetic, and I would like to include something close to that scale in my game. I think that a house that is somewhat small makes the user think about what plants they like and which ones they want to keep.
	Raising livestock	Although a feature like this would be a great addition to the game and user

		experience, I think it is out of the game's scope and would be very difficult to implement. This will not be an addition to my game.
	Plot of land for the user	The implementation of a plot of land for the user in Stardew Valley is very similar to how I had planned to use it in my game. A small house/greenhouse is exactly what I would like to include and is why I will be using a similar implementation.
	Map for the user	Before looking at this game I had not thought about allowing the user to see a map of the area. The way Stardew Valley uses a map is very aesthetic and simple and I think it would be interesting and challenging to have a top-down map for the user to look at when exploring. I will be adding a map into my game.
Stakeholder ideas	Score system	A score system will add a competitive aspect to the user experience, especially if paired with a high score system, and is why I will be implementing it.
	Different types of plants	Having different types of plants instead of just flowers is interesting and adds variation. This is why I will be adding it.
	Realistic variables	Although variables such as fatigue, thirst or hunger would make the game more realistic, it does not really fit with the context of this calming and simulator game which is why it is not being added.
	Roaming animals	Roaming animals is a feature that will be a challenge to implement but will add a lot to the atmosphere of the game world. I will be implementing a feature like this.
	Multiplayer	Multiplayer does not fit with the calming nature of the game and would be very difficult to include. This is why it will not be included.
	Bright colouring	I think that bright colouring fits with the low poly art style and makes the game very cheerful and appealing for players. This is why it will be used.
	Unique plants	Generating unique plants with different rarities may pose a tough challenge but I think it will add a lot of variation and is why I will include it.
	Calming sound	Calming sound effects and music fits with the atmosphere that I am trying to achieve, and it will be used.


Score system
Different types of plants
Multiplayer
Calming sound
Realistic variables
Roaming animals
Unique plants
Bright colouring
Stakeholder ideas
Opening screen
Raising livestock
Plot of land for user
Map for user
Player house
Stardew Valley
Randomly growing flowers
Randomly coloured flowers
Randomly growing trees
Simple grass texture
Cross breeding flowers
Watering plants
Animal Crossing
Time loop mechanic
Game over screen
Outer Wilds
Upgrade system
End goal
Point system
Growth system
Random land generation
Low poly style
Inventory system
Watering system
My ideas
3D gardening game

Computional methods

There are many different computational methods that I can apply to this project. The main reasoning for computational methods is the use of randomly generated landscape, multiple player environments and plant growth mechanics.

For example, this game would not work well as a board game, since it would be hard to physically create random landscapes. Furthermore, a lot of the processes will be performed automatically for the player if they use computational methods, whereas as a board game they would have to keep track of variables themselves or roll some dice to get random numbers.

Thinking abstractly

Using abstraction allows me to identify the necessary and unnecessary details. I decided on low poly graphics for the game since it does not take anything away from the user experience and is a very aesthetic and colourful art style. Furthermore, it takes away the task of processing complex shapes or textures, which optimizes the game performance. Along with this, I decided upon a very simple opening screen and game over screen, to ensure that every function is very straightforward and easy for the user to understand. I think that I will continue this trend throughout the game, such as within the inventory system in which I would like to include simple icons. Finally, I made the decision to not include any realistic variables into gameplay, such as player hunger, thirst or fatigue. This is because the game should focus on collecting, planting and caring for plants and I believe these player variables are out of scope and unnecessary.

Thinking ahead

The game worlds that I will include in my project include the following: an opening screen, a menu screen, a high-score screen, a setting screen, a save/load screen, a gameplay screen and a game over screen. The input mechanisms will be as follows: W for forward, S for backwards, A for left, D for right, Space for jump, E to interact and ESC to access menu. The reasoning behind this is due to the current gaming climate, since these controls are commonly used across various games, which makes playing the game very accessible and simple for the user.

Thinking procedurally and logically

Within this project, there are a few core mechanics or coded routines. They are as follows: moving the player, generating random landscapes, generating random plants, randomly placing plants and trees into the landscape, a time loop mechanic, a watering system, an inventory system, a growth system, a point system, an upgrade system, animals roaming the landscape, a score system, playing appropriate sounds and an end goal to work towards.

Thinking concurrently

Many of the mechanics in this game will occur simultaneously. At the start of each time loop, a random landscape will be generated, along with generating the random plants, and placing them into the landscape along with trees. The growth and watering systems only need to be active when the user is in their greenhouse, adding the time of their exploration to required variables every time they come back. Furthermore, the point and upgrade system only needs to be active when the user requests it to in the greenhouse. Other than this, all the other mechanics will be running simultaneously, such as sound effects playing over background music, or the character controller running alongside the algorithms controlling animal movement.

Specifying hardware and software requirements

Development

The system that I will be developing the game on will be running Windows 10 with 32GB of RAM, X64 3.8Ghz processor and 50GB of SSD space available. These specs fit the requirements for the software listed below.

Unity version 2021.3

I will be using the Unity game engine to develop the game and all its assets.

The reasons for using Unity are as follows:

- Effective implementation of 3D game development
- Extremely helpful and active community
- Integration with C# programming language
- Cross platform releases
- The large asset store with many free assets
- Flexible licensing

Operating system version	Windows 7 (SP1+) Windows 10 Windows 11 64-bit versions only
CPU	X64 architecture with SSE2 instruction set support
Graphics API	DX10, DX11, and DX12-capable GPUs
Additional requirements	Hardware vendor officially supported drivers

Microsoft Visual Studio 2022 version 17.2

I will be running Visual Studio to develop my code and programming for the game.

The reasons for using Visual Studio are as follows:

- Easy and convenient integration with Unity
- Includes a very effective C# programming language plugin
- The IDE includes colour, structuring and code highlighting.
- Helpful and effective debugger
- Flexible licensing

Supported operating systems	Windows 11 Windows 10 Windows Server 2022 Windows Server 2019 Windows Server 2016

Hardware	1.8 GHz or faster 64-bit processor; Quad-core or better recommended. ARM processors are not supported. Minimum of 4 GB of RAM. Hard disk space: Minimum of 850 MB up to 210 GB of available space. Video card that supports a minimum display resolution of WXGA (1366 by 768);
Additional Requirements and Guidance	Administrator rights are required to install or update Visual Studio. .NET Framework 4.5.2 or above is required to install Visual Studio. Visual Studio requires .NET Framework 4.8 to run. If .NET Framework 4.8 isn't already installed, it will be installed during setup. The WebView2 runtime is required to install Visual Studio. If it isn't already installed, it will be installed during setup.

Deployment

The user will require certain hardware and software to run the game.

System hardware	1.8GHz X64 processor minimum 4GB RAM minimum 2GB secondary storage minimum Video card supporting minimum resolution of 1366 x 768 Monitor or display
System software	Hardware vendor officially supported drivers.
Input hardware	Mouse Keyboard

Success criteria

There are certain criteria that the game must meet for it to be a success.

Number	Criteria	Justification
1	The opening screen must allow the user to create a new game, load a previous game save or access the settings menu.	This opening screen lets players jump right into the game, or even change any settings before they begin.
2	The save/load menu must allow users to save their current game or load their past save.	Players must be able to save past games so that they can play over multiple sittings. They might also want to start again with a new game.
3	Settings menu that allows the user to change the resolution and change the game volume.	Players may have different sized monitors or might want a certain level of volume when playing the game and should be able to change these options.
4	Pause menu that the user can access when playing the game.	Allows users to save/load their game, access the options page, return to the opening screen or resume their game.
5	Player movement in three dimensions, including the ability to jump.	This ensures that the user can move around the map however they would like.
6	Mouse control to move the character camera.	This ensures that the player can look up and down and change the character direction.
7	System that allows the user to explore the landscape and collect plants.	This is the main gameplay loop within this game.
8	10-minute time loop when exploring.	This adds a time-based pressure to the gameplay loop.
9	Display the current score on the user's screen when they are exploring.	This ensures that the player knows how many plants they have collected
10	A high score menu that the player can view to see previous top scores.	This lets the player see their past scores so they can see their progression and compete against themselves.
11	Player increases score every time they collect a plant when exploring.	This gives the player incentive to collect lots of plants.
12	Generate a random landscape every time the player explores.	This mechanic adds variation to the game every time the user plays.
13	Generate random plants with different rarities every time the player explores.	This adds variation to the game and makes the player prioritise the rarer plants.
14	Skybox that wraps around the landscape.	This ensures that the landscape is complete and aesthetic for the user.
15	House the player can return to after exploring.	Allows the user to grow and water their plants.
16	System that ages the plants every time the user leaves their house.	This ensures that the player can visually see progression when they come back from exploring.
17	System that simulates plant water intake.	This adds a time-based pressure to the game since the player will have to find water when exploring, then water plants when they return.
18	Map that the user can access while exploring	Allows the user to see a top-down view of the generated landscape. However, the player

		will not be able to see their current position since it would take challenge out of the game.
19	Animals that randomly roam the landscape.	Adds atmosphere to the game, making it more interesting for the player.
20	Inventory system that allows the user to see what plants they have collected on their journey and the rarity, and discard any they don't want.	Helps the player to prioritise and manage their inventory.
21	System that allows players to convert grown plants into points.	Adds a reason to grow rarer plants and sufficiently water them.
22	Upgrade system that allows players spend points to upgrade their tools for faster use times.	Creates a game loop and an incentive to collect plants and play the game.
23	An end goal that the player can work towards, perhaps bought with points.	Ensures that players have always got some incentive to play, so they can finish the game.
24	Game over screen that displays when the user gets lost.	This will make the user lose their current items, creating an incentive not to get lost.

Potential limitations

Below I discuss certain limitations that I consider not worth development time.

Limitation	Justification
Complex object models	Having to create complex object models, such as flowers for the plant objects, will require a lot of learning, particularly in Blender. This would take a lot of time, and since the change is purely cosmetic it is not worth the investment.
Infinite landscape	Whilst this feature would be a very interesting implementation, it would require skills that are beyond my current knowledge. Therefore, it would take a lot of time to implement, and would be better to just use a large landscape.
Player model/animations	The game that I will make is first person, therefore the user will never see the player model unless they look at their shadow. Since this is quite a minor aspect of the game, this would be a relative waste of time.
Swimming movement	This feature would make the game more realistic; however, it would take some time to actually implement water and the physics surrounding that, let alone the movement mechanics when the player is in the water. I do not think it is worth the time investment.
3D grass texture	Whilst this feature would add atmosphere to the game, I think that the hit to performance and the resources/models required to implement such a feature would not be worth the benefits, which are purely cosmetic.
Graphical raytracing	This feature would be extremely resource heavy and would also require a lot of learning on my end. It would reduce the amount of people who could play the game, and thus will not be included.
Weather systems	The idea of having a changing environment, with different weathers and time zones would add a lot to the game's atmosphere and variance. However, this would take an extremely long time to implement, and therefore I have decided it is a limitation.

Design methodology

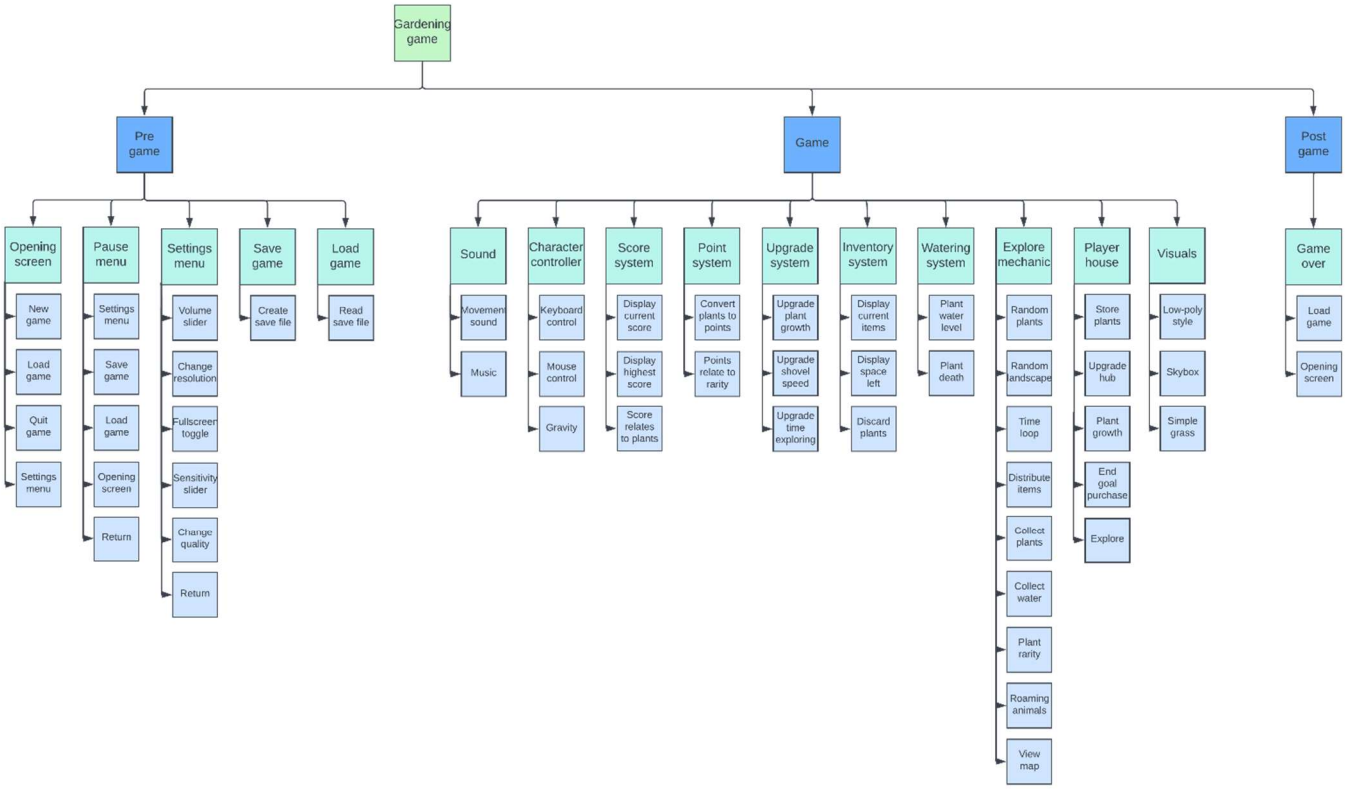
Before I begin work on this project, I need to pick a design methodology that I will follow when designing, developing, evaluating and maintaining this project. I decided that following an agile methodology is the best option here. I will split my project into milestones, a form of decomposition, breaking down my project into smaller pieces of work. When I have finished, these pieces of work will join together to form the overall solution.

In action this will consist of designing a milestone, developing a milestone and then evaluating said milestone. Once each iteration completes, I will move onto the next milestone. I have decided to tackle maintenance once I have completed all the milestones but will keep it in mind in the evaluation stage of this methodology.

Design

Systems diagram

Below is my proposed systems diagram for this project.



This top-down module design allows me to break down the problem into easier to manage subproblems. This ties into computational methods such as “thinking ahead”, allowing me to understand visually the different game worlds along with the different inputs and outputs. Additionally, it also shows the “thinking procedurally and logically” method by identifying the core routines that are required so the game can function. This systematic approach and decomposition of the problem is extremely helpful, giving my working solution for the problem structure.

Milestones

Below is a list of development milestones that I need to complete to finish my project.

Number	Name
1	Player movement
2	Basic user interface
3	Basic game loop
4	Plants and water
5	Growing and coins
6	Pause menu, save and load, music
7	Polishing and feedback

[1] Milestone

Below I have outlined all the steps that I need to take within this development milestone.

Name	Modules	Algorithms	Justification
Player movement **1**	Character controller: ✓ Keyboard control ✓ Mouse control ✓ Gravity	 The keys “W”, “A”, “S”, “D” should control the x-axis and z-axis of the character object. The mouse input should control the character camera. The “SPACE” key should increase the character’s positive vertical velocity, and a gravity value should bring the character back to the floor.	This should be the first step in my development since it is the most core mechanic of this game. The player needs to be able to move the character around.

[1] Character controller

Date: 20/07/22

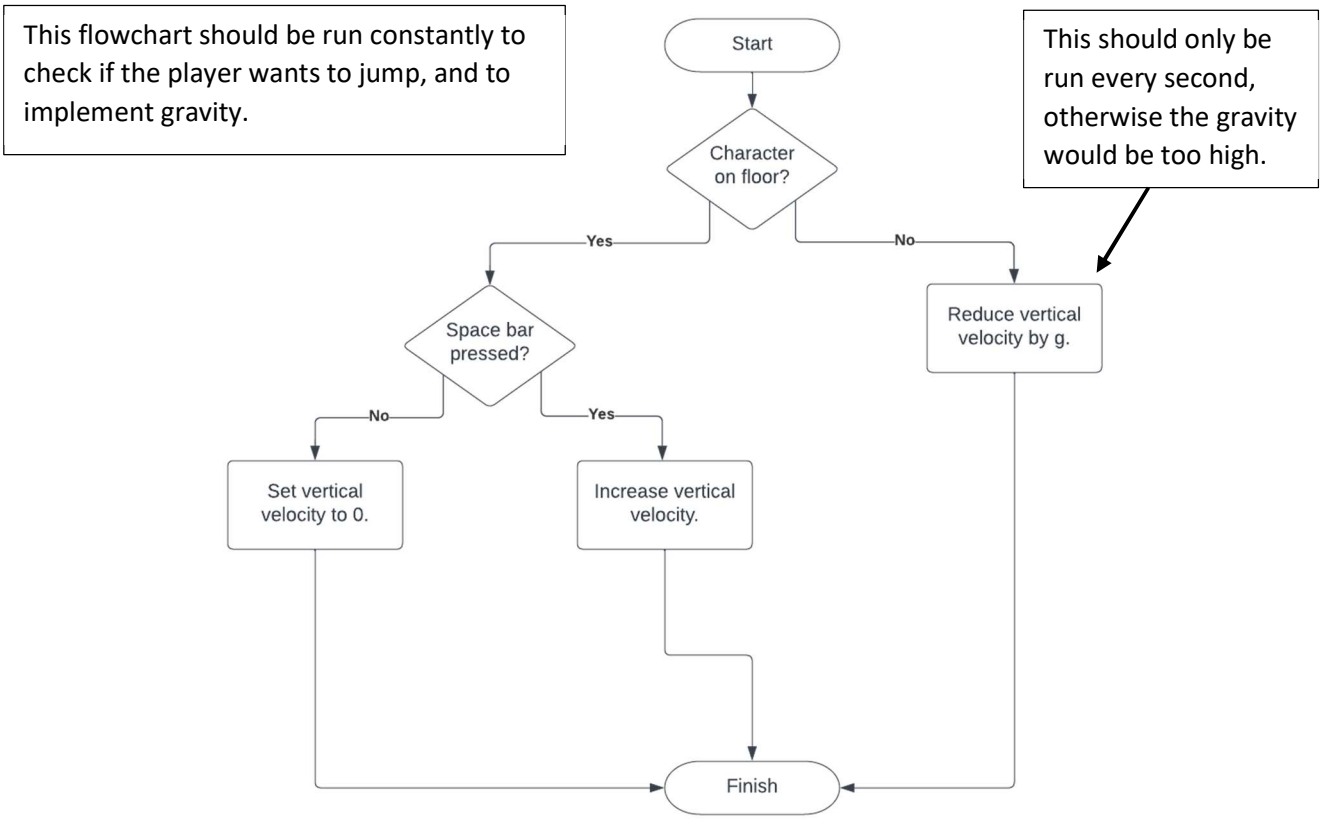
Below is pseudocode for how I will move the player using the character controller.

```
function playerMove(player, mouseX, mouseY, key)

        # mouse input
        player.direction.x = player.direction.x + mouseX
        player.direction.y = player.direction.y + mouseY

        # keyboard input
        switch key:
                case “W”:
                        player.position = player.position + player.direction.front
                case “A”:
                        player.position = player.position + player.direction.left
                case “S”:
                        player.position = player.position + player.direction.back
                case “D”:
                        player.position = player.position + player.direction.right
                default:
                        continue
        endswitch
endwhile
```

This should add the mouse movement to the player

This part of the program should move the player in the respective direction, according to the user

Below is a flow chart of how I will include gravity and jumping mechanics in my solution.



[1] Usability features

Date: 22/07/22

In this milestone, there were not any interfaces for me to consider, so there is nothing to discuss in this part of usability.

However, in terms of player experience, there are a few things to consider. The use of “W”, “A”, “S”, “D” and Space keys for character movement improves the player experience since these are very common character movement keys and the player will automatically assume that these are the keys they should use. This makes the need for a tutorial redundant.

[1] Class diagram

Date: 23/07/22

Below I have made diagrams for all classes within this milestone.

Character
x : float y : float z : float directionX : float directionY : float directionZ : float

speed : float mouseSensitivity : float slopeSteepness : float stepHeight : float gravity: float touchingFloor: bool
start(self) update(self) playerLook(self, mouseInput) playerMove(self, directionInput) playerJump(self)

[1] Variables

Date: 24/07/22

Variable	Data type	Purpose, justification and validation
Character.x Character.y Character.z	Float	Stores the position of the player in a 3D world. These will be converted to a single Vector3 to allow for easy manipulation of the player's position.
Character.directionX Character.directionY Character.directionZ	Float	Stores the direction that the player is looking. These will be converted to an Euler angle to allow for easy manipulation of the direction the player is looking.
Character.speed	Float	This variable will store a value which determines the player speed. I have decided to use a float so that I can fine tune this value. Once I have decided on a final value, I can treat this as a constant.
Character.mouseSensitivity	Float	This will act as a multiplier when manipulating the direction that the player is looking. Therefore, it will control the player's mouse sensitivity. A float is used because later in development it will be controlled by a slider, which returns a float value.
Character.slopeSteepness	Int	This will store the maximum angle that the player can walk up. Since it is storing an angle, I will be using an integer data type.
Character.stepHeight	Float	This will store the maximum height that the player can step up. This will be stored as a float value so that it can be fine-tuned.
Character.gravity	Double	This will store the g value for the project. It will most likely be 9.81, hence why I am storing it as a double.
Character.touchingFloor	Bool	This will store whether the player is touching the floor or not. This will be used to determine whether the player can jump or not.

[1] Files

There will be no use of external files in this milestone.

[1] Encoding

There will be no use of encoding in this milestone.

[1] Validation

Later in development, the player will be able to move around a very large world. Therefore, I do not need to make any location checks, testing whether they have left the map. Instead, the map will be so big that they cannot leave the map in time, before game over. This also gives them a sense of freedom when roaming around.

[1] Test data

Date: 25/07/22

Test number	What is being tested and inputs	Expected outputs	Justification
1	The player can look up and down by moving their mouse up and down.	The camera rotates up and down. It locks out at a certain distance up and down.	The player needs to be able to look up and down with their mouse so they can look around the landscape.
2	The player can look left and right by moving their mouse left and right.	The character body rotates left and right, changing direction.	The player needs to be able to look left and right with their mouse so they can direct their character.
3	The player can move forwards and backwards with the “W” and “S” keys.	The character moves forwards and backwards relative to direction.	The player needs to be able to walk forwards and backwards so they can explore and collect plants.
4	The player can move left and right with the “A” and “D” keys.	The character moves left and right relative to direction.	The player needs to be able to walk left and right so they can explore and collect plants.
5	The player can jump up with the “SPACE” key.	The character moves up into the air, then falls back down due to gravity. The gravity is an appropriate level.	The player needs to be able to jump and fall down mountains and hills.
6	The player can walk up steps.	The character moves up steps of certain height, by shifting up.	The player needs to be able to walk around rocks and obstacles, so the character controller needs to be tested with steps.
7	The player can walk up slopes.	The character walks up slopes of up to 45 degrees.	The player needs to be able to walk up mountains and hills, so the character controller needs to allow for walking up slopes.

[2] Milestone

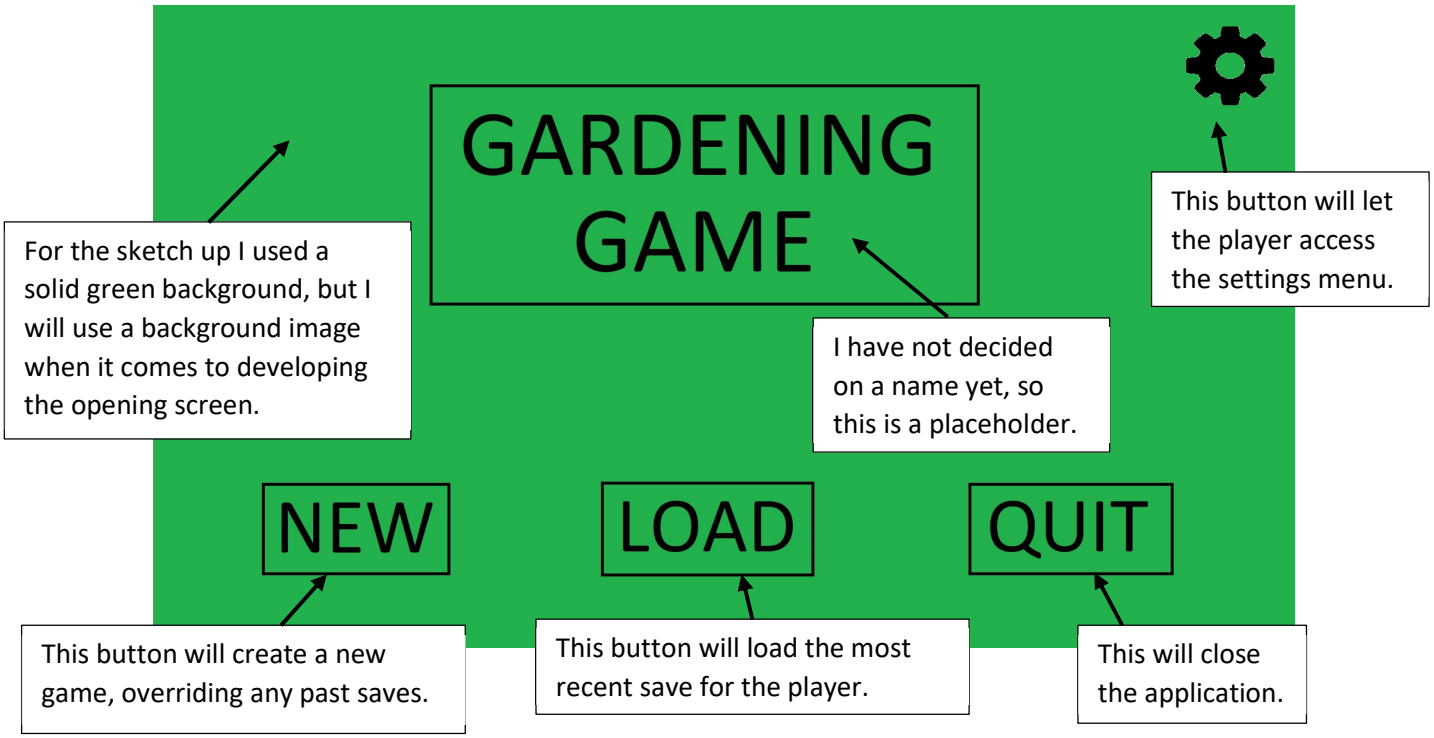
Below I have outlined all the steps that I need to take within this development milestone.

Name	Modules	Algorithms	Justification
Basic user interface **2**	Opening screen: ✓ New game ✓ Load game ✓ Quit game ✓ Settings menu Settings menu: ✓ Volume slider ✓ Change resolution ✓ Fullscreen toggle ✓ Sensitivity slider ✓ Change quality ✓ Return	This button should delete any saves, and then initialise the game. This button should load the latest save, then initialise the game. This button should exit the application. This button should open the settings menu. This slider should change the game's master volume. This drop down should change the game resolution. Should check what is resolutions the host machine has. This toggle should switch the game between fullscreen and windowed. This slider should change the game's sensitivity. This drop down should change the game quality. This button should return the user to their previous scene.	The next step should be the basic user interface. The settings options need to be considered for the rest of development, so it is best to do it near the beginning. The settings menu cannot be accessed without the opening screen or pause menu, so they opening screen should be developed too.

[2] Opening screen

Date: 12/08/22

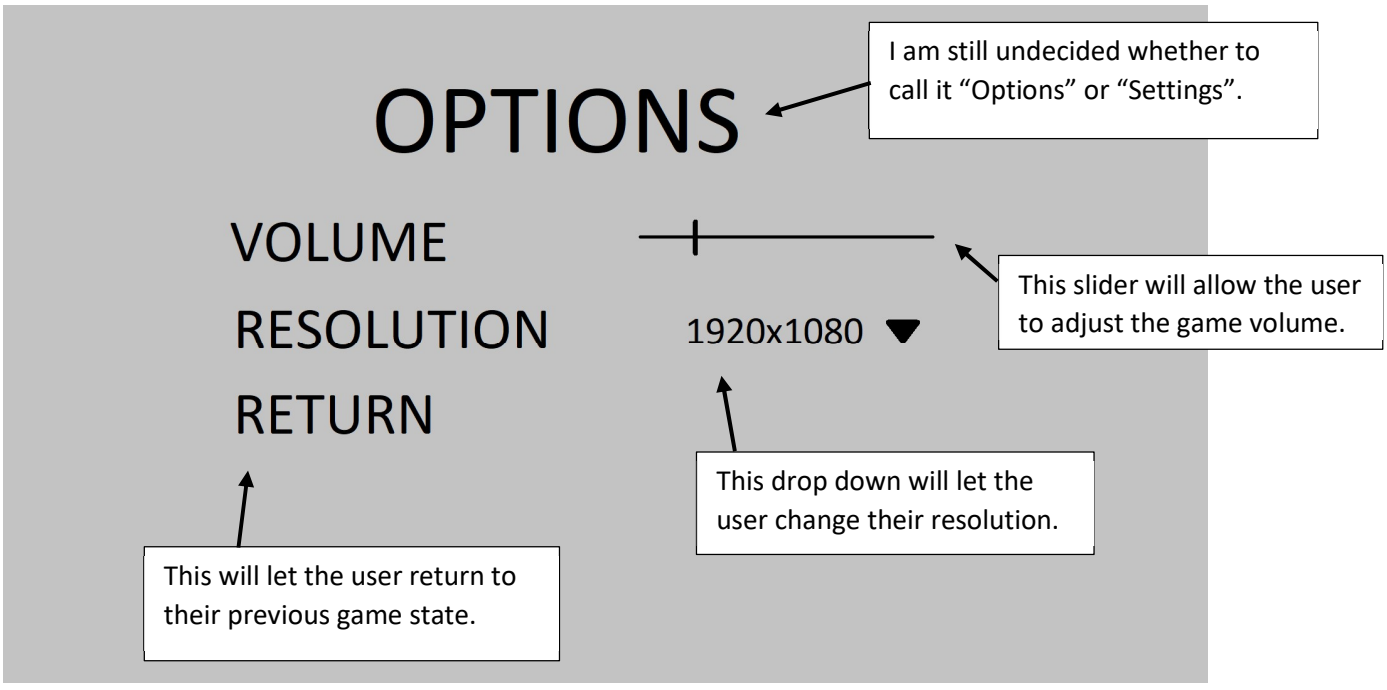
Below is a sketch up of what I would like my opening screen to look like.



[2] Settings menu

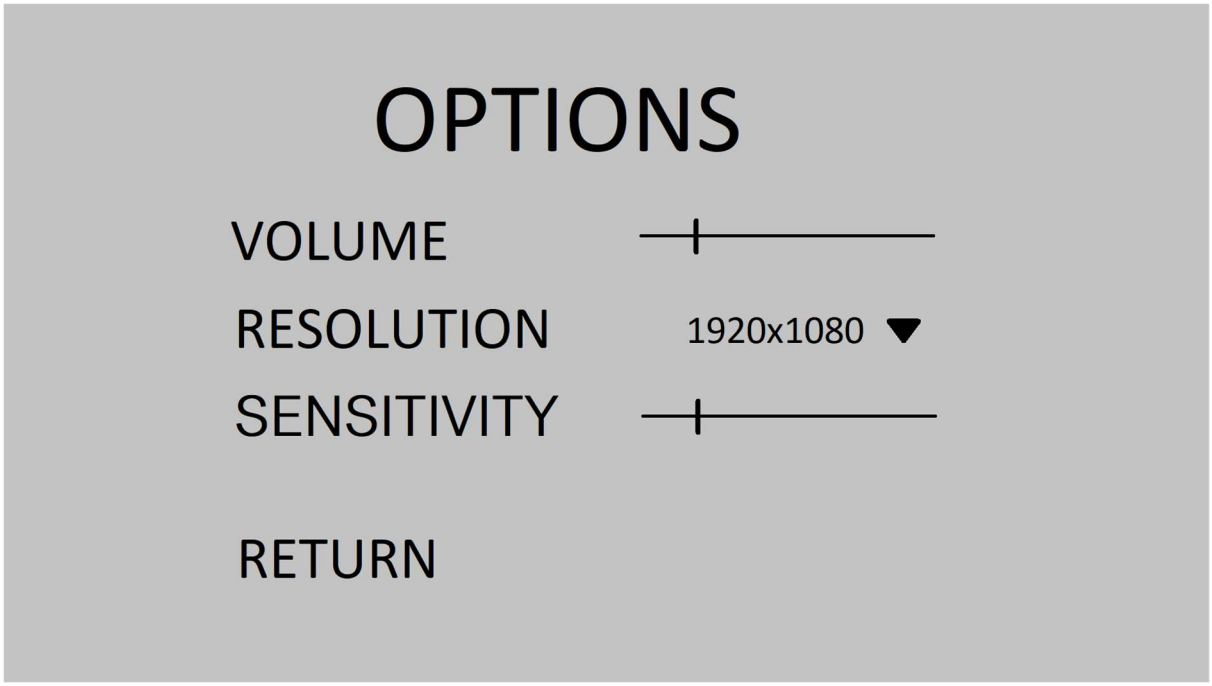
Date: 12/08/22

Below is a sketch up of what I would like the settings menu to look like.



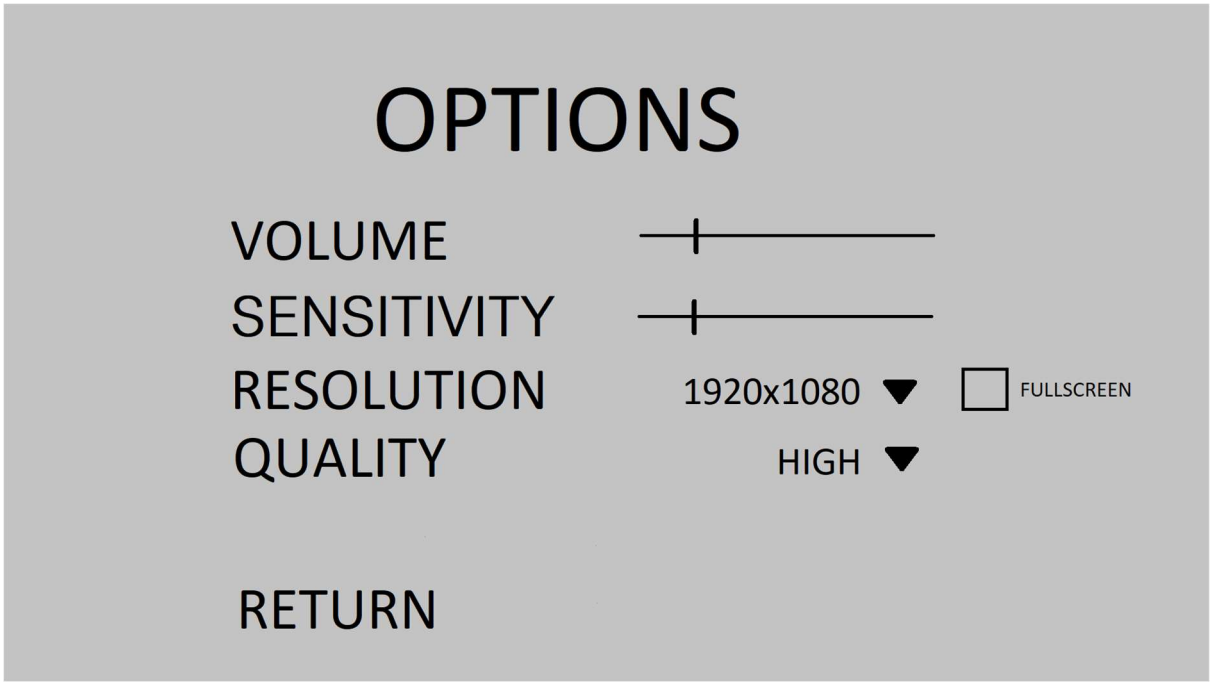
Date: 13/08/22

After reviewing this menu, I realised that the user needs to be able to change their game sensitivity. Below is an options menu that is representative of these changes.



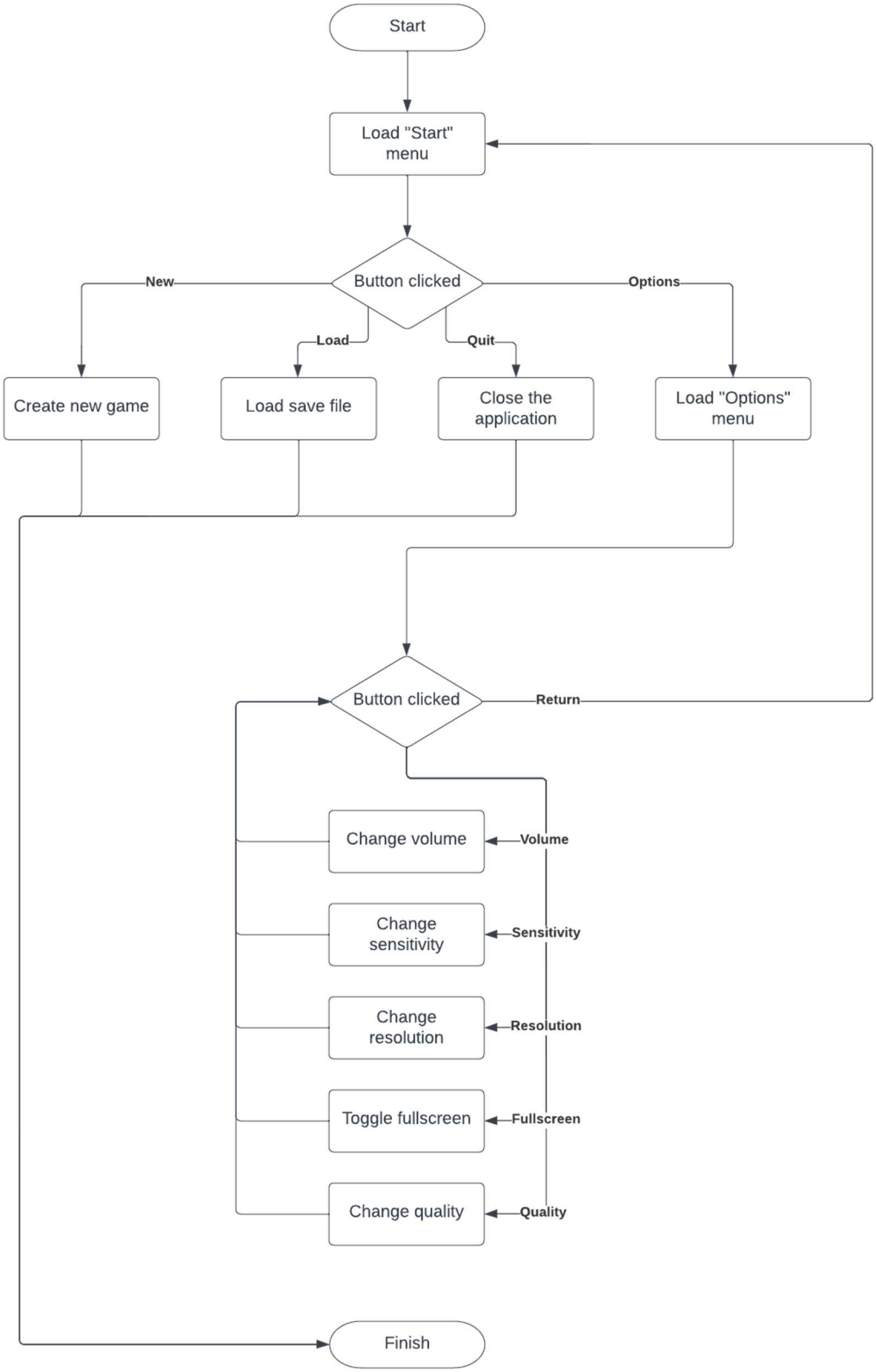
Date: 14/08/22

Once I reviewed this menu again, I saw that the sliders and drop downs could be shuffled around a bit, and that I needed to include a quality drop down and full screen check box.



Date: 15/08/22

At this point, I would like to outline a flowchart that I can refer to for the whole menu system. This can be seen below.



[2] Usability features

Date: 16/08/22

In terms of interface, there are many features that I have included in this section that are very relevant. The decision to use big and simple buttons was made to ensure that the menus were clear and readable. This might help users who have accessibility issues, such as bad vision. I wanted to make the menus very intuitive and easy to use. The black outlining used in the menus may be helpful to colour blind users.

There is not much to discuss here in terms of player experience.

[2] Class diagram

Date: 17/08/22

Below I have made diagrams for all classes within this milestone.

Main menu
This class has no variables.
new(self) load(self, save) quit(self) options(self)

Options menu
volume : float sensitivity : float resolution : string fullscreen : bool quality : string
volume(self, newVolume) sensitivity(self, newSensitivity) resolution(self, newResolution) fullscreen(self, newFullscreen) quality(self, newQuality) return(self)

[2] Variables

Date: 18/08/22

Variable	Data type	Purpose, justification and validation
Options.volume	Float	Stores the game's master volume. This will be changed with a slider, hence why it is stored as a float. The value should be pushed to the volume mixer when changed.
Options.sensitivity	Float	This value will store the mouse sensitivity when it is changed by the user. The value should be pushed to a global variable so that it affects gameplay.

Options.resolution	String	This string will store the resolution selected by the user. This will be pushed to the game settings during runtime so that it takes affect when the player changes their selection.
Options.fullscreen	Bool	This will allow the user to toggle fullscreen. This will be stored as a bool and will be pushed to the game settings during runtime, so the changes take effect when the user toggles the checkbox.
Options.quality	String	This string will store the user's selection of the game quality. This will be pushed to the game settings during runtime, so the changes take effect when the user makes their selection.

[2] Files

There will be no use of external files in this milestone.

[2] Encoding

There will be no use of encoding in this milestone.

[2] Validation

The menu system that I have implemented is a closed system. All pages have return functions that allow players to exit out of the current menu. There is no way that the user can get lost in the menus or get trapped without being able to exit. This design aspect means that I do not need to implement any further validation within the coded solution.

[2] Test data

Test number	What is being tested and inputs	Expected outputs	Justification
1	The player can access a start screen when the game is run.	The "Start" scene is loaded.	The player needs to be able to access the start screen, it should be the first thing they see.
2	The player can load into a new game by clicking the "New" button.	The "Inside" scene is loaded.	The player needs to be able to create a new game, and this feature needs to be tested.
3	The player can quit the application by clicking the "Quit" button.	The application closes.	This should be tested because the player needs to be able to quit the game safely.
4	The player can access an options menu by clicking the "Options" button.	The "Options" scene is loaded.	The player needs to be able to easily switch between menus, so this should be tested.
5	The player can change the volume by moving the "Volume" slider.	The master game volume changes and persists.	The player should be able to fine tune their game volume, and the feature should be tested.

6	The player can change the mouse sensitivity by moving the “Sensitivity” slider.	The mouse sensitivity changes and persists.	The player needs to be able to fine tune their mouse sensitivity to their liking, so I should test if this is possible.
7	The player can change the resolution by selecting a resolution from the “Resolution” drop down menu.	The game resolution changes to the selected resolution and persists.	The target audience may have different resolutions on their own computers, so I should test this feature and ensure it works well.
8	The player can toggle the game fullscreen or windowed.	The game becomes fullscreen or windowed and persists.	The player may prefer to play their game windowed or in fullscreen, so I need to test whether this implementation is robust.
9	The player can change the game quality by selecting a quality from the “Quality” dropdown.	The game becomes the quality that the user selected and persists.	The target audience will have a variance of computer resources, so they should be able to change the game quality to their liking. I should test this and ensure it works well.
10	The player can return to the start screen when clicking the “Return” button.	The “Start” scene is loaded.	The player needs to be able to easily navigate the menu, so I should test the robustness of the menu system.

[3] Milestone

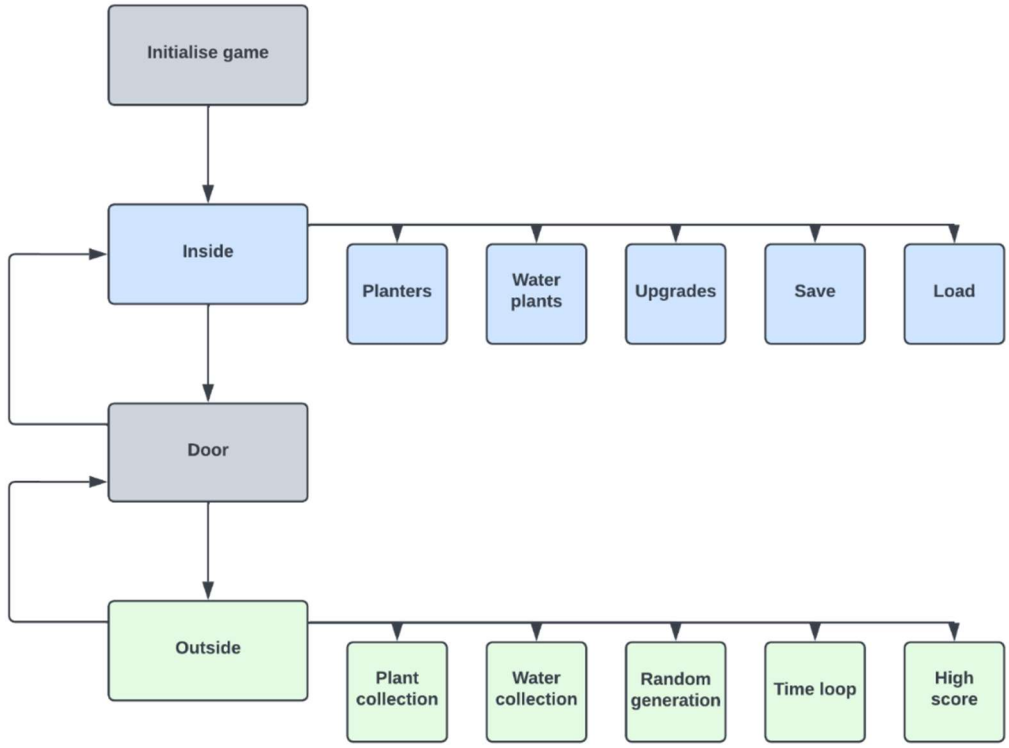
Below I have outlined all the steps that I need to take within this development milestone.

Name	Modules	Algorithms	Justification
Basic game loop **3**	Player house: ✓ Explore ✓ Store plants Game over: ✓ Load game ✓ Opening screen Explore mechanic: ✓ Random landscape ✓ Time loop	 The player should be able to walk towards the door, and once the character collides with the door object, the player should be moved into a new scene where the explore mechanic begins. The house should include planters where plants can be stored. This button should load the latest save, then initialise the game. This button should return the user to the opening screen. Perlin noise should be used with octaves to generate a realistic landscape surrounding the player house. A gradient might need to be added to ensure the player house can sit flat on the landscape. A timer should be started when the explore mechanic starts, and if the player is still exploring once 10 minutes is over, the game over scene should be loaded.	This step should be developed next to create the basic skeleton of the game. This consists of the basic game loop mechanics. At a later point in development, I can add to this basic skeleton.

[3] Player house

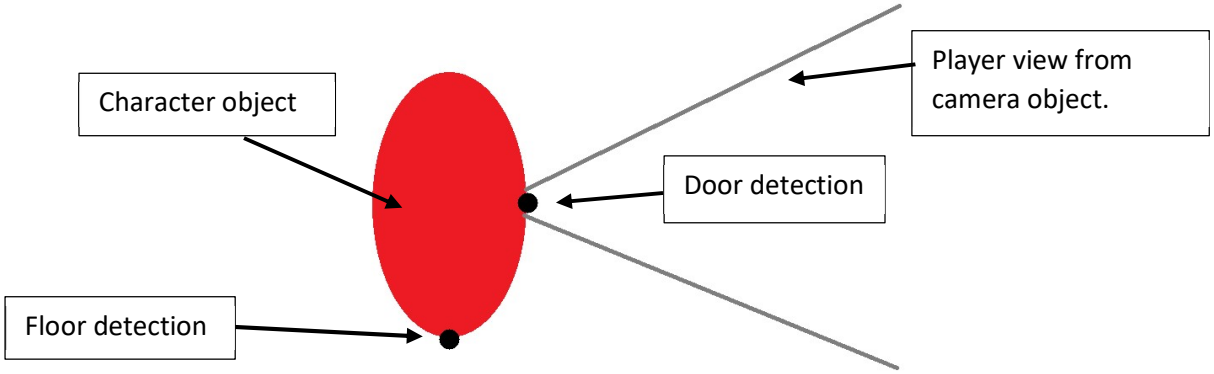
Date: 20/09/22

This diagram was made using abstraction to simplify my game from the user's perspective, focusing on the core mechanics and the game loop.



Date: 10/11/22

In terms of developing this solution, I was planning on creating an object on the front of the character object, similar to that used in the character controller to detect when the character touches the floor. This object will instead detect when the character touches the door and will then change the scene from inside to outside, and vice versa.



I decided that this would be the most convenient implementation of interaction mechanics for the player. It is simple and intuitive. All the player has to do is walk into the door to activate the mechanic. This implementation also ensures that they walk into the door whilst facing the door and cannot activate it by backing into the door. This means that they made the active decision to go through the door. The convenience of this design has made me rethink how I will implement other kinds of player interaction later on in the game, such as picking up flowers and accessing upgrades.

Below is some pseudocode for this section of development. The aim of this piece of code is to detect if the user is touching a door object, and if so, the scene should change.

```
procedure doordetect():
        # check if the character object is touching a door object
        if character touching door:
                # switch the scenes if it is
                if scene = "outside":
                        scene = "inside"
                else if scene = "inside":
                        scene = "outside"
                end if
        end if
end procedure
```

Here I will use the door detection object to determine whether the character is touching a door object.

I can use Unity's inbuilt layering system to label all door object with a "Door" layer.

[3] Game over

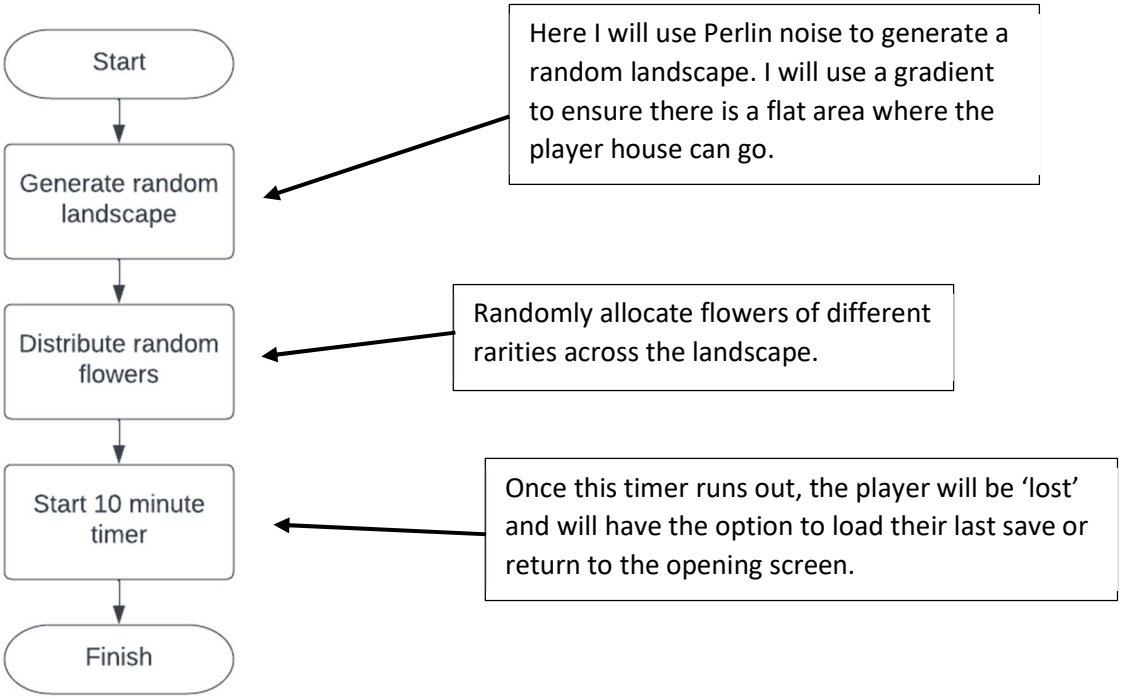
Date: 12/08/22

Below is a sketch up of what I would like the game over screen to look like.

[3] Explore mechanic

Date: 10/11/22

Below is a flowchart of the processes that need to be completed when the explore mechanic begins.



Date: 23/11/22

Below is some pseudocode I will use for the timer. I researched how to create a timer in Unity[14].

```
procedure timer(timeLeft):
        if timeLeft > 0:
                timeLeft = timeLeft - Unity.TimePassed
                Unity.Object("Timer") = timeLeft
        else:
                Unity.SetScene("Game over")
        end if
end procedure
```

This is quite abstracted. I will need to convert the raw time value to minutes and seconds, and then display this on a screen as a Unity object.

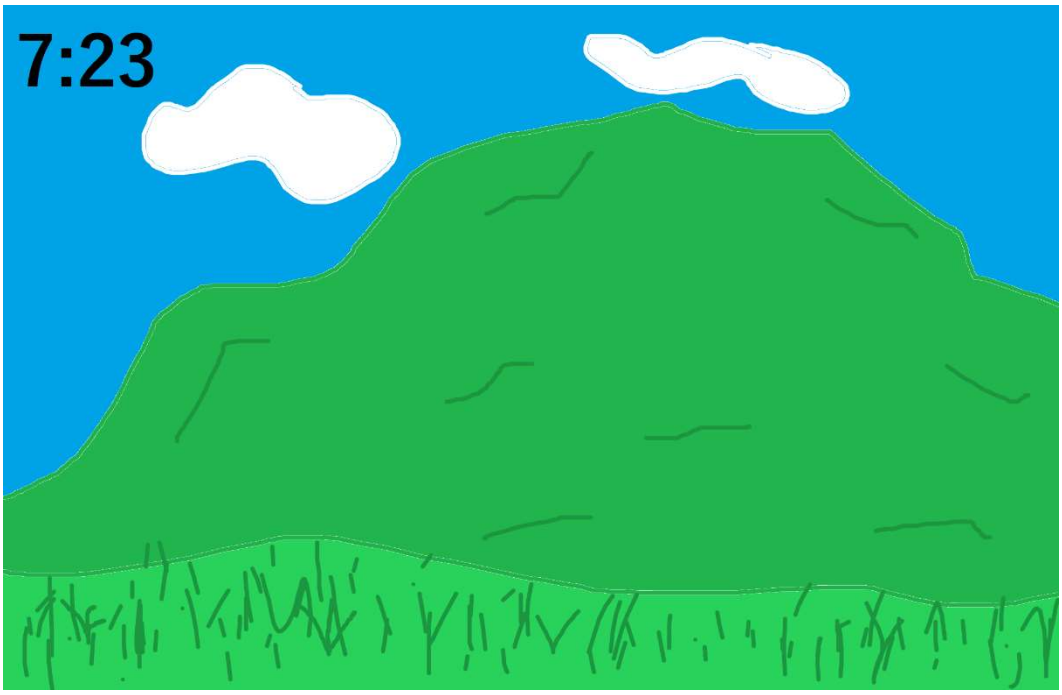
Once the timer is up, the scene will change to "Game over".

I decided that for now, I will display the time remaining on the user's screen. This will help me debug, and I can change it later in development, replacing it with sounds as decided in the analysis phase.

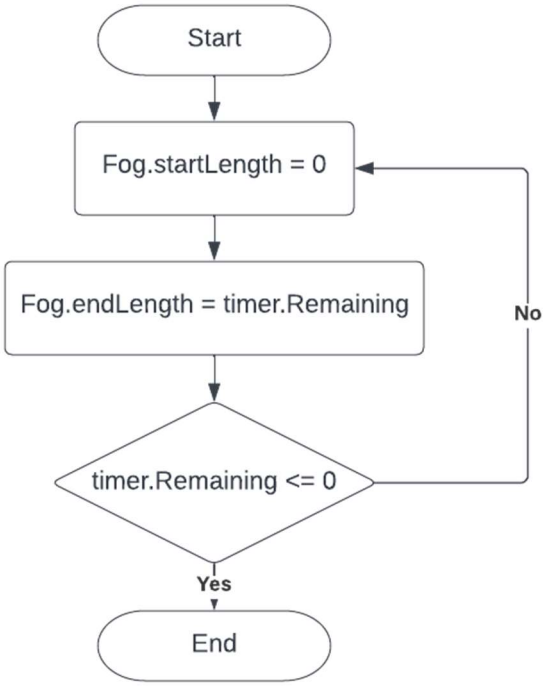
[14] (Demirçin 2022)

Date: 04/12/22

Here, I have designed where the timer will be displayed on the user screen. This positioning may change in the future, or the timer may be removed all together.



Another feature that I may include would be to set the fog end distance to the time remaining in seconds. This would mean that the fog would slowly creep in as the timer reduced. This would add a new level of difficulty to the game, and it would be a challenge not to get lost.



Date: 05/12/22

Here, I need to design a script that can generate an array of randomly generated heights, which can be pushed onto a terrain object in Unity, creating a random landscape using Perlin noise.

Below is the pseudocode for this problem.

```
procedure terrainGeneration():
        x_size = 100
        z_size = 100
        y_size = 20
        scale = 20

        terrain = Unity.Object("Terrain")
        heights = Array[x_size, z_size]

        for i=0 to x_size
                for j=0 to z_size
                        x = (i / x_size) * scale
                        z = (j / z_size) * scale
                        heights[i, j] = PerlinNoise(x, z)
                next j
        next i

        terrain.size = (x_size, z_size, y_size)
        terrain.heights = heights
end procedure
```

Defines the x, z, and y size of the landscape. Here I also set a scale, which I can adjust.

Reference to the terrain object.

Creates a new 2-dimensional array with "x_size" and "y_size".

This loop will go through all the items on the "heights" array, giving each item a new value from the Perlin noise function.

Normalises with x and z size.

Here the "heights" array will be pushed onto the "terrain" object.

Perlin noise is an algorithm used to produce a type of noise that is particularly helpful in the context of terrain generation[15]. The noise produced from this algorithm is very natural looking, so can be used to generate "waves", or in this case I will use it for mountains and hills. Unity has its own Perlin noise function, which outputs a height value for any given (x, z) coordinate. This will help me a lot, since it is quite a complex algorithm and I do not have the time or knowledge to personally implement such an algorithm.

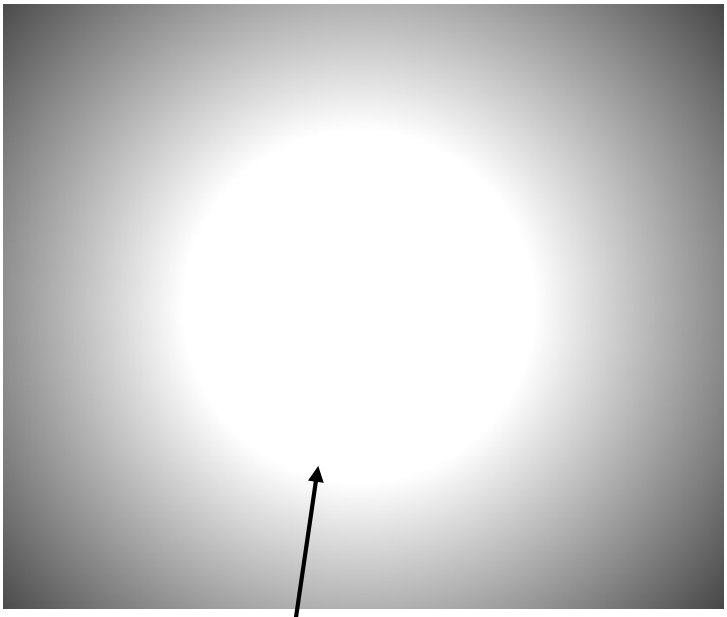
"Octaves" will be used alongside this code. "Octaves" can be seen as repeats, with each repeat the landscape becomes less influential, but with higher frequency. This creates a landscape which has better detail. "Octaves" can be described as creating mountains, then boulders, then rocks, then pebbles.

To achieve better results, I can change the "Amplitude", "Frequency" and "Scale" of the landmass. Tweaking these values will help me find the right landscape. "Amplitude" determines how much strong each "Octave" is. "Frequency" affects the noise frequency, changing levels of fine grain detail. "Scale" determines the distance at which the noise map is viewed from.

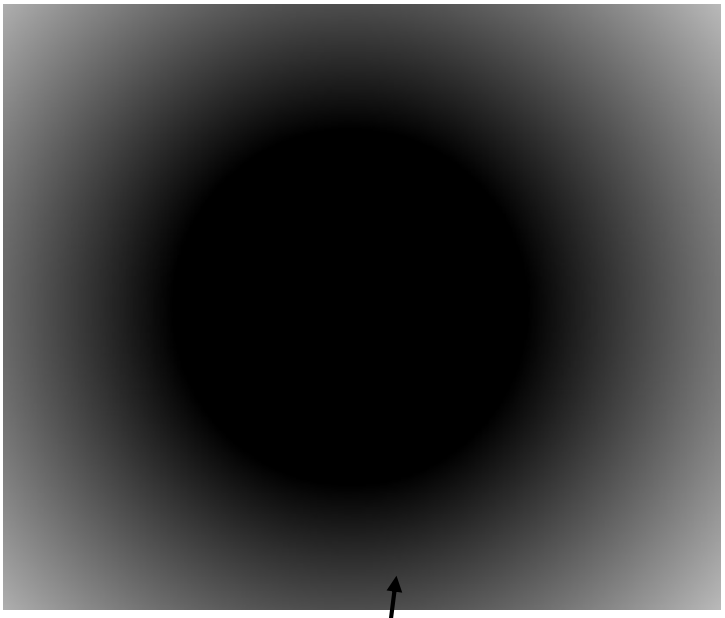
[15] (Flafla2 2014)

Date: 25/11/22

Below I have outlined what sort of gradients I would like to overlay the Perlin noise with to increase the playability of my game. I need to use a combination of two to achieve the required result.



Overlaying a gradient such as this would create a landscape that trails off the further away the player gets from their house.



Overlaying a gradient such as this would create a landscape that is flat around the house, and then becomes normal the further out the player goes.

Using a gradient such as this would be ideal. It would create a flat area near the player house, so there are no problems getting in and out of the house. It would then gradually become normal terrain, and finally trail off if the player gets too far out.

When it comes to deploying this solution, the gradient would be on a much larger scale with smoother transitions between flat and mountainous.

[3] Usability features

In terms of interface, the only feature that comes to mind would be the addition of a timer in the top left of the screen allows the user to easily see and understand the time remaining before they are lost. Although this feature may be removed later in development, it is still a usability feature for now. In addition to this, the link between fog and time remaining in seconds gives the player a visual warning of the time remaining.

On the other hand, in terms of player experience, I have added a few features. Firstly, the landscape is automatically generated when the player walks into the door. This is quite intuitive and easy for the player since they do not have to click any buttons to begin the explore mechanic. They simply just leave their house through the door, just like real life. The use of a gradient, or any other means of positioning the player safely on the landscape, is also another usability feature. The player does not need to worry about glitching through the floor, or any other issues. They will easily be able to walk out the house and begin exploring with no issues.

[3] Class diagram

Date: 26/11/23

Below I have made diagrams for all classes within this milestone.

Terrain
xSize : integer zSize : integer ySize : integer octaves : integer amplitude : float frequency : float scale : float
start(self) generateTerrain(self)

Timer
timeRemaining : integer
start(self) update(self, text)

Lost menu
This class has no variables.
load(self, save) quit(self)

[3] Variables

Date: 27/11/22

Variable	Data type	Purpose, justification and validation
Terrain.xSize	Integer	This will store the width of the terrain. It will be an integer and will be used when creating the terrain.
Terrain.zSize	Integer	This will store the length of the terrain. It will be an integer and will be used when creating the terrain.
Terrain.ySize	Integer	This will store the height of the terrain. It will be an integer and will be used when creating the terrain.
Terrain.octaves	Integer	This stores the number of octaves that will be used during terrain generation. Each octave repeats the generation process, but with less amplitude and higher frequency. This adds detail to the terrain.
Terrain.amplitude	Float	This float will store the current amplitude value during terrain generation. This changes throughout the process, especially when new octaves are run.
Terrain.frequency	Float	This float will store the current frequency value during terrain generation. This changes throughout the process, especially when new octaves are run.
Terrain.scale	Float	This float acts as a multiplier for the terrain as a whole and will be constant during terrain generation.
Timer.timeRemaining	Integer	This value will store the time the player has remaining in seconds. This will be converted into minutes and seconds and displayed in the UI. When this reaches 0, the game over screen will be displayed.

[3] Files

There will be no use of external files in this milestone.

[3] Encoding

There will be no use of encoding in this milestone.

[3] Validation

In this section, there is not much to talk about in terms of validation. The only feature that comes to mind is ensuring that once the “timeRemaining” variable is less than or equal to 0, the game over scene should be loaded. This ensures the player is “lost” once their timer runs out.

[3] Test data

Test number	What is being tested and inputs	Expected outputs	Justification
1	The player can see planters inside their house.	Planter objects should be visible.	These objects are a core part of the gameplay, so I should test that the player can see them clearly.
2	The player can leave their house by walking through the door.	The “Outside” scene is loaded.	This feature will be used a lot by the player, so I need to test

			that it is robust and works well consistently.
3	A random landscape is loaded when the player leaves their house.	Random landscape is produced.	This is one of the core gameplay loops in the game, so it should work well and be tested.
4	The house is positioned correctly on the landscape.	The house sits correctly on the landscape.	The player needs to be able to leave their house easily when they walk through the door, so this should be tested thoroughly.
5	A timer is visible when the player leaves their house.	10:00 can be seen somewhere on the screen.	The player should know how much time they have left before they are lost, so this should be tested to ensure it works correctly.
6	The timer is shown in minutes and seconds and is decreasing.	The timer decreases by 1 second every second.	The speed at which the timer decreases should be accurate, so this needs to be tested.
7	A gradient is applied to the landscape so that it trails off at the end.	The landscape gradually trails off to sea level.	This feature provides a cut off to the landscape, and the shape of the gradient should be checked and tested.
8	A game over screen is displayed when there is no time left on the timer.	The “Game over” scene is loaded.	This is an important and core feature, and I need to check that it works correctly and consistently.

[4] Milestone

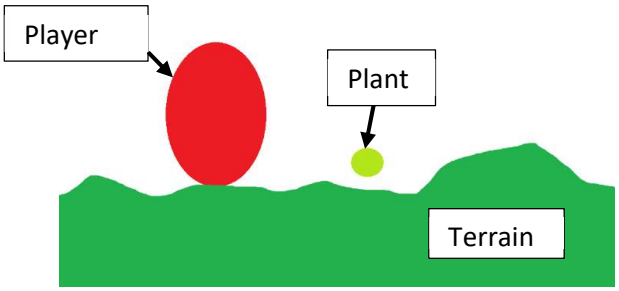
Below I have outlined all the steps that I need to take within this development milestone.

Name	Modules	Algorithms	Justification
Plants and water **4**	Explore mechanic: ✓ Distribute items ✓ Collect plants ✓ Collect water Inventory system: ✓ Display current items Score system: ✓ Display current score ✓ Score relates to plants	 Items should be randomly distributed around the landscape surface, including trees, plants and water objects. Plant objects should be collected when the player walks over the plant. The plant should be removed from the landscape and moved to the player inventory. Water objects should be collected when the player walks over it. The water should be removed from the landscape and moved to the player inventory. The player should be able to access a screen that displays their current items when they press a certain key. The current score should be displayed on the player's screen. Plants should be converted to points in a 1 to 1 ratio.	This step of development should ensure that the player can leave their house and collect plant/water objects, which should be moved into their inventory. A basic score system should be implemented also.

Explore mechanic [4]

Date: 02/12/22

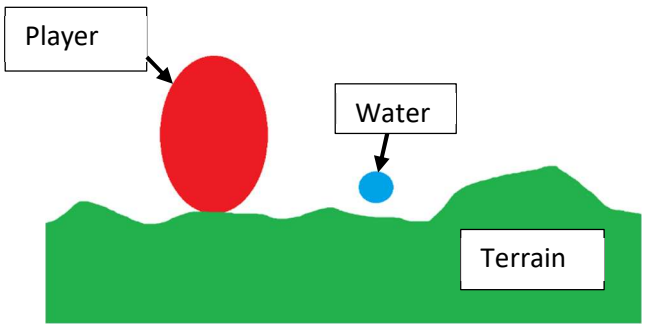
In this subsection of design, I will outline how the player can collect the plants in the landscape.



When the player walks over a plant object, it will be "collected". The plant object should be deleted and moved to the player inventory.

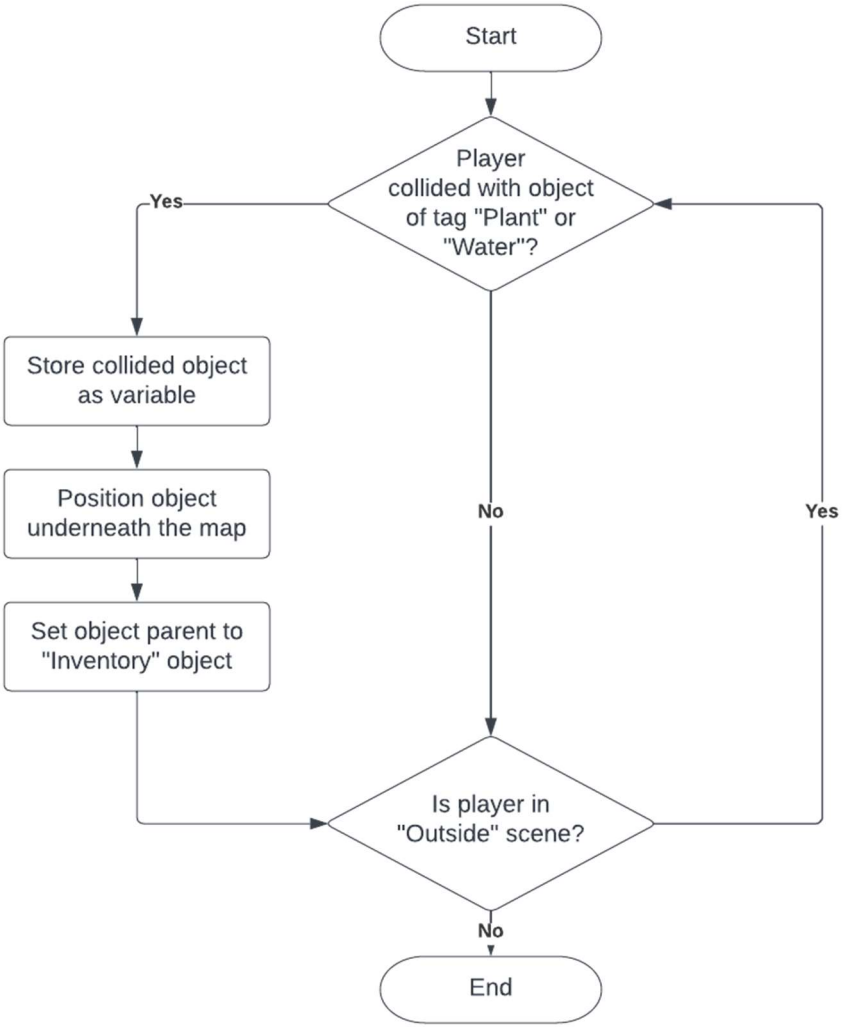
Date: 02/12/22

In this subsection of design, I will outline how the player can collect water in the landscape.



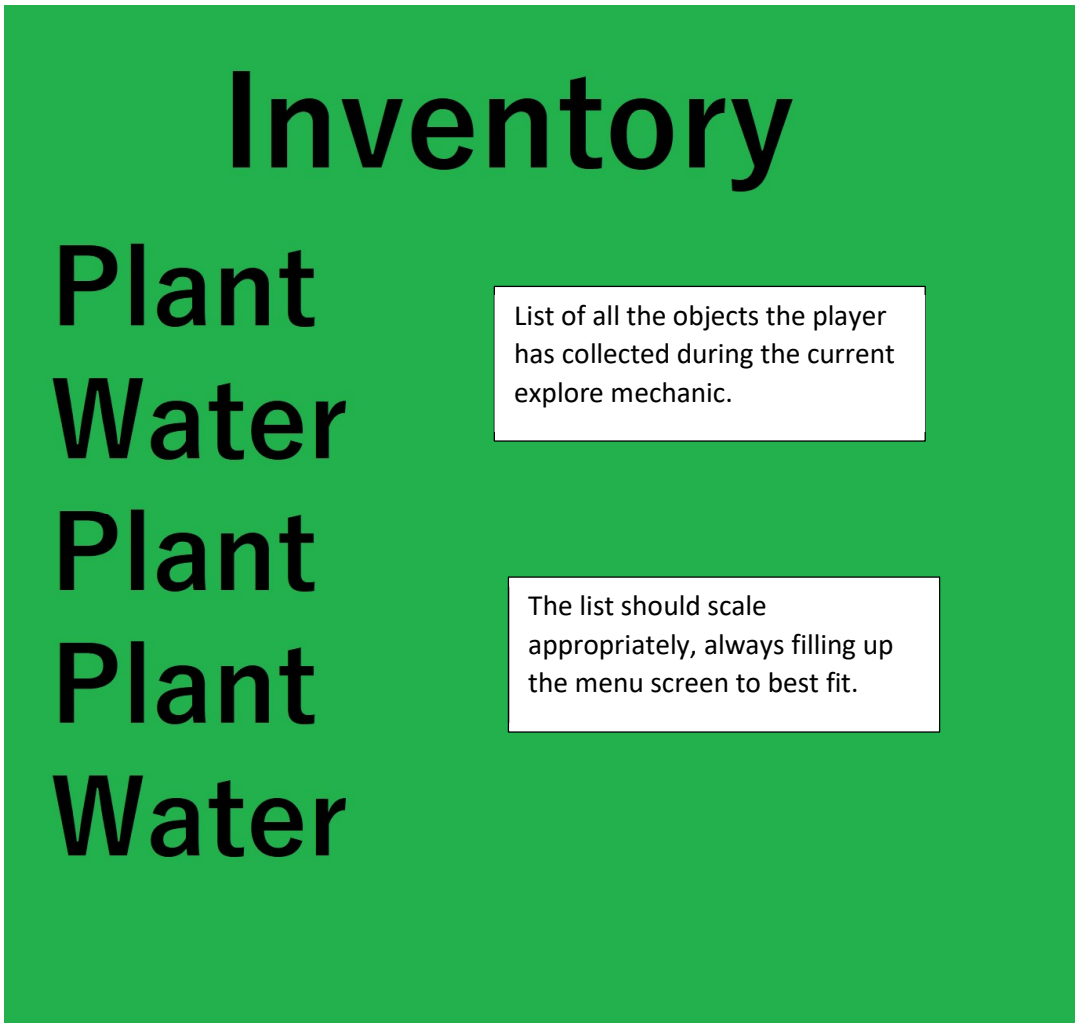
When the player walks over a water object, it will be “collected”. The water object should be deleted and moved to the player inventory.

Below, I have outlined an iteration that will be ran when the player is exploring. This will always check if a player has collided with a plant or water object and carry out procedures accordingly.



Inventory system [4]

Here, I needed to design a simple menu for the inventory, which would just display the current items that the user had collected. For now, there were no interfacing elements. The player should be able to access this screen whilst playing the game, perhaps by holding down a button.



Below is some pseudocode that will be ran every time a new item is picked up. This will update and populate the inventory display.

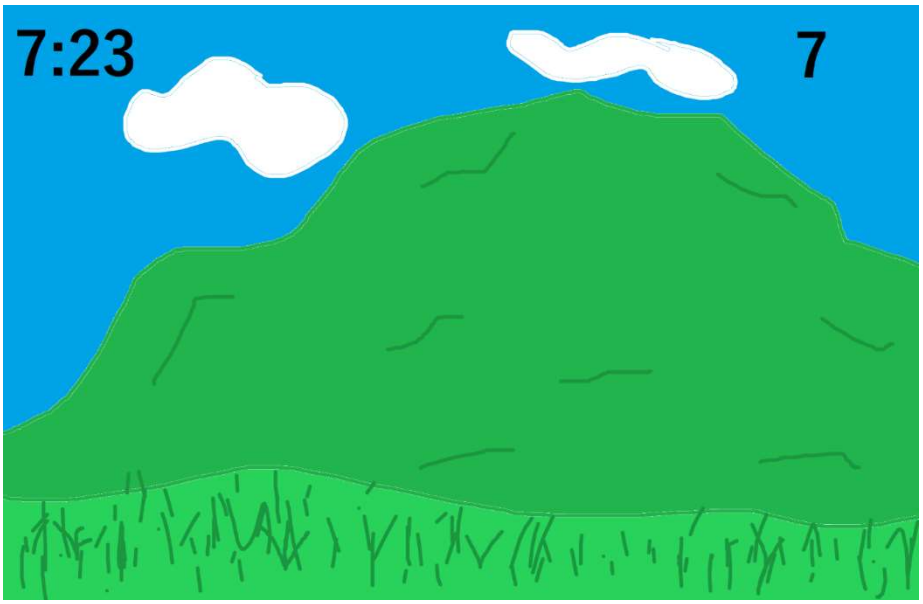
```
function populateInventory(item, inventory):
    string = inventory + "\n"
    inventory = inventory + string
    return inventory
end function
```

The input consists of two strings: the name of the collected item, and the current inventory list.

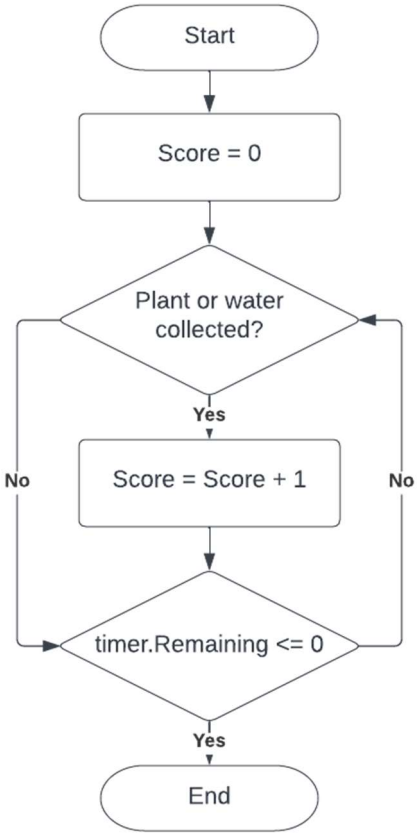
The output of this function can then be pushed to the UI element.

Score system [4]

Here, I have laid out how I would like the score to be displayed on the player's user interface, alongside the timer as discussed previously.



For now, the score system will be quite simple. Any time a player picks up a plant or water object, the score will be increased by 1. No high score element is going to be implemented at this point.



[4] Usability features

In this milestone, the implementation of an inventory screen can certainly be seen as a usability feature. The fact that the player will be able to check their inventory at any point during the explore mechanic is very helpful for them, meaning that they can focus on what they need more of: plants or water. Furthermore, the use of clear, black text on a green backdrop is helpful to the colourblind or any users with sight issues. Resizing the text to best fit is important too. For example, it does not matter whether the player currently has 2 or 23 items, they will always be able to easily read the inventory screen. Finally, the score counter in the top right of the screen lets the player know what their current score is, which updates every time they pick up a new item. This means that they do not have to keep track themselves.

In terms of player experience, there are another few usability features that I have implemented. Firstly, the ability to just walk over a plant or water object to pick it up is simple and intuitive for the user. It is then automatically moved into their inventory, which is quick and easy for them. Finally, the tree objects have no collider attached to them, since I thought this would be a major inconvenience for the player when they are exploring. This also benefits performance.

[4] Class diagram

Date: 07/12/22

Below I have made diagrams for all classes within this milestone.

Plant
This class has no variables.
objectCollide(self, object)

Water
This class has no variables.
objectCollide(self, object)

Tree
This class has no variables.
This class has no methods.

Score
count : integer
start(self) update(self, text)

Inventory
itemsList : list<string> itemsText : string
start(self) update(self, text) getInventory(self) toggleMenu(self)

[4] Variables

Date: 09/12/22

Variable	Data type	Purpose, justification and validation
Score.count	Integer	This will store the user's current score whilst exploring. It will increase every time they collect a plant or water object. This value will be reset to 0 every time the explore.
Inventory.itemsList	String list	This list will store the current objects collected by the player. It can be generated by looping through the child objects of the "Inventory" parent.
Inventory.itemsText	String	This string will store all the items collected by the user, in a text format, ready to be pushed to the text attribute in UI. It can be created by converting the list to a string.

[4] Files

There will be no use of external files in this milestone.

[4] Encoding

There will be no use of encoding in this milestone.

[4] Validation

There is very little validation that needs to be done in this section. Ensuring that plants and water are removed from the landscape once collected could be seen as validation, making sure the player can only pick up an item once.

[4] Test data

Test number	What is being tested and inputs	Expected outputs	Justification
1	Tree objects are distributed evenly across the landscape.	Tree objects can be seen across the landscape.	The number of trees may not be optimal, so I should test the value that I use to ensure it works well.
2	Plant/water objects are distributed evenly across the landscape.	Plant/water objects can be seen across the landscape.	The number of objects used may need to be changed and tweaked, so I should test the value and ensure the distribution is even.

3	An inventory screen can be seen when the “I” button is held down.	An inventory screen can be seen, displaying the player’s current objects.	I want the inventory screen to only be shown when “I” is held down, so I need to check whether the implementation works.
4	The plant/water objects are moved to the player inventory when walked over.	The plant/water object is removed from player sight and moved to their inventory.	The player needs to be able to pick up objects, since this is the core gameplay mechanic. The collisions should be tested to check that objects can be picked up, and it is reliable and consistent.
5	The score is displayed on the player user interface.	The score can be seen in the top right corner of the screen.	This feature is relatively simple, and lets the player know what score they are on. I need to check that the score is readable.
6	The score increases every time a player collects a plant/water.	The score increments by one.	The score should update, and the player should be able to see their real-time score. This needs to be tested to ensure the feature works.

[5] Milestone

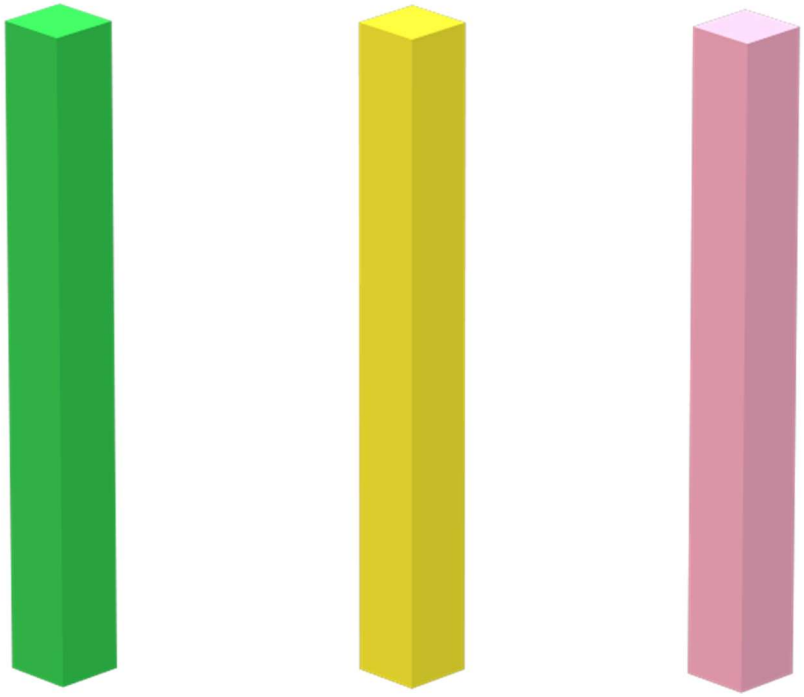
Below I have outlined all the steps that I need to take within this development milestone.

Name	Modules	Algorithms	Justification
Growing and coins **5**	Player house: ✓ Plant growth Watering system: ✓ Plant death Explore mechanic: ✓ Plant rarity Point system: ✓ Convert plants to points ✓ Points relate to rarity.	 The plants height should increase by 1 if it is watered after an explore mechanic. If a plant is not watered after the player explores, it should die. 3 different rarity plants should be implemented, each having a different effect on the score. The player should receive points, or coins, when a plant reaches a height of 4. They should also receive coins if they bring back plants but have no space in their planters. More coins should be rewarded for higher rarity plants.	This step of development should allow players to grow the plants that they collect, water them, and collect coins. Furthermore, players should be able to collect plants of differing rarities during the explore mechanic.

[5] Explore mechanic

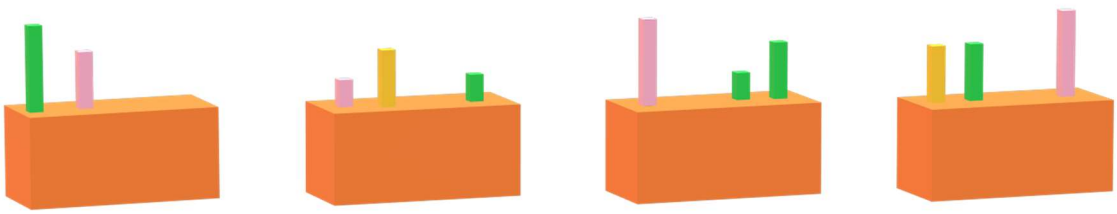
Date: 15/01/23

This section, adding different plant rarities, should be quite easy to implement. I will create 3 plant rarities of 3 different colours. I will also start using rectangular prism shapes for plants instead of small spheres, since this will be easier for the player to walk into and see. Below is an image mock-up of the 3 different plants, from lowest rarity to highest rarity.



[5] Player house

I decided to keep the plant growth system as simple as possible. Using abstraction, I was able to focus on the most important details: plants grow if watered and growing is shown by increased height. This is much easier to develop than having a lifetime counter and incrementing the minutes and seconds of each explore mechanic. I will allow for 4 slots in each planter, meaning the player can store 16 plants total. Below is an example of the planter objects at runtime.



The above data can be stored as a 16x2 2D integer array. I will represent green as 1, yellow as 2 and pink as 4. The x-axis will represent each plant, and the y-axis will store colour and height information. Below is the array table for the above example at runtime.

	Plant															
Colour	1	4	0	0	4	2	0	1	4	0	1	1	2	1	0	4
Height	3	2	0	0	1	2	0	1	3	0	1	2	2	2	0	3

I also need to design some code that will be ran every time the player enters their house. This will update the planter 2D array with any new plants that fit.

```
procedure planting(inventory)
        // calculate inventory length
        length = len(inventory)
        // iterate through all the items in inventory
        for i=0 to (len-1)
                // check if it is a plant
                if "Plant" in inventory[i]
                        // iterate through planter slots
                        for j=0 to 15
                                // if there is an empty slot
                                if global.planter[j][0] = 0:
                                        // if green plant
                                        if "green" in inventory[i]
                                                // set colour and height
                                                global.planter[j][0] = 1
                                                global.planter[j][1] = 1
                                        // if yellow plant
                                        else if "yellow" in inventory[i]
                                                // set colour and height
                                                global.planter[j][0] = 2
                                                global.planter[j][1] = 1
                                        // if pink plant
                                        else if "pink" in inventory[i]
                                                // set colour and height
                                                global.planter[j][0] = 4
                                                global.planter[j][1] = 1
                                        end if
                                // if there is not an empty slot
                                else
                                        // skip to the next j
                                        pass
                                end if
                        next j
                // if it is not a plant
                else
                        // skip to the next i
                        pass
                end if
        next i
end procedure
```

The input here will be a list of the player's inventory.

This code will add any plants collected during the explore mechanic to empty slots in the "inventory" array. This script should be called every time the player enters the house.

With the functionality of the array, I now need to run a further script every time the player enters their house. I need to display the planter list with in game objects.

```
procedure movePlants()
        // iterate through planter array
        for i=0 to 15
                // store the array values as seperate variables
                colour = planter[i][0]
                height = planter[i][1]
                // clone objects according to
                switch (colour)
                        // green plant
                        case 1:
                                // clone a green plant
                                plant = object.clone("greenPlant")
                        // yellow plant
                        case 2:
                                // clone a yellow plant
                                plant = object.clone("yellowPlant")
                        // pink plant
                        case 4:
                                // clone a pink plant
                                plant = object.clone("pinkPlant")
                end switch
                // move the clones to their locations (x, y, z)
                x = i + (i DIV 4)
                y = 4 + height
                z = 10
                plant.location = (x, y, z)
        next i
end procedure
```

[5] Watering system

Here I have continued with the theme of simplicity. Any plant that is not watered after an explore mechanic will instantly die. This is much easier to develop for me, and also adds a sense of urgency to the game. Below is pseudocode outlining the processes that will occur every time the player returns to their house after exploring.

The input here will be a list of the player's inventory.

Here I find the integer value of the amount of water the player collected whilst exploring.

```
procedure watering(inventory):
    // calculate amount of water in inventory
    amount = 0
    for item in inventory
        if item == "Water"
            amount = amount + 1
        end if
    // iterate through all the plants currently stored
    for i=0 to 15
        // check if there is water left and there is a plant
        if amount > 0 AND global.planter[i][0] != 0:
            // increase the current plant height by 1
            global.planter[i][1] = global.planter[i][1] + 1
            // decrement the amount value
            amount = amount - 1
        // if there is no water remaining and a plant
        else if amount <= 0 AND global.planter[i][0] != 0
            // kill the current plant, set colour and height to 0
            global.planter[i][0] = 0
            global.planter[i][1] = 0
        // if none of the above options
        else
            // skip to the next i
            pass
        end if
    next i
end procedure
```

In short, this code will apply all the water collected during the explore mechanic to available plants, increasing their height, and "kill" any plants that have not been watered. This will be called every time the player enters their house.

[5] Point system

Date: 17/01/23

Here, I need to implement a point or coin system. Again, this will be very simple. If a player grows a plant to its maximum height, 4, then they are rewarded in coins. Furthermore, if the player brings back plants to their house but there is no space in their planters, they should also be rewarded. Below is a table which displays the coins rewarded for each scenario. I wanted to reward the player generously for growing plants, since this is the main game loop.

Plant colour	Coins if planters are full	Coins if plant fully grows
Green	1	100
Yellow	2	200
Pink	4	400

There is some pseudocode I can add for this section. Firstly, I can edit the pseudocode from the "planting" procedure to convert unplanted plants to coins.

```
// if there is not an empty slot
else
        // if there is no space left in planters
        if j >= 15
                // convert any remaining plants to coins
                if "green" in inventory[i]
                        global.coins = global.coins + 1
                // if yellow plant
                else if "yellow" in inventory[i]
                        global.coins = global.coins + 2
                // if pink plant
                else if "pink" in inventory[i]
                        global.coins = global.coins + 4
                end if
        //if there are still planter spaces left
        else:
                // skip to next j
                pass
        end if
end if
```

Next, I can create a procedure to check for fully grown plants and convert them to their respective coin reward.

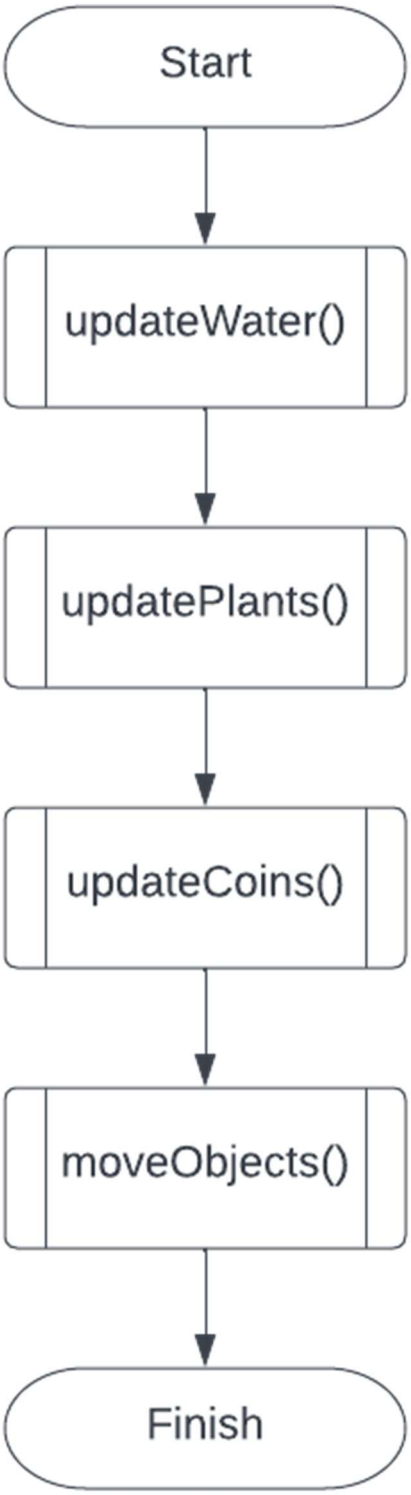
```
procedure plantToCoin()
        // iterate through planter array, check if height is max
        for i=0 to 15
                // if the plant is at max height
                if global.planter[i][1] == 4
                        // reward player with coins
                        global.coins = global.coins + (global.planter[i][0] * 100)
                        // kill the plant
                        global.planter[i][0] = 0
                        global.planter[i][1] = 0
                // if the plant isn't at max height
                else:
                        // skip to the next i
                        pass
                end if
        next i
end procedure
```

Keeping code reusability and efficiency in mind, I have realised that in most of my pseudocode for this milestone I use code that converts the integer colour value to string colour value, and vice versa. The following are two functions I could use to cut down the amount of code I need to use.

```
function string intToStr(int num)
        // use switch case
        switch (num)
                case 1:
                        name = "Green plant"
                case 2:
                        name = "Yellow plant"
                case 4:
                        name = "Pink plant"
        // return the correct string name
        return name

function int strToInt(string name)
        // use branching
        if "Green" in name
                num = 1
        else if "Yellow" in name
                num = 2
        else if "Pink" in name
                num = 4
        // return the correct integer number
        return num
```

I use “in” here, since the name parameter will not always be “Green plant” for example. Sometimes it will be the object name, which will be more like “GreenPlant.Clone”. So, the use of this keyword bypasses this issue.

To simplify this milestone, I have made a flowchart outlining what needs to be done every time the player enters their house.



[5] Usability features

Date: 18/01/23

In this milestone, there is only one implemented feature that aids the player in terms of interface. This would be displaying the number of coins they have collected on the wall of their house. Firstly, the added coin icon helps them to understand what the number means, how many coins they have collected. Putting this on the wall of their house also makes the player's coins feel more special and unique, since no other value is displayed on the wall. It does not move and is always in the same location, which makes it easier for the player.

In terms of player experience, many features have been added in this milestone. The automatic nature of the procedures being implemented means that the player does not need to worry about actually planting the plants, or watering the plants, or converting them into points. This is done for them automatically when they walk into their house. All the water they have collected is used up watering their plants. All the plants they have collected is automatically distributed into the planters. Finally, any grown plants are automatically converted into coins. This reduces the amount of work the player has to do in order to play the game and get more coins.

[5] Class diagram

Date: 20/01/23

Below I have made diagrams for all classes within this milestone.

Planters
plants : integer[,]
start(self) updateWater(self, inventory) updatePlants(self, inventory) updateCoins(self, coins) moveObjects(self)

Coins
amount : integer
start(self) update(self, text)

[5] Variables

Date: 21/01/23

Variable	Data type	Purpose, justification and validation
Planters.plants	2D integer array	This will store information, colour and height, about the plants within the player house.
Coins.amount	Integer	This will store the number of coins the player has collected. It will be pushed to a UI element for the player to see.

[5] Files

There will be no use of external files in this milestone.

[5] Encoding

Encoding is used in this milestone. So that I can use a 2D array, I have to store colour names as integer values. I encode “Green” as the number “1”, “Yellow” as the number “2”, and “Pink” as the number “4”. This is because I can multiply the numbers by 100 to calculate the number of coins rewarded when the respective plant is grown.

[5] Validation

In terms of validation, there is none to be considered here. Every part of this milestone is done automatically for the user.

[5] Test data

Date: 23/01/23

Test number	What is being tested and inputs	Expected outputs	Justification
1	The plants the player has collected are distributed into the house planters.	Plants are put into the planters, at height of 1, from left to right. There should be 4 plants per planter.	I need to get the plant spacing correct, so this needs to be tested to ensure all the plants are visible and centred.
2	The water the player has collected is used to increase plant height.	Plants are watered from left to right and will increase by a height of 1.	This is a big feature, so this should be tested, checking whether plant height increases the correct amount.
3	The plants which are not watered die.	Plants that are not watered should be removed from the house planters.	This is also a core mechanic, so it should be tested, checking the correct plants die.
4	The plants which reach a maximum height are converted to coins.	Green plants generate 100 coins, yellow 200 coins and pink 400 coins. If they are converted to coins, they are removed from the house planters.	Without this mechanic the plants will continue growing forever without rewarding the player, so I need to test this and get it right.
5	Any excess plants collected are converted to coins.	Green plants generate 1 coin, yellow 2 coins and pink 4 coins if they are excess. They should not be planted inside.	This feature may be a bit tricky, so I should test it to check whether the implementation works.
6	The player’s coin value is displayed and updated on the house wall.	The globalised integer value should be displayed on the player wall.	The player should be able to see their current coin value, so I should test this to ensure it updates and is visible.

[6] Milestones

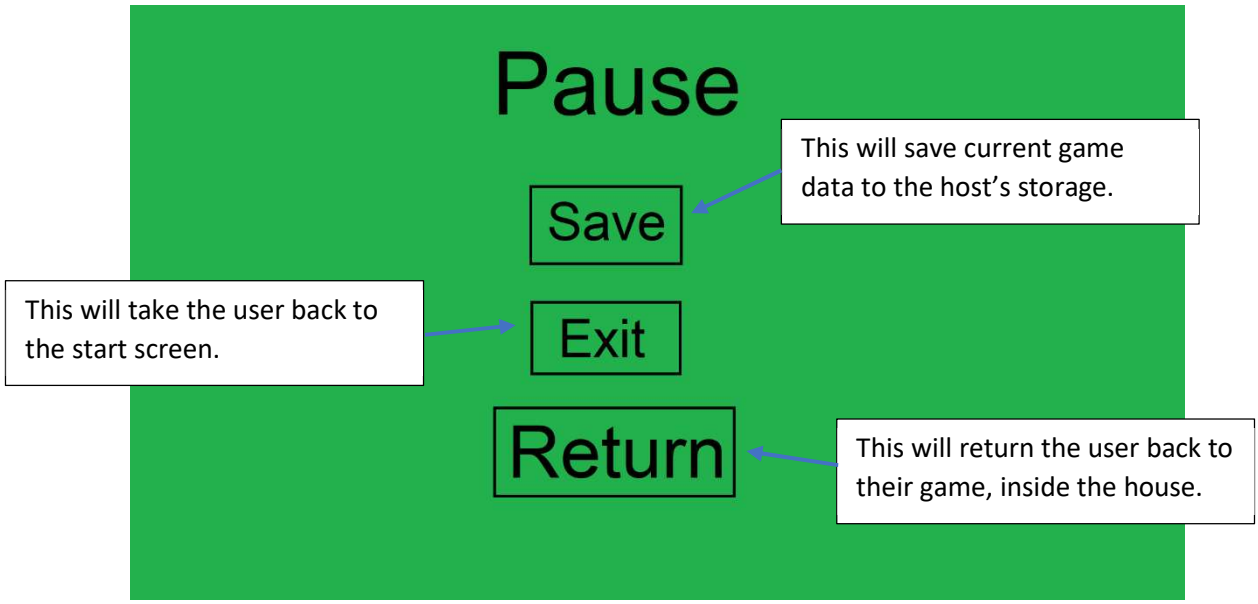
Below I have outlined all the steps that I need to take within this development milestone.

Name	Modules	Algorithms	Justification
Pause menu, save and load, music **6**	Pause menu: ✓ Save ✓ Opening screen ✓ Return Save game: ✓ Create save file Load game: ✓ Read save file Sound: ✓ Music	 This button will allow players to save the game whilst playing. This button will allow players to exit to the opening screen. This button will allow players to return back to their game. This function should create a new binary save file containing all the current data. This function should read the save file if available and load the data. Here I would like to add some simple music into the menu's and explore mechanic.	This step of development should tie together the last few features and allow players to interchange between the main menu and the game. Furthermore, they will be able to save and load their game, and also hear a simple music soundtrack.

[6] Pause menu

Date: 04/02/23

Below I have made a sketch-up of how I would like the pause menu to look like. This screen can be accessed when the user presses the escape key.



[6] Save game

Date: 04/02/23

Here, I need to determine a solution to player saves. The only two variables that need to be saved are player coins, and the planter array. To simplify this problem, I will only allow the player to save within the pause menu within the “Inside” scene. This solution means that the pause menu can only be accessed from the “Inside” scene.

In terms of implementation, I have decided to use the “PlayerPrefs” class that is built into Unity. This allows the storage of variables and their values in the player’s local registry, unencrypted. Whilst this may be a vulnerability later on, since the player may be able to edit their save file, this is the easiest way to save data by a large margin. Therefore, I will use this method to save player data.

[6] Load game

Date: 05/02/23

Due to the convenience of the “PlayerPrefs” inbuilt class, I will also use this method to load variables in the game. The load procedure should be accessible from the start screen and game over screen.

[6] Usability features

Date: 06/02/23

Within this milestone, I have implemented a few features which aid player usability in terms of interface. The implementation of a pause menu allows players to exit the game, to change settings for example, and then load back into their game. This gives the user flexibility when playing. Furthermore, the simple implementation of the save and load functions mean that the player is not confused with the interface. The user simply saves in the pause menu and loads from the start menu. This is a very intuitive implementation.

In terms of player experience, the save and load functions add a lot of usability. The player can now close the application and load their progress again at a later date since their progress is stored on their disk. They do not need to start from scratch every time now.

[6] Class diagram

Date: 07/02/23

Below I have made diagrams for all classes within this milestone.

Pause menu
This class has no attributes.
save(self) exit(self) return(self)

[6] Variables

Date: 08/02/23

There are no variables to be discussed in this milestone.

[6] Files

In this milestone, I need to save data to the player's host computer. There are multiple options here: saving data as a text document, saving data in a binary file, saving individual data items in windows registry. Each of these options have their own pros and cons, and implementing a binary file save and load will be the most rugged.

[6] Encoding

There is no point in this milestone where encoding is used.

[6] Validation

In terms of validation, the pause menu system needs to be considered. Ensuring that the menu is a closed system, and that it works well switching between scenes. Furthermore, the saving system needs tested and it should be a reliable implementation.

[6] Test data

Date: 09/02/23

Test number	What is being tested and inputs	Expected outputs	Justification
1	The menu music is played in a loop when the player is in the "Start" scene.	Menu music is played in loop.	The menu music should be played on loop, so I need to test the implementation of the music mixer.
2	The inside music is played in a loop when the player is in the "Inside" scene.	Inside music is played in loop.	The inside music should be played on loop, so I need to test the implementation of the music mixer.
3	The outside music is played in a loop when the player is in the "Outside" scene.	Outside music is played in loop.	The outside music should be played on loop, so I need to test the implementation of the music mixer.
4	The pause menu can be accessed by pressing the "escape" key.	Pause menu is loaded.	The scene must be changed upon the press of the "Escape" key, so I should test that this is the case within the implementation.
5	The player can return back to their game by clicking "return" in the pause menu.	Player is returned to game.	The player needs to be able to return back to the main game after pausing, so I should test this button.
6	The player can save their game by clicking "save" in the pause menu.	Essential variables are saved.	This feature may cause a lot of issues, so I should test that it

			works and functions correctly.
7	The player can return to the opening screen by clicking “exit” in the pause menu.	Start screen is loaded.	The player should be able to leave the game from the pause menu, so I should check that the scene changes correctly when this button is pressed.
8	The player can load their game by clicking “load” in the main menu.	Game save is loaded.	This feature also may bring up a lot of problems, so I should check that the implementation is robust and secure.
9	The player can start a new game by clicking “new” in the main menu.	New game is loaded.	I created the button for this feature much earlier in development, so now I need to test this feature works well alongside the new save/load system.

[7] Milestones

Below I have outlined all the steps that I need to take within this development milestone.

Name	Modules	Algorithms	Justification
Polishing and feedback **7**	Feedback: ✓ Timer fix ✓ Easier item pickup ✓ Score to points ✓ Game over load	An extra zero needs to be added to the timer when a digit is on its own. The size, or radius, of the water objects needs to be increased so that the collision between the player and the water objects are more consistent and reliable. Here, I need to remove any algorithms that I implemented to convert excess collected plants to coins, and simply add the score value to the coin value when the player returns to their house. The load button in the game over screen does not work, so I need to add functionality to this button, using the already existing load function.	This final step in development will polish implemented features and include any feedback that I have received during my evaluation stage.

[7] Timer fix

Date: 02/03/23

Here, I need to add some code to populate the timer seconds and minute with a 0 prefix if there is a single digit. This will be quite a simple change, consisting of a few if statements, and the pseudocode can be seen below.

```
if seconds.length() == 1 then
        seconds = "0" + seconds
if minutes.length() == 1 then
        minutes = "0" + minutes
```

[7] Easier item pickup

At this point I will need to increase the size of the water objects. This is a simple change, just requiring an increase of object radius.

[7] Score to points

This section will include the removal of all the non-functional code that converts excess collected plants into coins and replace it with a simple line of code that adds the score value to the coin value every time the player returns to their house.

```
global.coins = global.coins + global.score
```

[7] Game over load

Here, I need to add functionality to the load button in the game over scene. This is simple and consists of calling the load function when the button is clicked.

[7] Usability features

Date: 06/03/23

In this milestone, there is one fix that I have made that help player usability in terms of user interface. This is the addition of the functionality to the "Load" button in the game over screen, which before this milestone did not work since I had forgotten about it. This helps the player since it allows them to skip out a scene change and go straight back into their save from the game over scene.

In terms of player experience, there are many fixes implemented in this milestone that aid the player's usability. Firstly, the timer fix which added a zero onto any single digits helps avoid any confusion when it comes to reading the time left. Furthermore, the easier item pickup fix lets the user consistently and reliably pick up water objects, without having to try multiple times like previously. Finally, the score to points fix adds more reason to the score mechanism, and lets the user slowly but consistently collect coins, instead of relying solely on plant growth.

[7] Class diagram

There are no additional classes introduced in this milestone that need to be discussed.

[7] Variables

There are no variables to be discussed in this milestone.

[7] Files

There are no external files that have not already been discussed in this milestone.

[7] Encoding

There is no point in this milestone where encoding is used.

[7] Validation

When validating this milestone, I should ensure that the score to points system, when paired with the pause menu, functions correctly, and that the user cannot find a vulnerability.

[7] Test data

Date: 07/03/23

Test number	What is being tested and inputs	Expected outputs	Justification
1	The timer populates single digits in the minutes or seconds with a 0 prefix.	A zero can be seen before any single digits.	This was an issue before the implemented fix, so I need to test to ensure that the problem is solved.
2	Water objects can be easily and reliably collected when walked over.	The item can be picked up in a single action, and multiple attempts are not required.	This is somewhat opinion, whether the objects can be picked up reliably, so I should test it and clearly

			demonstrate that the problem has been solved.
3	The player's score value is added to the coin value every time they return.	The coin value is updated with the addition of the player's score.	This should be quite a simple fix, but I should still test this to ensure that the solution works.
4	The player's save file can be loaded from the game over screen.	Any saved data values are loaded, and the scene is changed to "Inside" when the load button is pressed.	This should also be quite easy to implement, but I should test it to check whether the procedure is correctly called.

Post development testing

Date: 20/03/23

In this section, I will outline the tests that need to be carried out after development.

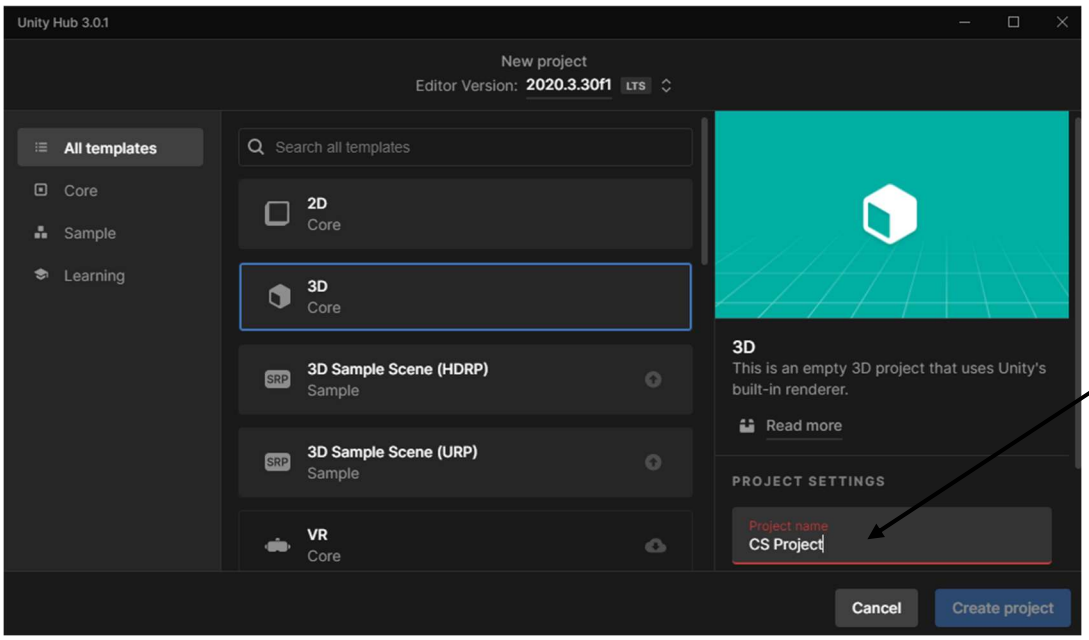
Post development test	Testing to be performed	Justification
The menu system is intuitive and easy to navigate.	The menu is a closed system and allows users to navigate back and forth between scenes.	Users need to be able to navigate the menus to edit settings and save their game, this ensures that they have an easy experience doing so.
The music is suitable and not overwhelming.	The music does not become annoying and is not too loud in scenes.	This ensures that players do not quit the game due to the music, and it makes the game more relaxing.
When exploring, landscape is randomly generated, with trees, plants and water distributed across it.	Landscape is different each time, with new plants and water objects.	This also adds to replay-ability, so that the player has a new experience every time they explore, which is more fun.
Plants and water objects can be collected when exploring.	Objects are picked up when the player walks over them.	This is a core function and is required for the user to play the game and have fun.
Plants can be grown with collected water, and coins are rewarded.	Plants increase in height when watered and are converted to coins if they reach maximum height.	The concept of players growing their plants over time adds a lot of fun and strategy to the explore mechanic.
Score is converted to coins each time the player returns.	The score value is added to the total coin value every time the player returns to their house.	This adds a basic function to the score concept, pushing users to explore further out and for longer, having more fun in doing so.
The planter information and coin value can be saved locally on the host computer.	Data can be saved by clicking the save button.	The concept of players saving their progress gives users a sense of achievement.
Game saves can be loaded from local files.	Data can be loaded by clicking the load button.	This ensures users can pick up where they left off, adding replay-ability and fun over multiple sittings.

Developing a coded solution

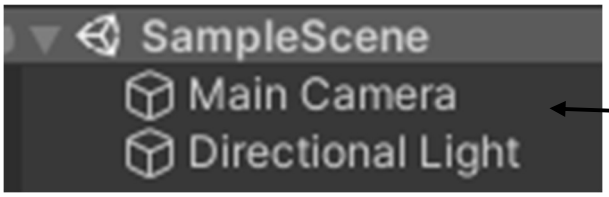
[1] Initialise project

Date: 26/07/22

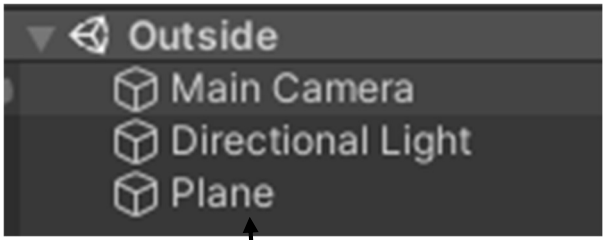
To begin with, I created a new project in Unity with the “3D Core” template. This ensures that the rendering engine loaded into the project is compatible with a 3D game. This creates a “SampleScene” with two objects, consisting of a “Main Camera” for the game and “Directional Light” to illuminate the scene. To begin with, I renamed the scene to “Outside” which might be changed later in development but is good for now. Along with this I created a temporary “Plane” object so I could begin development. Screenshots from this process can be seen below.

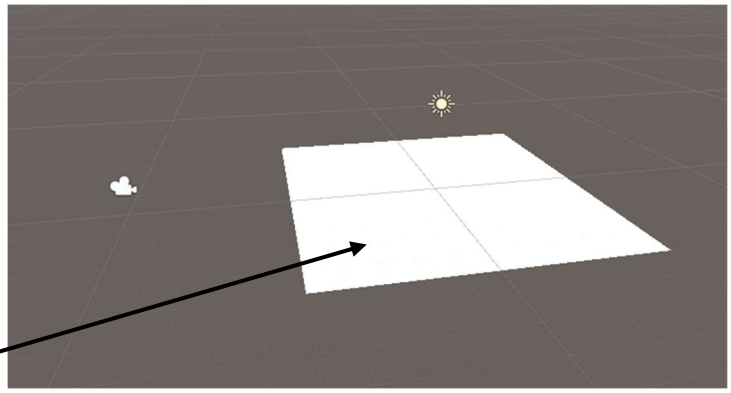


This can be changed once I decide on a name for the game.



The drop down beneath each scene shows the game objects it contains.



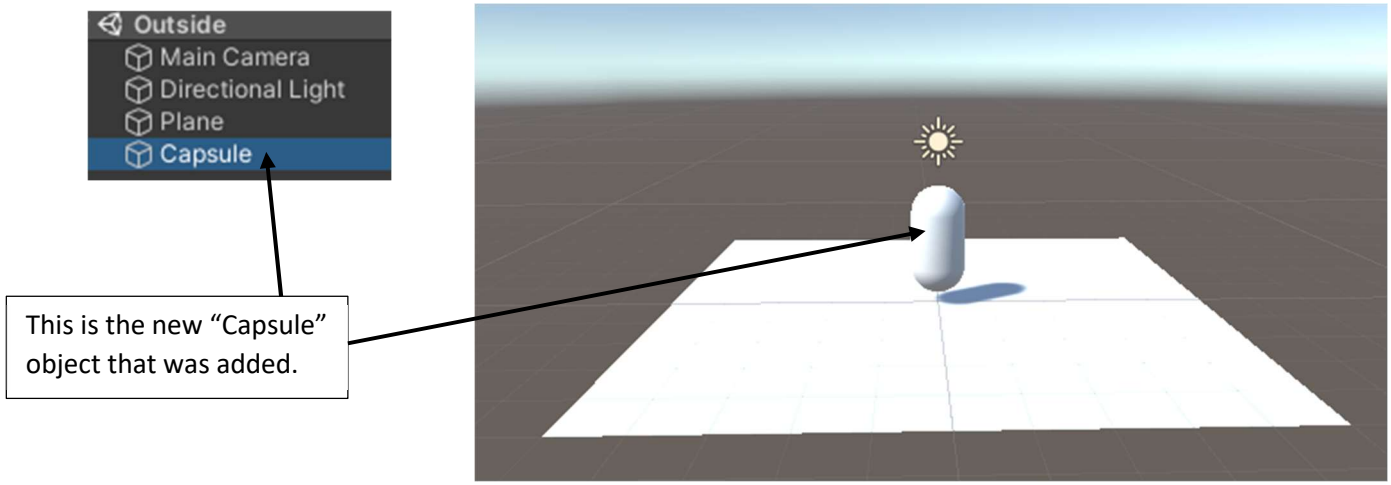


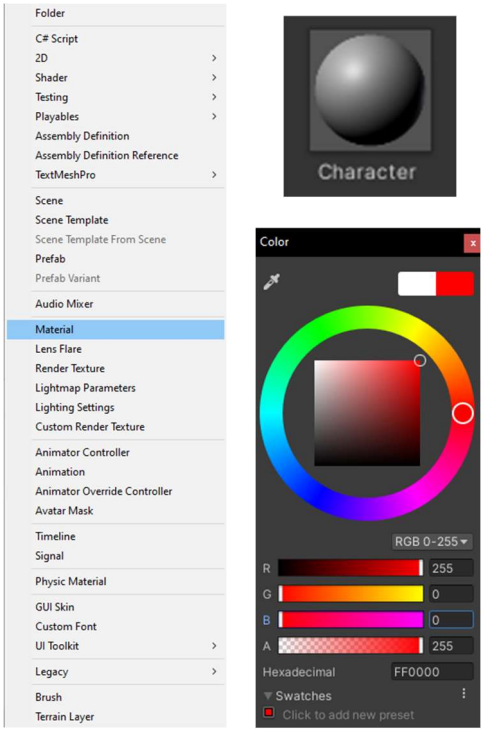
This is the new “Plane” object that was added.

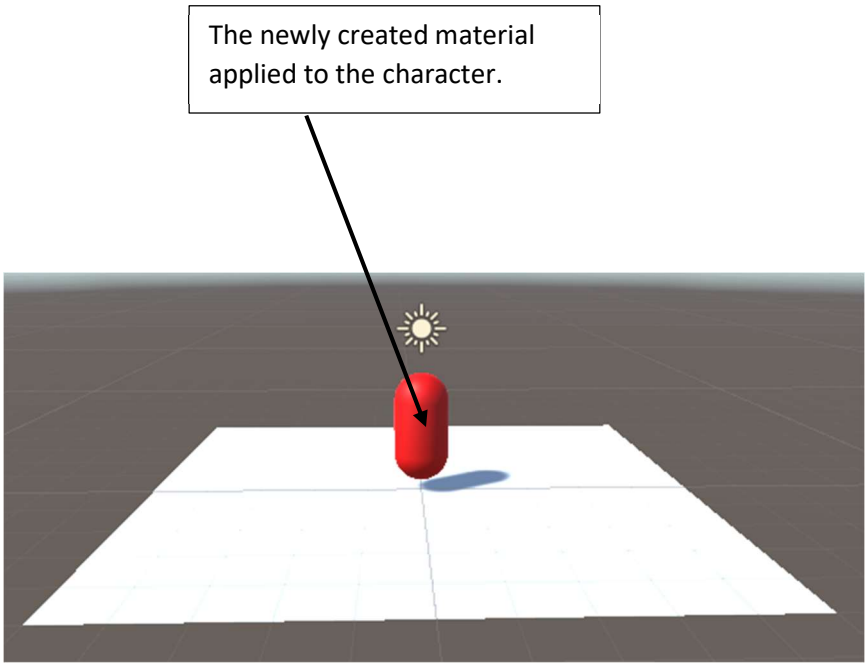
[1] Character controller

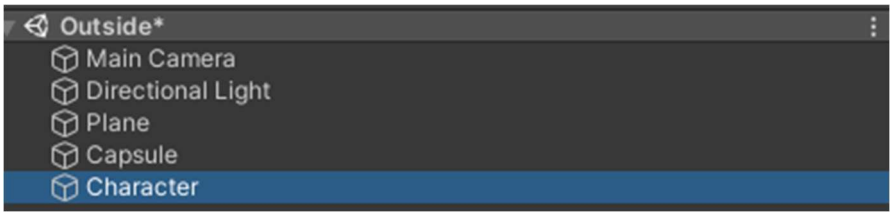
Date: 26/07/22

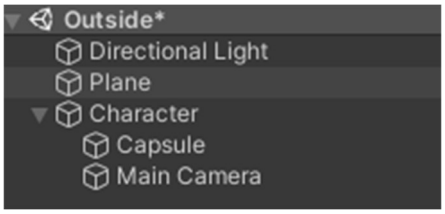
To begin this stage of development, I created a new “Capsule” game object to act as the character. The shape is temporary but is a good substitute until I change it in the future. To make it easier to see and develop with, I decided to create a new material and apply it to this object. I selected create new item, then clicked on “Material”. This made a blank material, which I renamed to “Character”. After changing the RGB values to make a red colour, I drag and dropped it onto my character. I then made a materials folder and moved the “Character” material into there. Next, I made a new “Empty” object and named it “Character”. From here I could move the “Main Camera” and “Capsule” into this object. Finally, I added a new component called “Character Controller” to the “Character” object. This allows the object to be moved around and manipulated. I need to create scripts that interact with the character controller so it can function. This process can be seen in the screenshots below.



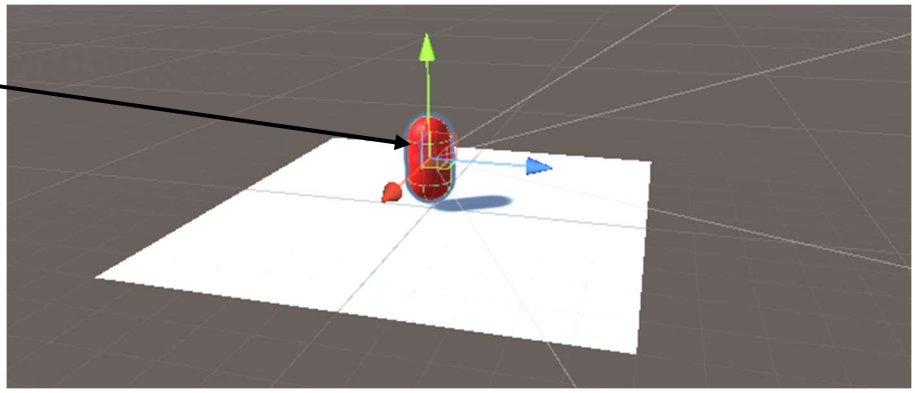




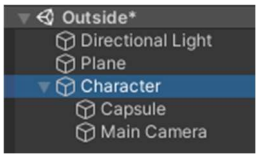


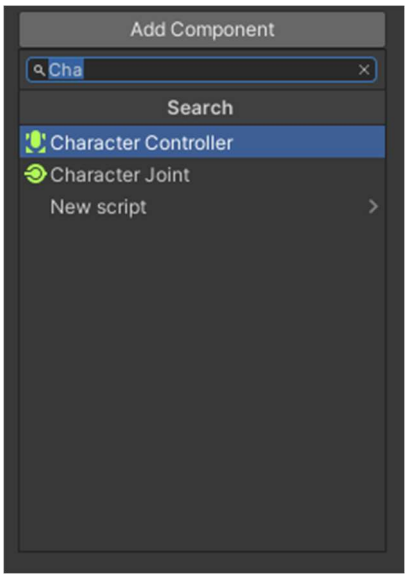


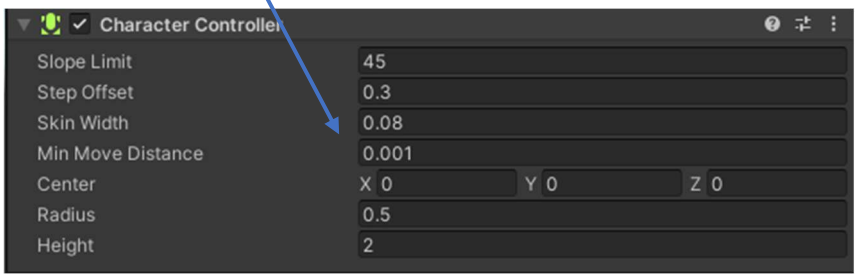
This is the “Character” object which contains the “Capsule” shape and a “Main Camera”.



I did not have to change any of these values. This might change in the future, such as the boundary box since I might change the shape of the character.







Date: 27/07/22

At this point, I begun creating code to manipulate the camera which acts as head movement. I tried to follow my pseudocode when creating this script, but some parts did not translate to C# very well. Initially, I created a new script called “MouseLook” and attached it to the “Main Camera” object. I then made a “Scripts” folder and moved it into there. Unity adds a basic structure to each new script, which is explained below. After this, I wrote some code to fetch the mouse axis and store it in variables. Next, I implemented a way for the x-axis to rotate the player body, and the y-axis to rotate the main camera. The code rotates the player body around the y-axis, and the main camera around the x-axis. I came across an error since the y-rotation of the camera was inverted, so I changed the “+=” to a “-=”. Test 1 and 2 from the design section can now be passed. This entire process can be seen in the below annotated screenshots.

Imports various useful objects such as lists, queues, bit arrays, hash tables and dictionaries.

Creates MouseLook class.

Required when making Unity scripts.

Imports a collection of all the classes related to Unity.

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

public class MouseLook : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

These are basic Unity functions. The “Start” function is ran once when the script is called. The ”Update” function is called every frame.

This variable contains a float value controlling the mouse speed, or sensitivity. Here it is set to 100, but I can change it later on if it does not work as intended.

These are pre-programmed axis within Unity which store the position of the mouse.

```
public class MouseLook : MonoBehaviour
{
    // create variable for mouse sensitivity
    public float mouseSensitivity = 100f;

    // start is called before the first frame update
    void Start()
    {

    }

    // update is called once per frame
    void Update()
    {
        // creates variables to store the mouse axis
        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;
    }
}
```

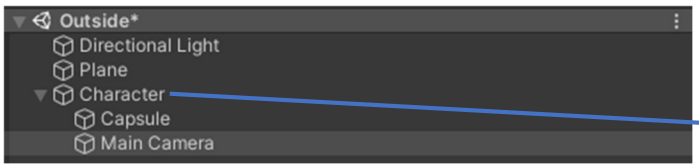
The variables must be multiplied by the set mouse sensitivity so that it can take effect. It is also multiplied by “Time.deltaTime”. This is the amount of time that has gone by since the last time the update function was called. This ensures that the player rotates independent of their current frame rate.

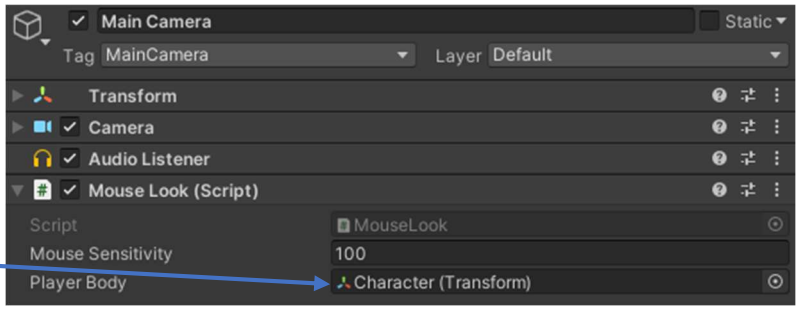
```
public class MouseLook : MonoBehaviour
{
    // create variable for mouse sensitivity
    public float mouseSensitivity = 100f;
    // create transform for the player body
    public Transform playerBody;

    // start is called before the first frame update
    void Start()
    {

    }

    // update is called once per frame
    void Update()
    {
        // creates variables to store the mouse axis
        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;

        // rotate the player body around the y-axis
        playerBody.Rotate(Vector3.up * mouseX);
    }
}
```

Here I created a new public variable to contain the "playerBody". I will need to reference the "Character" object in the Unity editor.

This will rotate the "playerBody" around the "Vector3.up", or y-axis. It is multiplied by "mouseX" which is the amount to rotate by.

For this script to function, I had to reference the "playerBody" by dragging the "Character" into the newly made "Player Body" variable.





This creates a new variable called "xRotation" which will be used to rotate the main camera around the x-axis.

This will lock the player's system cursor to the centre of their screen when playing the game. This ensures that it does not go into other windows. It only needs to be called once; hence it is put into the "Start" method.

```
public class MouseLook : MonoBehaviour
{
    // create variable for mouse sensitivity
    public float mouseSensitivity = 100f;
    // create transform for the player body
    public Transform playerBody;
    // create a variable for the x-rotation
    float xRotation = 0f;

    // start is called before the first frame update
    void Start()
    {
        // lock the cursor to the centre of the screen
        Cursor.lockState = CursorLockMode.Locked;
    }

    // update is called once per frame
    void Update()
    {
        // creates variables to store the mouse axis
        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;

        // every frame increase xRotation by mouseY
        xRotation += mouseY;
        // clamp the rotation
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);


        // rotate camera around the x-axis
        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        // rotate the player body around the y-axis
        playerBody.Rotate(Vector3.up * mouseX);
    }
}
```

Since this is within the update function, this code increases the "xRotation" variable by "mouseY" every frame.

This code will clamp the "xRotation" variable to 180 degrees. This means that the player cannot look behind them for example.

Since the "xRotation" variable is being clamped, I could not use the "Rotate" function and needed to use the "Quaternion.Euler" method. This rotates the camera around the x-axis by the "xRotation" variable.

```
// every frame increase xRotation by mouseY
xRotation += mouseY;
```

This stops the axis from being inverted when playing the game.

```
// every frame increase xRotation by mouseY
xRotation -= mouseY;
```

Date: 28/07/22

From here I began on the movement aspect of the character controller. Again, I tried to use my pseudocode for this section but found it hard to implement it in C#. In Unity, the input system is pre-programmed with the keys “W”, “A”, “S” and “D”. “W” gives the vertical axis a value of “1”, “S” gives a value of “-1”. On the other hand, “D” gives the horizontal axis a value of “1” and “A” gives a value of “-1”. This also translates to controller input as well, although I am not currently planning on taking advantage of this. The first step was to create a new script called “PlayerMovement”, attaching it to the “Character” object. Next, I wrote code to receive keyboard input from the player. From here, I had to convert this input into a direction that the character should move. After this, I could reference the character controller so that the character could move. I then implemented a speed value since I thought it would be helpful for testing different character speeds. Test 3 and 4 from my design section will now be passed. At this point the movement functions well, and the process is documented below with annotated screenshots.

This has a value of “+1” for “D” and “-1” for “A”.

This has a value of “+1” for “W” and “-1” for “”.

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

public class PlayerMovement : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        // assign keyboard input to variables
        float x = Input.GetAxis("Horizontal"); // A and D
        float z = Input.GetAxis("Vertical"); // W and S
    }
}
```

This takes the direction that the player is facing. This is then multiplied by the user input, such as “+1” or “-1” giving the correct movement for “W” and “S” inputs.

```
// Update is called once per frame
void Update()
{
    // assign keyboard input to variables
    float x = Input.GetAxis("Horizontal"); // A and D
    float z = Input.GetAxis("Vertical"); // W and S

    // direction that the character will move
    Vector3 move = transform.right * x + transform.forward * z;
}
```

This takes the direction that the player is facing and turns it to the right. This is then multiplied by the user input, such as “+1” or “-1” giving the correct movement for “D” and “A” inputs.

These two directional values can be combined with addition to find the final direction that the player wants to move.

```
public class PlayerMovement : MonoBehaviour
{
    // reference the character controller
    public CharacterController controller;

    // Update is called once per frame
    void Update()
    {
        // assign keyboard input to variables
        float x = Input.GetAxis("Horizontal"); // A and D
        float z = Input.GetAxis("Vertical"); // W and S

        // direction that the character will move
        Vector3 move = transform.right * x + transform.forward * z;

        // call the character controller function, takes a Vector3
        controller.Move(move);
    }
}
```

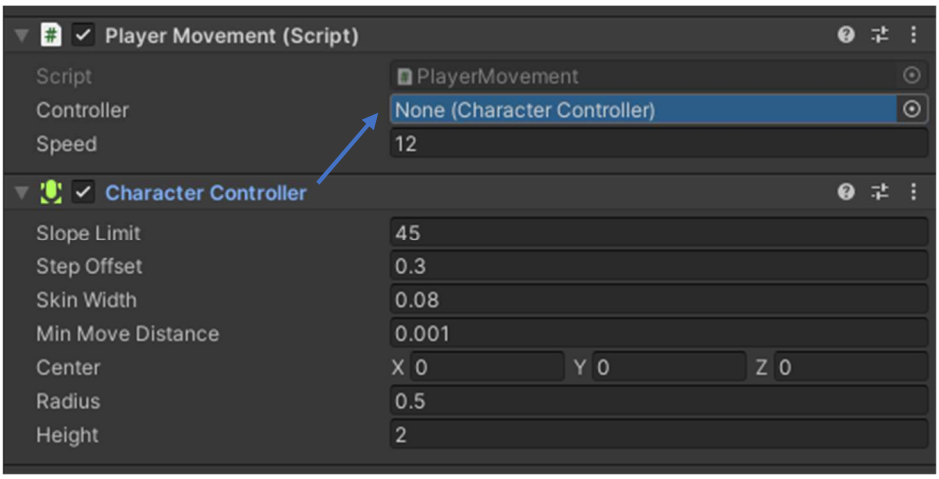
References the character controller so that it can be accessed from the script.

This calls the character controller's prebuilt "Move" function, which takes a "Vector3" input. Here I input the "move" variable created earlier.

```
public class PlayerMovement : MonoBehaviour
{
    // reference the character controller
    public CharacterController controller;

    public float speed = 12f;

    // Update is called once per frame
    void Update()
    {
        // assign keyboard input to variables
        float x = Input.GetAxis("Horizontal"); // A and D
        float z = Input.GetAxis("Vertical"); // W and S

        // direction that the character will move
        Vector3 move = transform.right * x + transform.forward * z;

        // call the character controller function, takes a Vector3
        controller.Move(move * speed * Time.deltaTime);
    }
}
```

This variable stores the set value for character speed.

Here the "move" vector is multiplied by the speed so that it can take effect. It is also multiplied by "Time.deltaTime" so that movement is framerate independent.

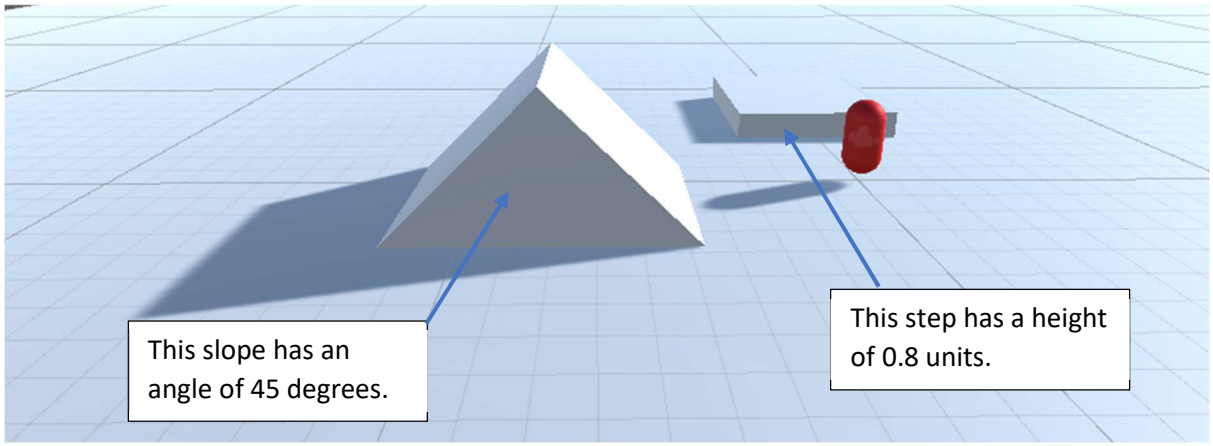


The character controller has to be drag and dropped into the input box here. This is so that the script can communicate with the character controller.

Date: 30/07/22

At this point, the character movement was good, but a few features were missing. This includes going up steps, slopes and falling. I was able to follow my gravity and jumping flowchart here. Luckily, the character controller has an in-built step function, known as “Step Offset”, along with “Slope Limit”. For now, I changed the “Step Offset” to “0.8” so that the character can get up most steps, but I can fine tune this later in development. I left the “Slope Limit” value at “45” since I don’t think the character should be able to go up slopes any steeper than 45 degrees. To test these values, I created some two cubes and put them in the scene, and upscaled the plane so that it covered a larger area. I named the cubes “Slope” and “Step”. I also moved all the terrain objects into a new object called “Terrain” for manageability. The character was able to pass both of the tests, but since there is no gravity implemented it remained at the same level. Next, I needed to implement velocity, so that I can add a gravity value to increase the velocity when falling. After this, the feature worked but the velocity would not reset upon the character touching the floor. I created an empty object within the character object to detect this called “GroundCheck” and moved it to the bottom of the character. This object will project a sphere object to see what it is connected to, thus checking if the player is on the floor. Test 6 and 7 from the design section will now be passed. After referencing the correct objects, the feature worked well and is documented below with screenshots.

This sets the maximum step height to 0.8 units.



This slope has an angle of 45 degrees.

This step has a height of 0.8 units.

$$s = ut + \frac{1}{2}at^2$$

Since “u” will always be 0 in this context, I can remove “ut” from the equation. This produces the equation below.

In this context, “s” becomes “Δy” since that is the distance to move the character. Furthermore, “a” becomes “g” because that is the acceleration acting on the character.

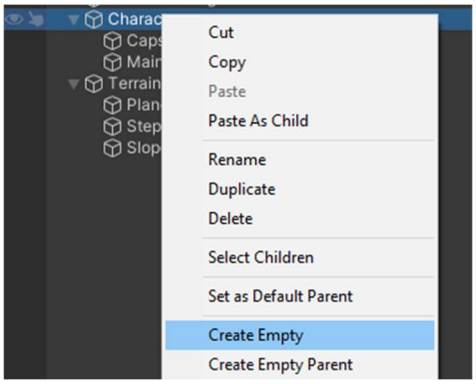
$$\Delta y = \frac{1}{2}g \cdot t^2$$

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

public class PlayerMovement : MonoBehaviour
{
    // reference the character controller
    public CharacterController controller;

    // speed of character movement
    public float speed = 12f;
    // value for gravity
    public float gravity = -9.81f;

    // vector to store velocity
    Vector3 velocity;

    // Update is called once per frame
    void Update()
    {
        // assign keyboard input to variables
        float x = Input.GetAxis("Horizontal"); // A and D
        float z = Input.GetAxis("Vertical"); // W and S

        // direction that the character will move
        Vector3 move = transform.right * x + transform.forward * z;

        // call the character controller function, takes a Vector3
        controller.Move(move * speed * Time.deltaTime);

        // increase the velocity by gravity variable
        velocity.y += gravity * Time.deltaTime;

        // reference character controller and make player fall
        controller.Move(velocity * Time.deltaTime);
    }
}
```

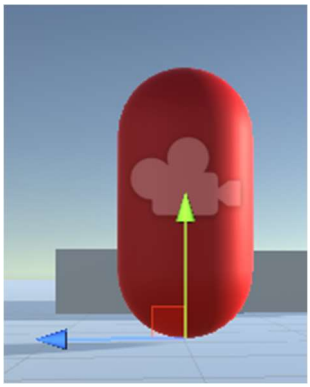
Here I create a new float to store the value of g, which is required for the freefall calculation.

I also made a new vector variable called “velocity” to store the vertical velocity of the player.

Here the equation above is being used over a couple lines of code. “gravity” is multiplied with “Time.deltaTime” and stored in the velocity vector. This is then multiplied again with “Time.deltaTime” and input into the “controller.Move” function. This uses the “$\frac{1}{2}gt^2$” equation.



This is the process of making a new empty object within the “Character” object. This was then moved to the bottom of the character.



```
public class PlayerMovement : MonoBehaviour
{
    // reference the character controller
    public CharacterController controller;

    // speed of character movement
    public float speed = 12f;
    // value for gravity
    public float gravity = -9.81f;

    // reference the GroundCheck object
    public Transform groundCheck;
    // radius of sphere to project
    public float groundDistance = 0.4f;
    // create a mask to tell sphere what objects to check for
    public LayerMask groundMask;

    // vector to store velocity
    Vector3 velocity;

    // boolean to store whether player is grounded or not
    bool isGrounded;

    // Update is called once per frame
    void Update()
    {
        // creates sphere of player position, radius groundDistance and mask groundMask
        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

        if (isGrounded && velocity.y < 0)
        {
            velocity.y = 0f;
        }

        // assign keyboard input to variables
        float x = Input.GetAxis("Horizontal"); // A and D
        float z = Input.GetAxis("Vertical"); // W and S

        // direction that the character will move
        Vector3 move = transform.right * x + transform.forward * z;

        // call the character controller function, takes a Vector3
        controller.Move(move * speed * Time.deltaTime);

        // increase the velocity by gravity variable
        velocity.y += gravity * Time.deltaTime;

        // reference character controller and make player fall
        controller.Move(velocity * Time.deltaTime);
    }
}
```

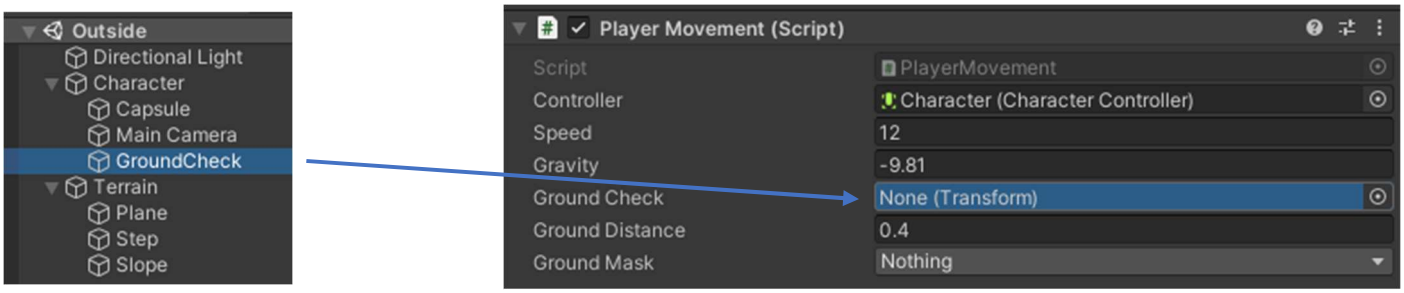
This is to reference the GroundCheck object on the character.

This float stores the radius of the sphere that will be produced.

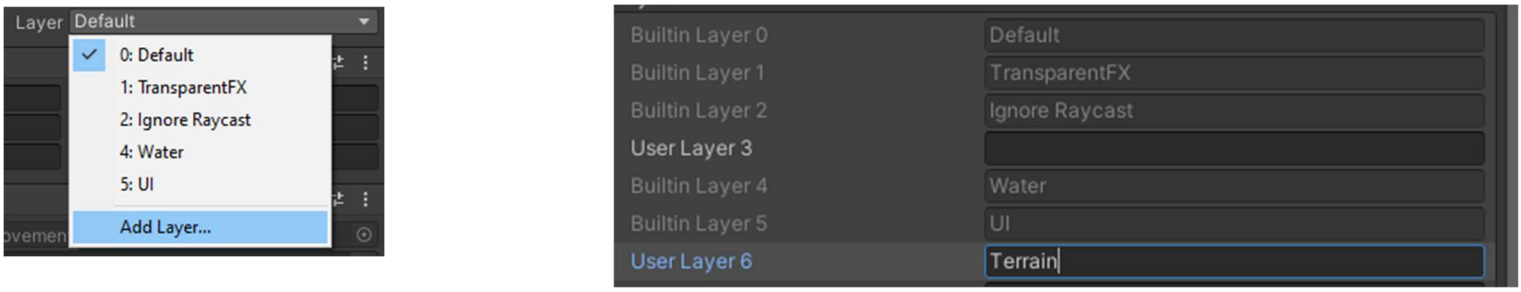
This references the layer that the sphere will check for objects within.

Here, velocity is set to 0 if the player is grounded and the velocity is above 0.

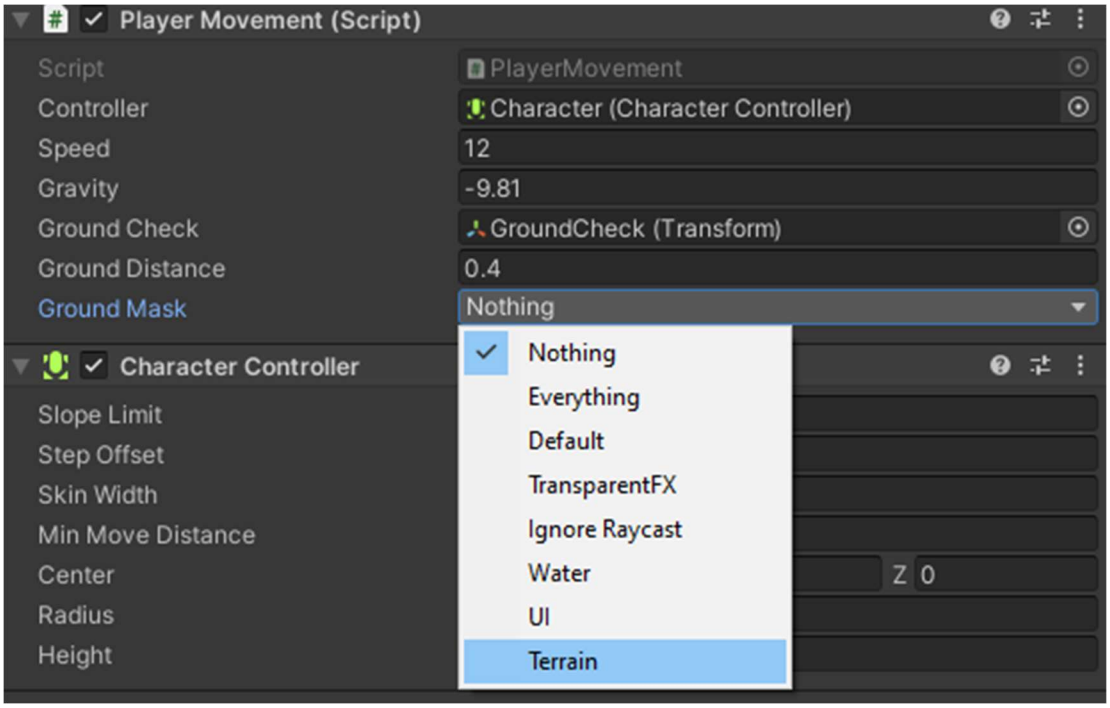
Here the “isGrounded” Boolean is calculated. A sphere is created with a position of “groundCheck.position”, which is the position of the “GroundCheck” object on the character. The sphere has a radius of “groundDistance” so 0.4 units. The mask input is “GroundMask”. Unity then returns true or false depending on whether the sphere contacts any objects within the input layer.



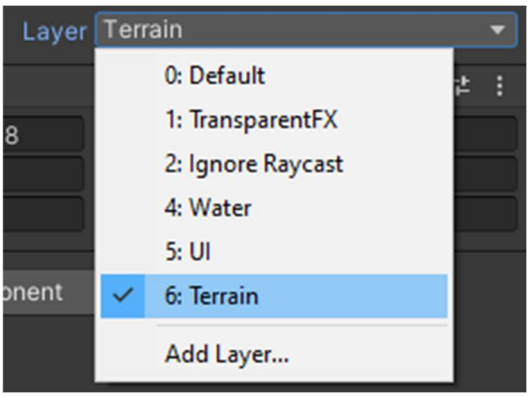
Here I referenced the GroundCheck object for the script.

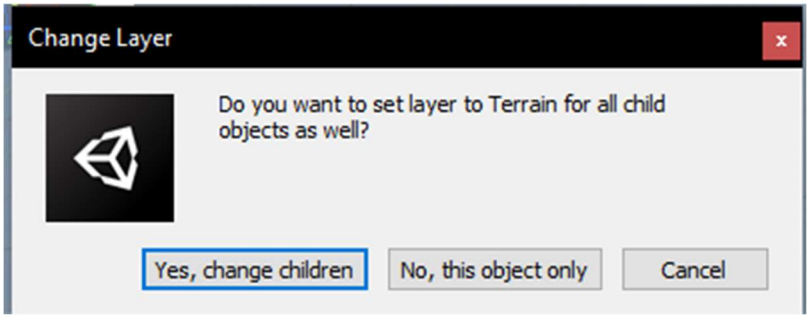


I then created a new layer called "Terrain".



I selected this layer as the "GroundMask" for the script.





Finally, I changed the "Terrain" object's layer to the "Terrain" layer, changing all the children objects too.

Date: 10/08/22

Finally, the only feature left for the character controller was to implement a jump mechanic. I began by finding an equation to work out the initial velocity required for the character to jump a certain height. After integrating this equation into my code, the mechanic functioned well. I reduced the character's jump height to "1" instead of "3" since it was too high. Test 5 from the design section can now be passed. This process is documented below with annotated screenshots.

This step added the context to the program. The displacement becomes "h" because I am trying to work out the height of the jump. This is when the final velocity will be "0". The acceleration becomes "g" due to freefall, and "g" in my script is set to "-9.81"

$$v^2 = u^2 + 2as$$

$$0^2 = u^2 + 2gh$$

$$u^2 = -2gh$$

$$u = \sqrt{-2gh}$$

This is a SUVAT equation that combines the final velocity, v, initial velocity, u, acceleration, a, and the displacement, s, of an object.

After rearranging this is the formula I found.

```
public class PlayerMovement : MonoBehaviour
{
    // reference the character controller
    public CharacterController controller;

    // speed of character movement
    public float speed = 12f;
    // value for gravity
    public float gravity = -9.81f;
    // value for jump height
    public float jumpHeight = 3f;

    // reference the GroundCheck object
    public Transform groundCheck;
    // radius of sphere to project
    public float groundDistance = 0.4f;
    // create a mask to tell sphere what objects to check for
    public LayerMask groundMask;

    // vector to store velocity
    Vector3 velocity;

    // boolean to store whether player is grounded or not
    bool isGrounded;

    // Update is called once per frame
    void Update()
    {
        // creates sphere of player position, radius groundDistance and mask groundMask
        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

        if (isGrounded && velocity.y < 0)
        {
            velocity.y = 0f;
        }

        // assign keyboard input to variables
        float x = Input.GetAxis("Horizontal"); // A and D
        float z = Input.GetAxis("Vertical"); // W and S

        // direction that the character will move
        Vector3 move = transform.right * x + transform.forward * z;

        // call the character controller function, takes a Vector3
        controller.Move(move * speed * Time.deltaTime);

        // check if the player wants to jump
        if (Input.GetButtonDown("Jump") && isGrounded)
        {
            velocity.y = Mathf.Sqrt(-2 * gravity * jumpHeight);
        }

        // increase the velocity by gravity variable
        velocity.y += gravity * Time.deltaTime;

        // reference character controller and make player fall
        controller.Move(velocity * Time.deltaTime);
    }
}
```

New public variable to store the player's jump height.

Default input that maps to the "space" key.

Apply the equation discussed above.

```
// value for jump height
public float jumpHeight = 1f;
```

Changed the value from "3f" to "1f".

Sources

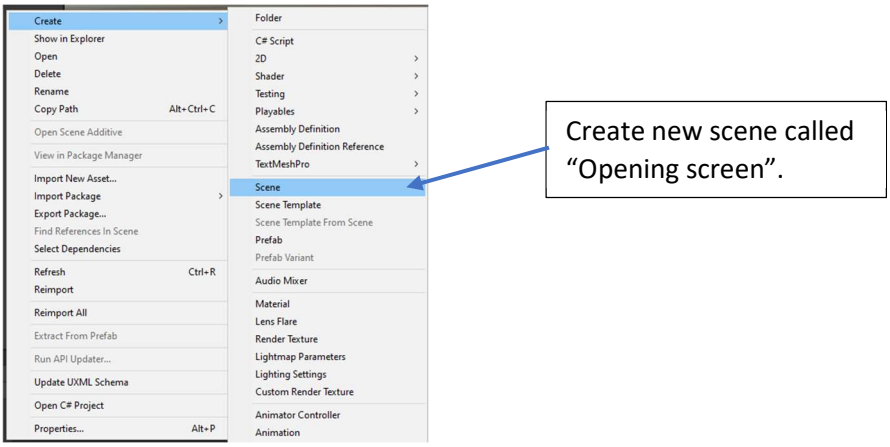
For this section of my development, I used a YouTube tutorial[16] as a guide for creating a character controller.

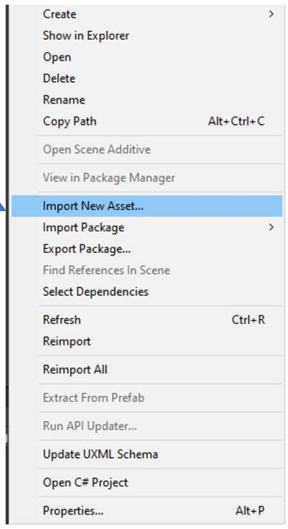
[16] (Thirslund, FIRST PERSON MOVEMENT in Unity - FPS Controller 2019)

[2] Opening screen

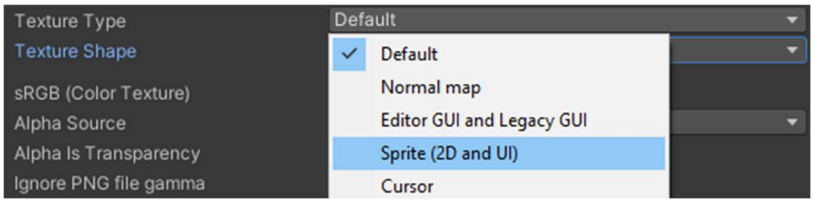
Date: 18/09/22

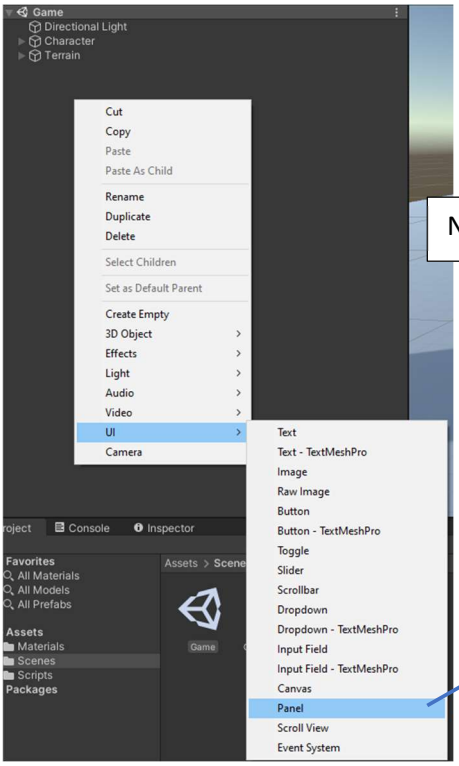
In this next section of development, I needed to create an opening screen that the user can interact with, including creating a new save, loading an old save, quit the game and access the settings menu. Initially, I changed the current scene name from “Outside” to “Game” to simplify things for myself. Then I created a new scene called “Opening screen” where I could create my opening menu. Then I created a “Panel” object, which has a “Canvas” parent object. Next I created a new folder called “Images”, then imported a temporary background image. I had to change texture type from “Default” to “Sprite (2D and UI)”. Then I set this image as the “Source Image” for the “Panel” object. I had to increase the alpha channel in the “Colour” section. Finally, I renamed the “Panel” object to “Background”. From here, I created a new “Button” object under the “Canvas” parent object. I scaled the button up slightly, then moved it up the panel. Then I changed the button colour to completely black, and unchecked “Fill center”. This gave a black outline to the button. Within the text part of the button object, I changed the text to “New” and changed the colour to black. I repeated this for all the other buttons, and recreated the mockup I created in my design section. This included creating a text object to act as the game title. I renamed each button object according to its function. Then I created the “Settings” button. This has no text and consists of a cog image. Then, I changed the buttons’ settings for when they are hovered over, and clicked. This gives the user visual feedback for their input. When I tested the scene, the scaling was completely off. I spent some time setting the scaling anchors properly so that the scene ran smoothly at different resolution. Finally, I collected all the objects under a “MainMenu” object. This simplifies it for me when it comes to adding functionality to the buttons using scripts. However, this did include adding more anchors in for the new object. Now test 1 from the design section can be passed. This whole procedure can be seen below in the form of annotated screenshots.



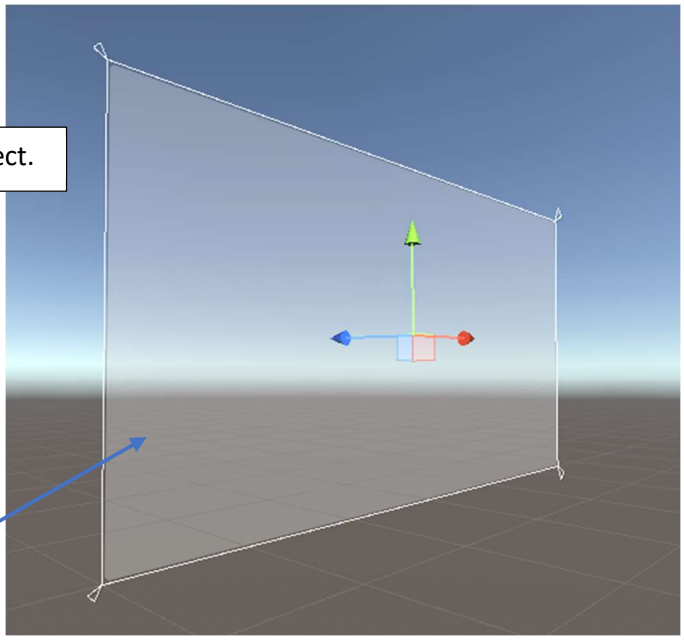


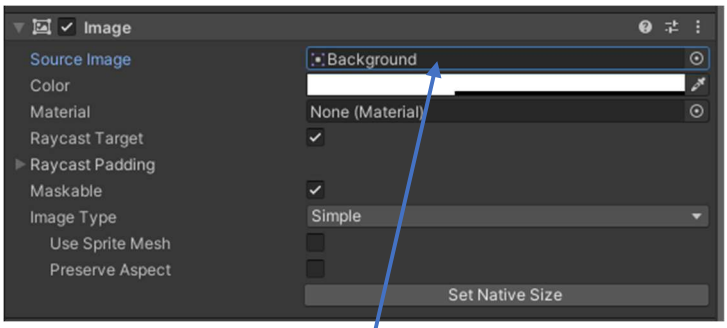
Imported ”Background.png” asset.



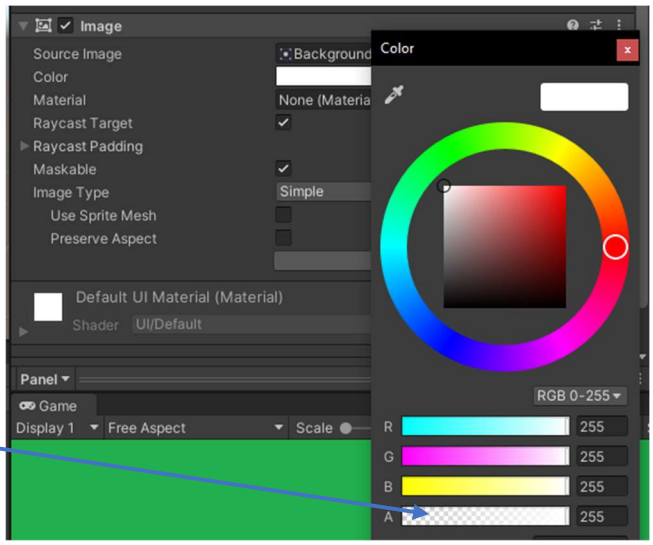


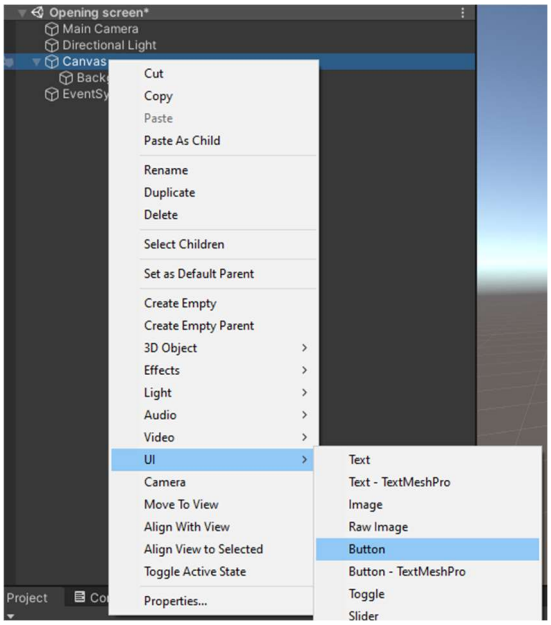
New panel object.

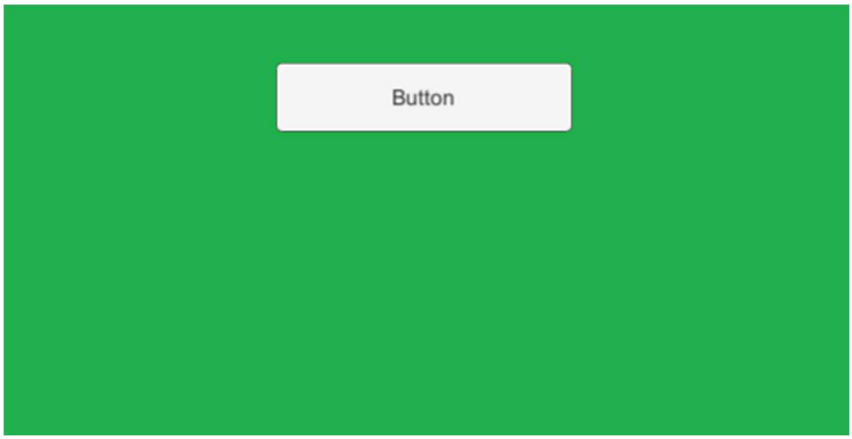




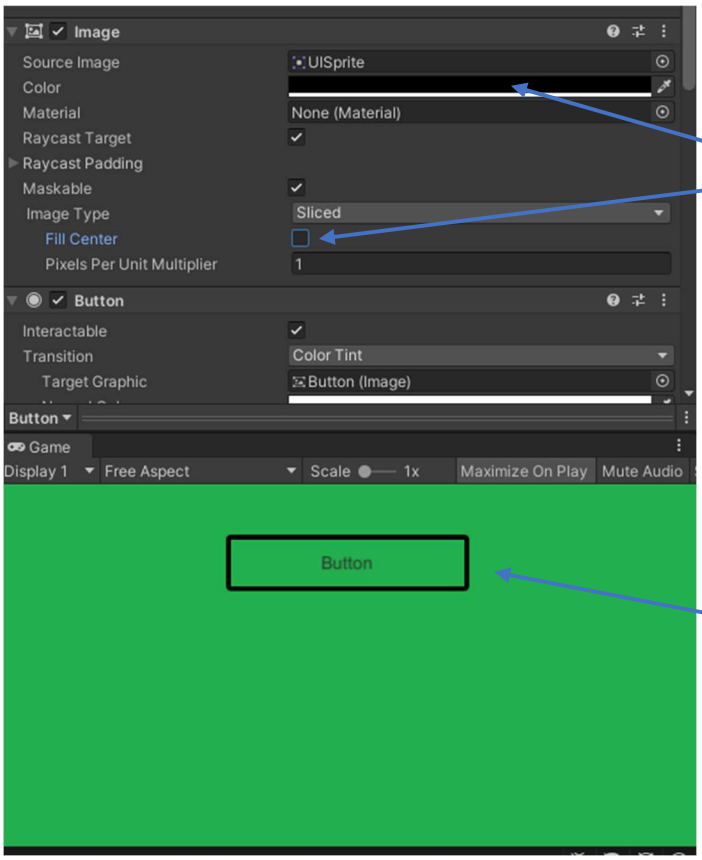
Set the “Background.png” asset to the “Source Image”. Then I increased the alpha channel to 255.





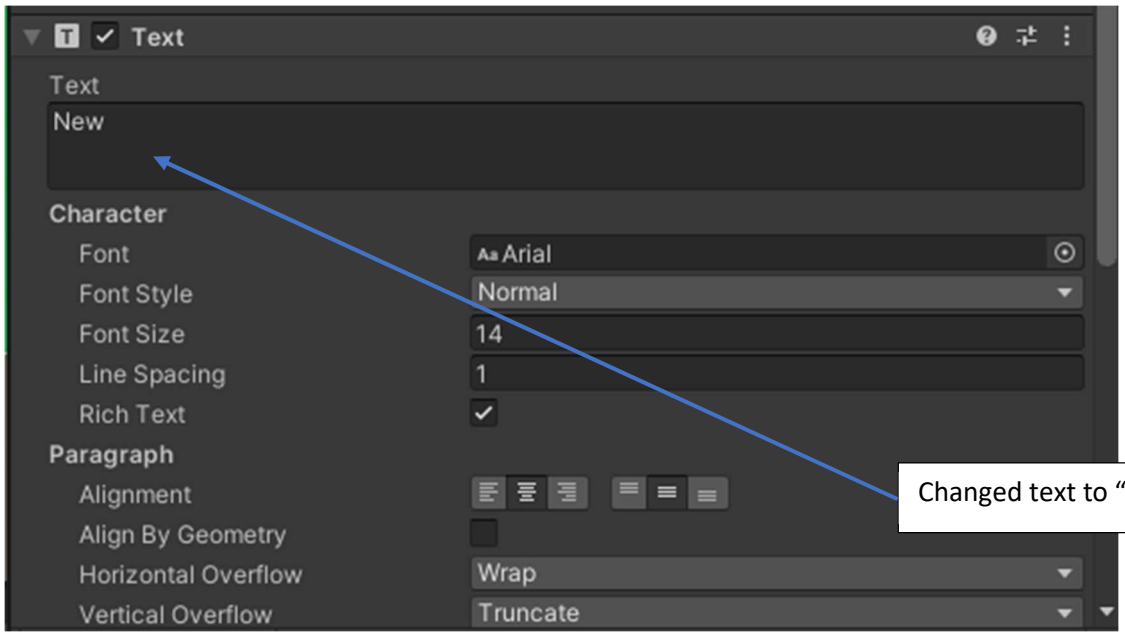


Here is the process of creating the “Button” object. On the right is a 2D view of the canvas.

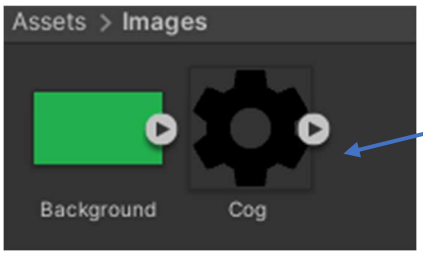


The “Color” was changed to black, and I unchecked the “Fill Center” box.

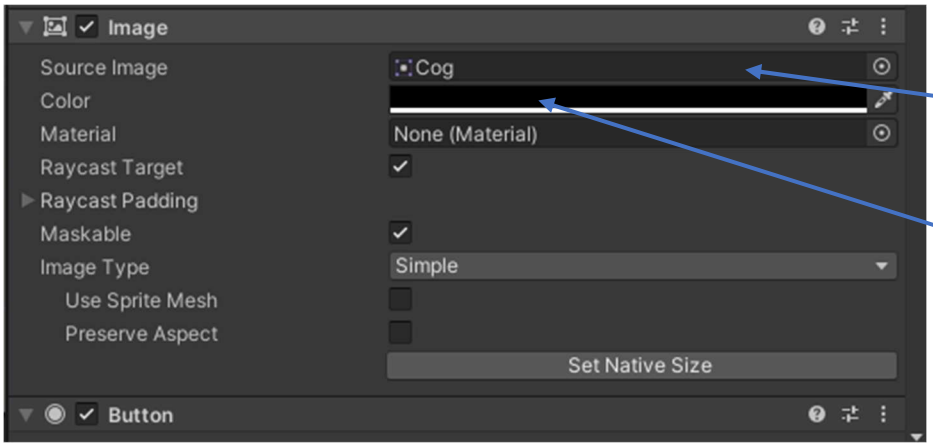
Here you can see what this looks like to the user.



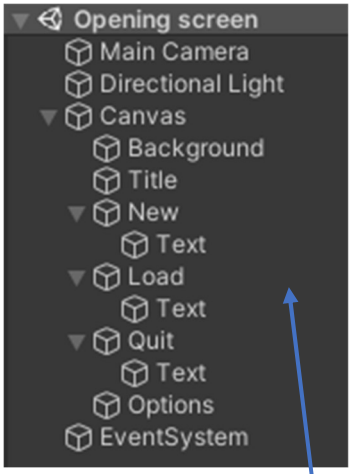
Changed text to “New”



Here I imported the "Cog.png" image. I had to change the "Texture Type" from "Default" to "Sprite (2D and UI" again so it would function in the menu. I used creative commons google search to find this image.

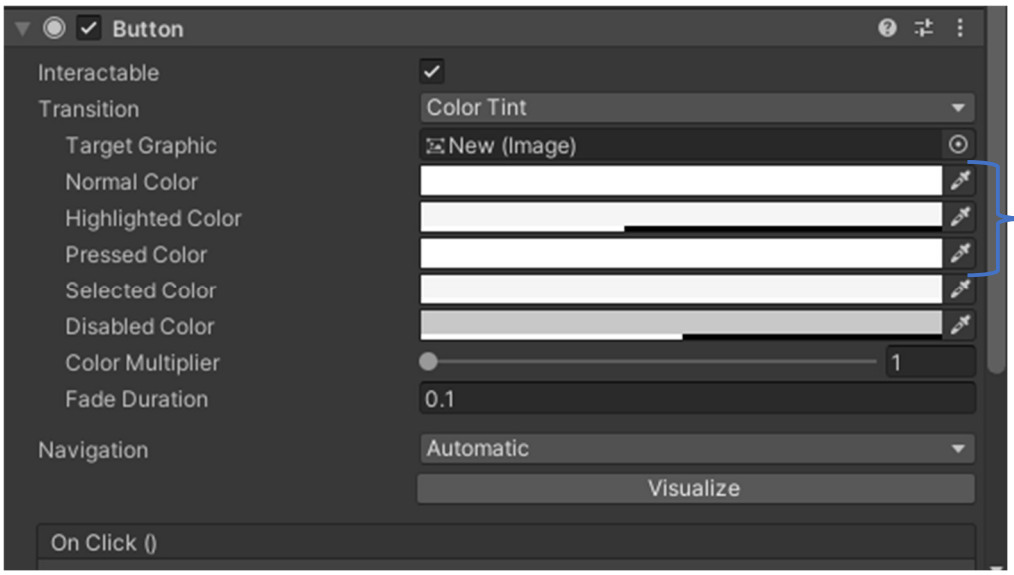


At this point, I selected "Cog.png" as the "Source Image" for the options button object. It has no text attribute since I am using an image. I then set the colour to black and set the alpha channel to max. Once changing the position and size of the button, it looked good on the menu.

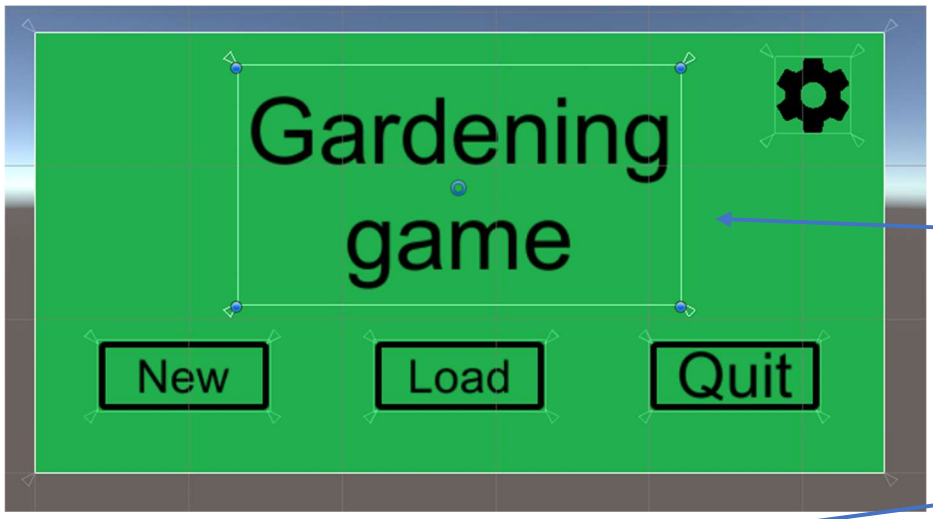


Current object hierarchy for this scene.

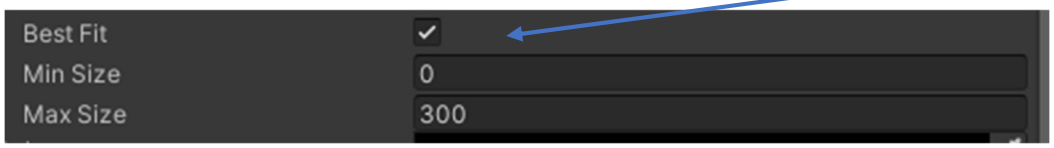
Current opening screen.

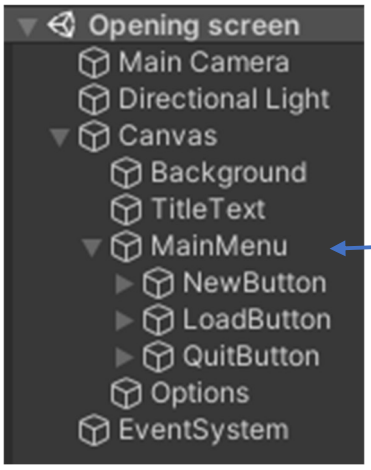


Here I changed the button colour's alpha values for different states. I made the normal alpha value 255. When a user hovers over a button, it drops to 100. Then when pressed the alpha returns to 255.



To fix the issues I was having with scaling, I set all the objects up with anchors to the canvas. Then I ensured that each text object had "Best Fit" checked and set the "Max Size" to the largest value.

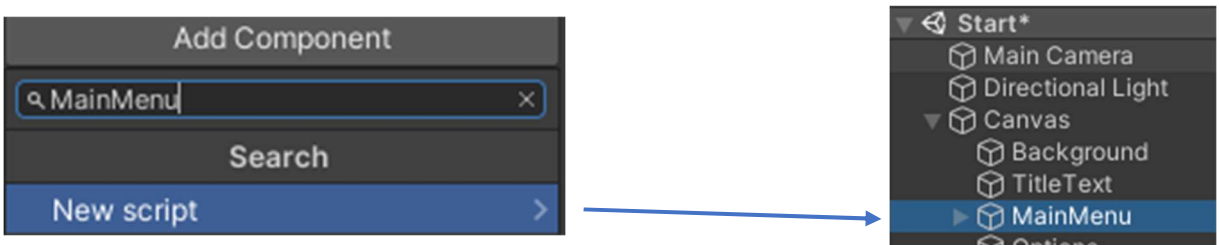




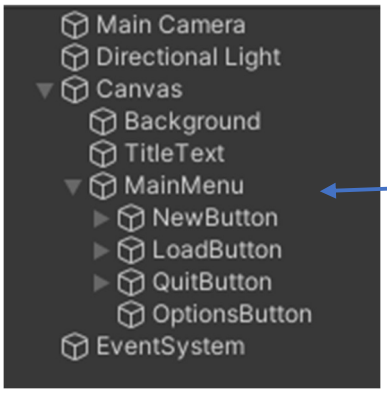
Here is a final look at the object hierarchy before I add functionality to this menu.

Date: 19/09/22

The next step of creating the opening screen was to add functionality to each of the buttons. For now, I am not going to implement any save/load mechanics but will instead make the “New” button load the “Game” scene and the “Quit” button quit the application. Other features such as the options menu will be implemented later on in development. The first thing I did was change the “Opening screen” scene name to “Start” because I thought it was more appropriate and understandable. Then I created a new script called “MainMenu” and set it as a component of the “MainMenu” object. From here I could create new functions for all the buttons in the “MainMenu” object. I also realised I had not put the “Options” button in the “MainMenu” object, so I moved it in. The first function I created was for the “New” button. To get the function to work, I had to edit the build settings and create a scene order. Next, I had to connect the “NewButton” object to the “NewGame” function. Once done, this worked great. The scene quickly changed to the “Game” scene when I clicked the “New” button. After changing scenes, the lighting was not properly rendered in the “Game” scene. After researching, I found out this only occurs in the Unity Editor to be more efficient and will not occur in the game build. Then I created the “QuitGame” function, which exits the application when the “Quit” button is pressed. This does not occur in the editor, only the game build. With the functionality of the buttons complete, this section of development is finished. With this done, test 2 and 3 from the design section will be passed. The process is documented below in annotated screenshots.



Created new script as a component of “MainMenu” object.



Updated object hierarchy.

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

public class MainMenu : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

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

public class MainMenu : MonoBehaviour
{

}
```

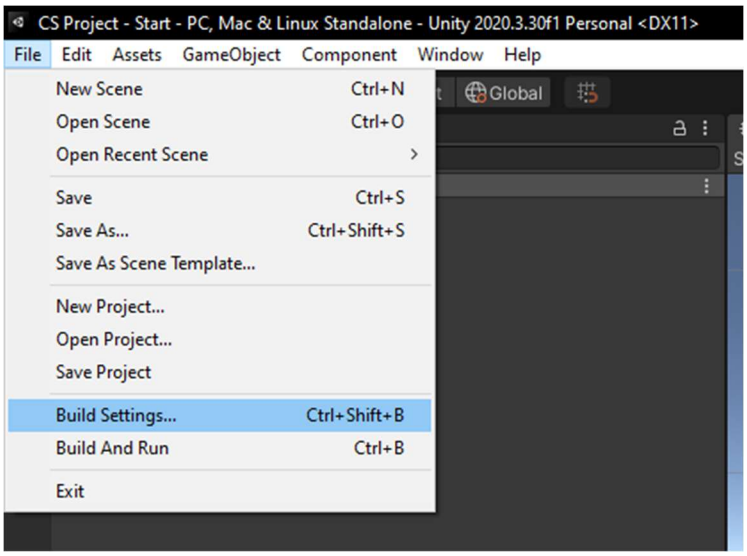
I removed the default methods within the script since they are not required.

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

public class MainMenu : MonoBehaviour
{
    // function to run when "New" button is pressed
    public void NewGame()
    {
        // get index of currently loaded level
        int index = SceneManager.GetActiveScene().buildIndex;
        // load the index plus 1
        SceneManager.LoadScene(index + 1);
    }
}
```

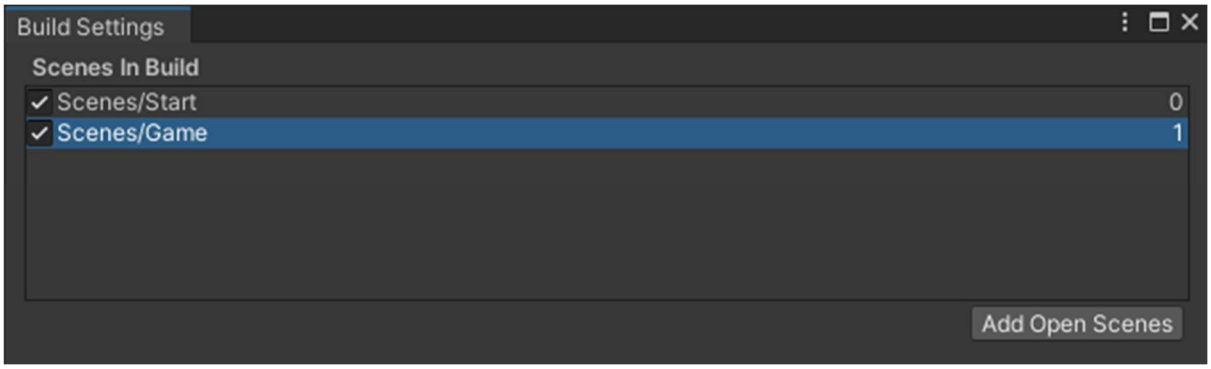
This has to be used so that the script can change the game scenes.

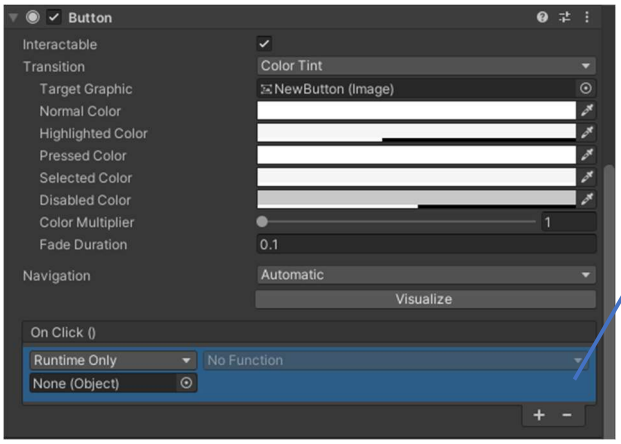
The function has to be public so that it can be called from the button.

Here I can put the scene name, its relative "buildIndex" or go to the next scene in the queue. To make my life easier in the future, I will load the next scene in the queue.



Here I edited the build settings and made an order for the scenes. Each one is assigned an index, which can be seen to the right of the name.



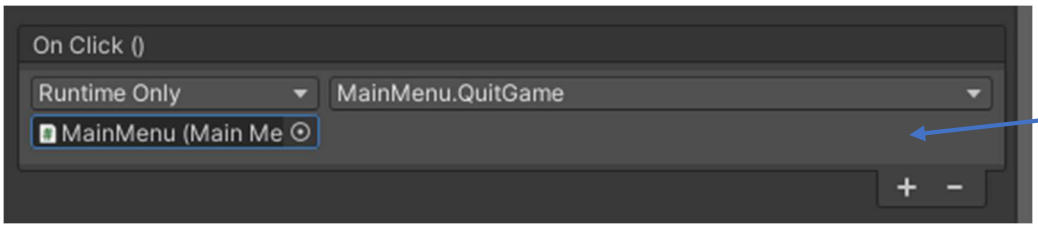


I added a new “On Click ()” option. I selected the “MainMenu” object, and then the “NewGame()” function within the “MainMenu” script.

```
public class MainMenu : MonoBehaviour
{
    // function to run when "New" button is pressed
    public void NewGame()
    {
        // get index of currently loaded level
        int index = SceneManager.GetActiveScene().buildIndex;
        // load the index plus 1
        SceneManager.LoadScene(index + 1);
    }

    // function to run when "Quit" button is pressed
    public void QuitGame()
    {
        // quit the application
        Application.Quit();
    }
}
```

New public function called "QuitGame"

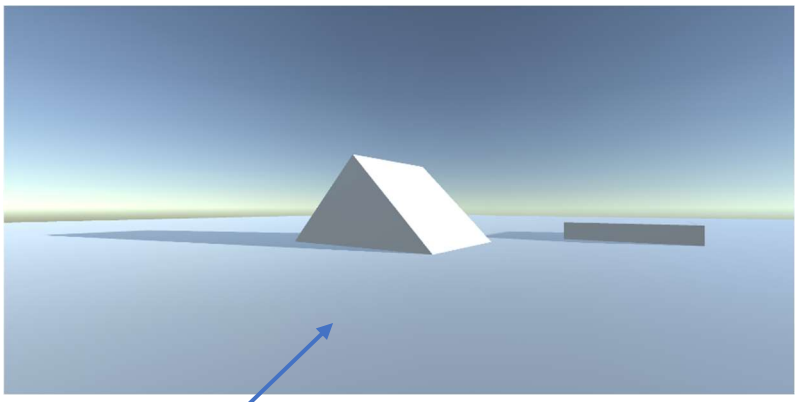
Quits the application when game is built.



Run "QuitGame" function when "Quit" button is clicked.

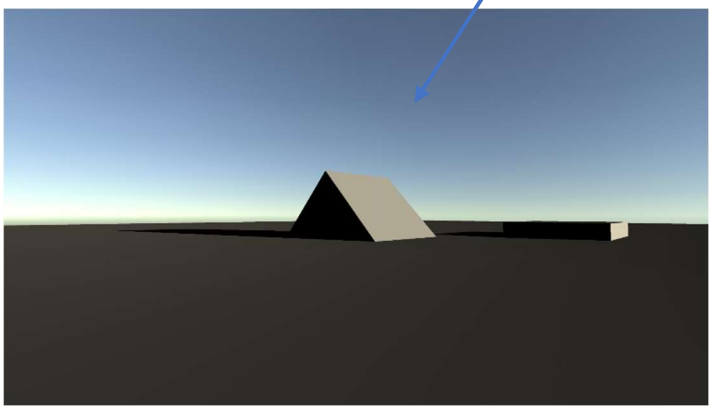
Date: 20/09/22

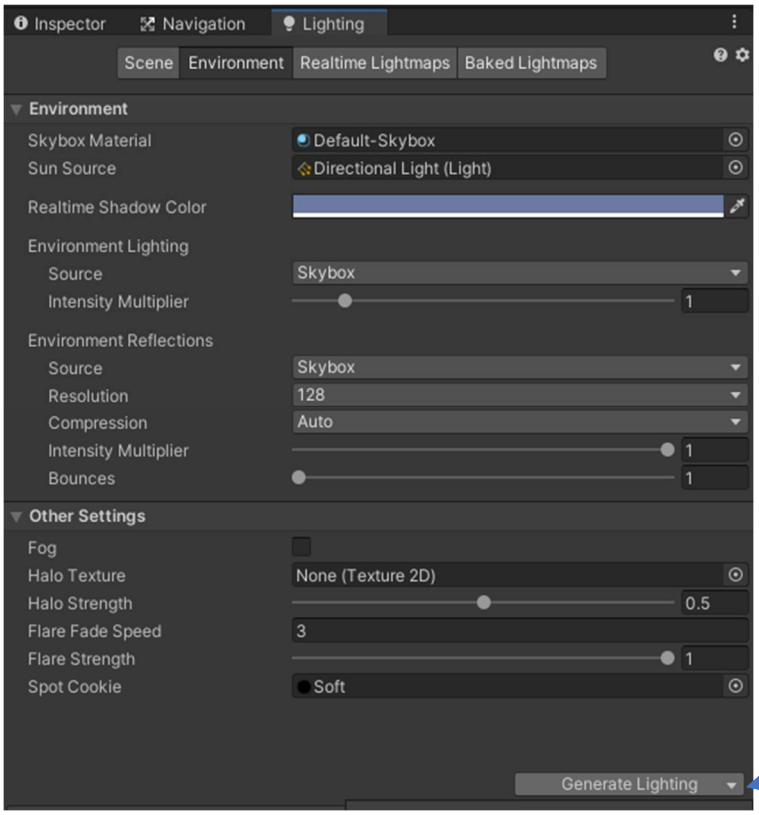
Since there were a lot of features that I could not test in the editor, I wanted to build the game in its current state and check the functionality of the lighting and "QuitGame" function. To do this, I just built the game in WebGL format and hosted it on a GitHub page. I could have simply built it as an executable, but I wanted to be able to access the game from different devices and browsers. The game worked well in this format but the lighting after switching scene was still not rendering properly, so I also built it in "PC, Mac & Linux Standalone" format to test further. The "QuitGame" function worked well, but again the lighting was not rendering properly. I decided to manually generate the scene's lighting in the game editor, which makes lighting files and stores them in a folder. This fixed the problem but might not be a permanent fix.

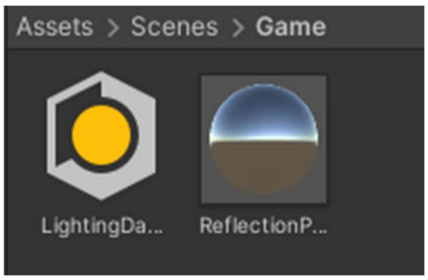


Game when ran in the Unity editor.

Game when built and ran.







This created lighting files in a new folder called “game”.

I pressed “Generate Lighting” at the bottom of this menu, which means that it is not generated every time the scene is loaded.

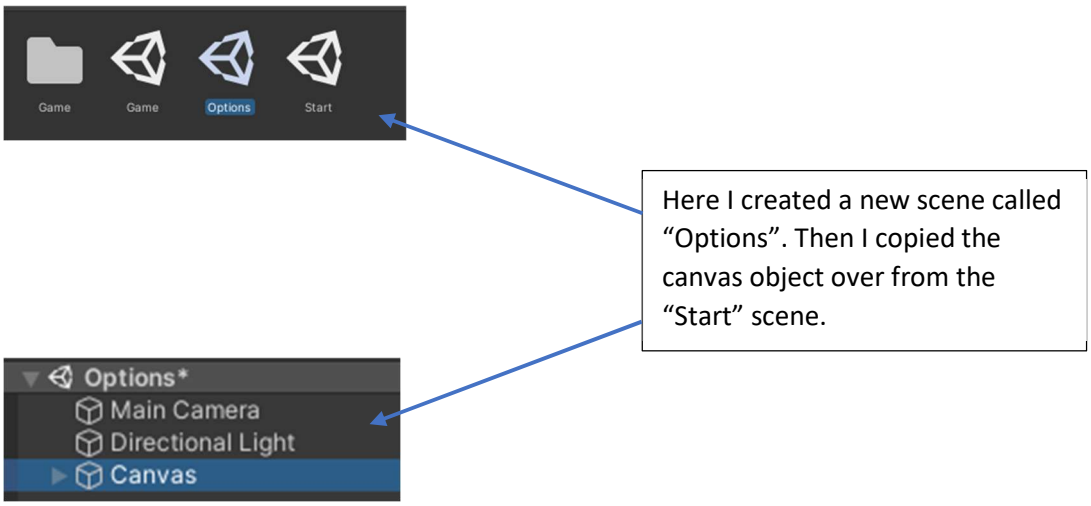
Sources

In this section of development, I used a YouTube tutorial[17] to help me with the main structure of creating an opening screen.

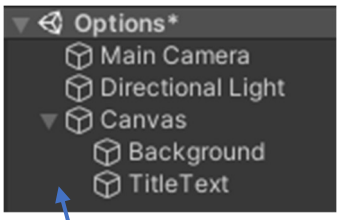
[2] Settings menu

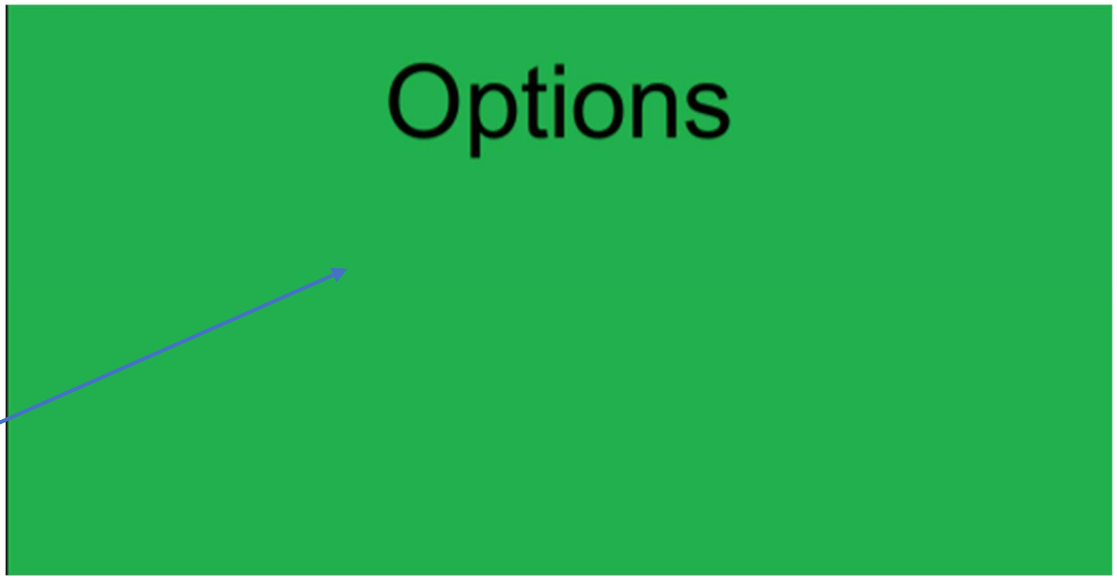
Date: 21/09/22

I decided that the best section to develop next was the options menu. This is because I want to ensure the changes remain throughout scene changes and loads, and it could be harder to implement later on in development once everything is already coded. Initially, I created a canvas similar to the one in the opening screen but recreated my option menu mock up from the design section. I added a few sliders, drop down menus, a toggle box and a return button. Again, I had to set all the anchors properly so that it would scale well with resolution. I was having an issue where I could not interact with the menu, but I fixed it by moving over the "EventSystem" object, which I initially forgot to include. From here I made the sliders look nicer and less out of place. After this, I did the same with the drop-down menus and toggle. I then added an empty "OptionsMenu" object that I could use similarly to the "MainMenu" object in the "Start" scene. I can add a new script component to this object and begin programming. This entire process can be seen below as annotated screenshots.

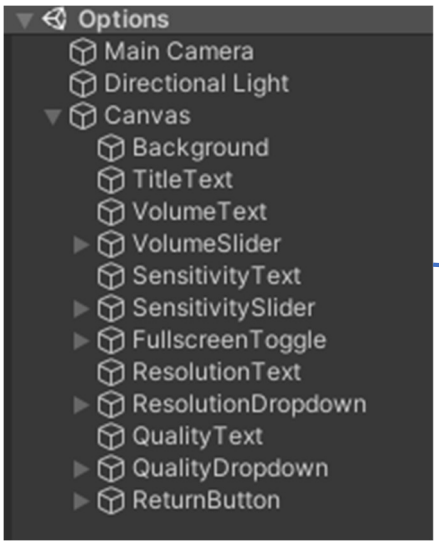


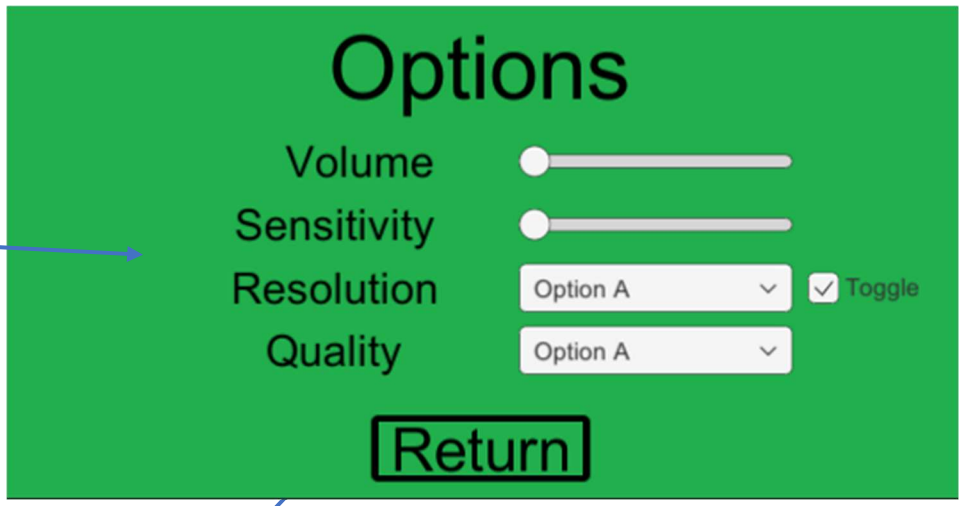
[17] (Thirslund, START MENU in Unity 2017)



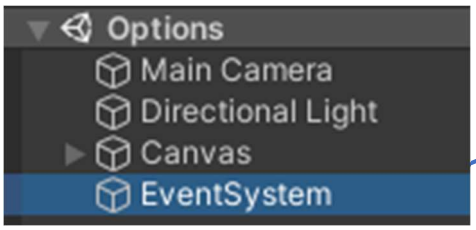


Here I deleted all the unnecessary objects within the canvas and was left with this screen.

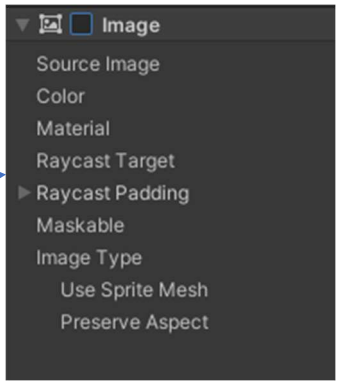




After adding sliders, drop down menus, a toggle box and a return button, this was the result.

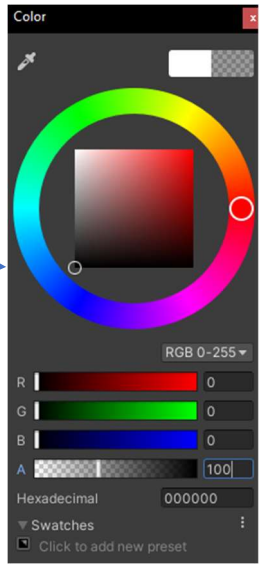


Added in the "EventSystem" object.

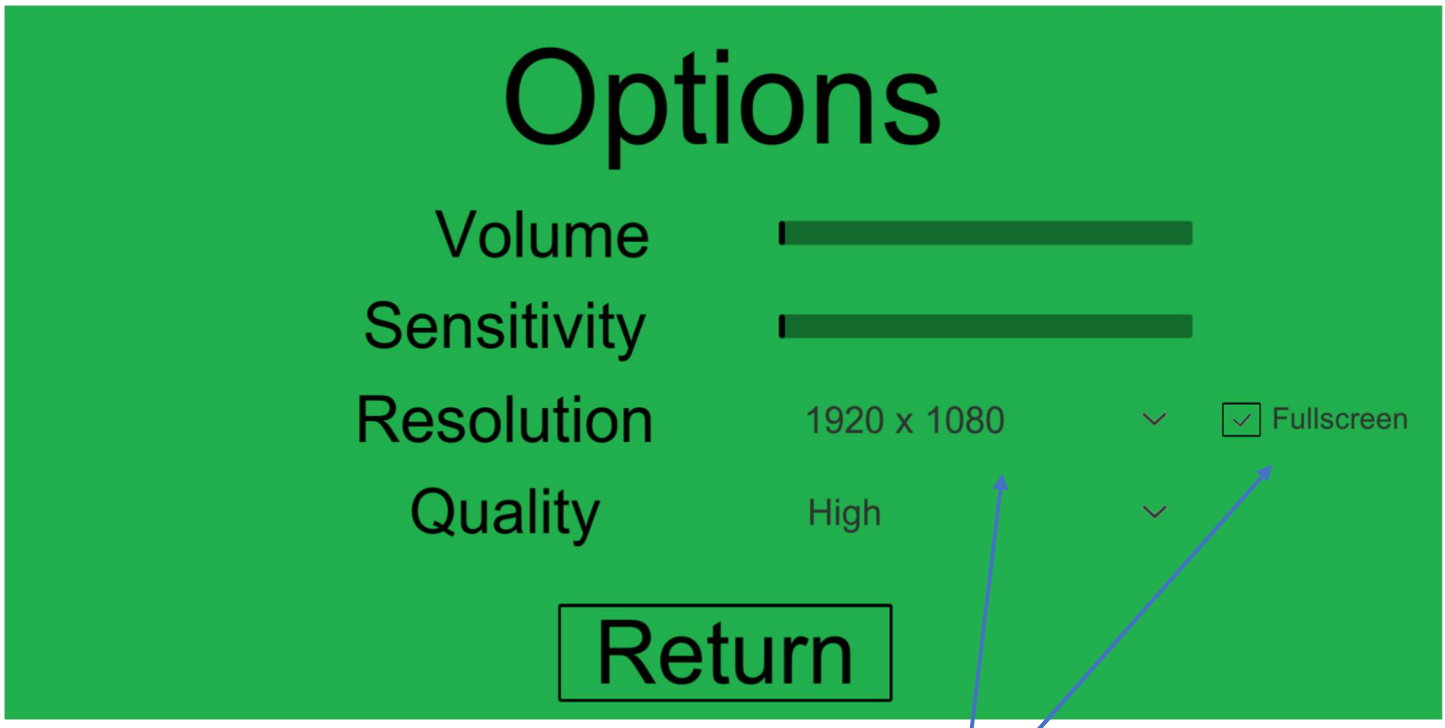


Disabled image component of slider handle.

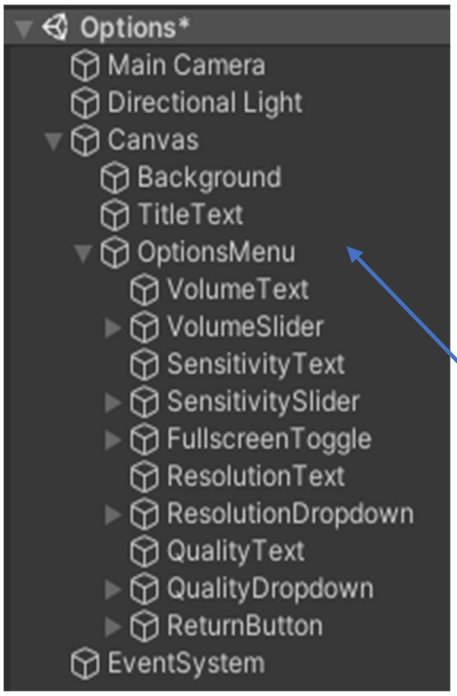
Changed fill colour to black with 255 alpha.



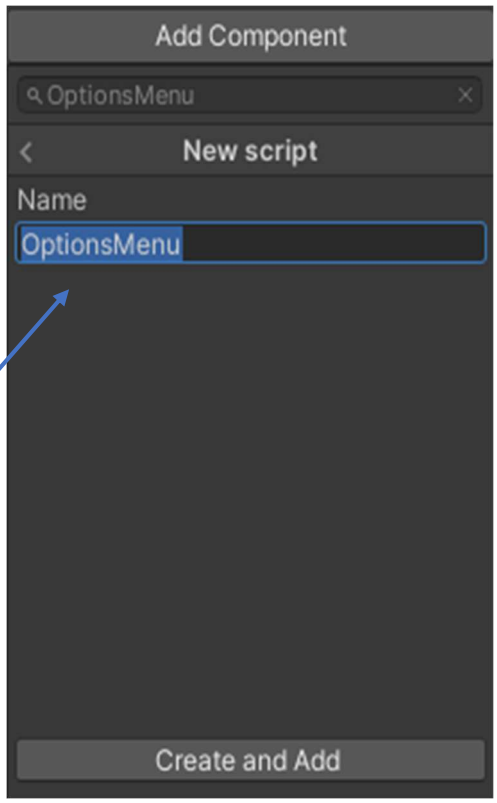
Changed background colour to black with 100 alpha.



Changed colours and alpha channels of the drop-down menus and toggle.



Added an “OptionsMenu” parent object to make my life easier when writing code for the UI elements. Created a new script called “OptionsMenu” and attached it to this object.

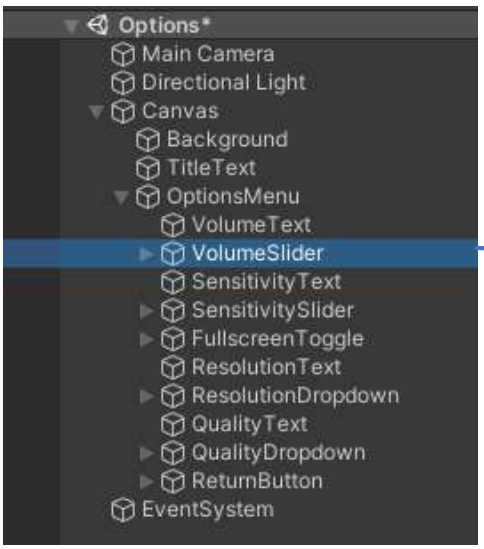


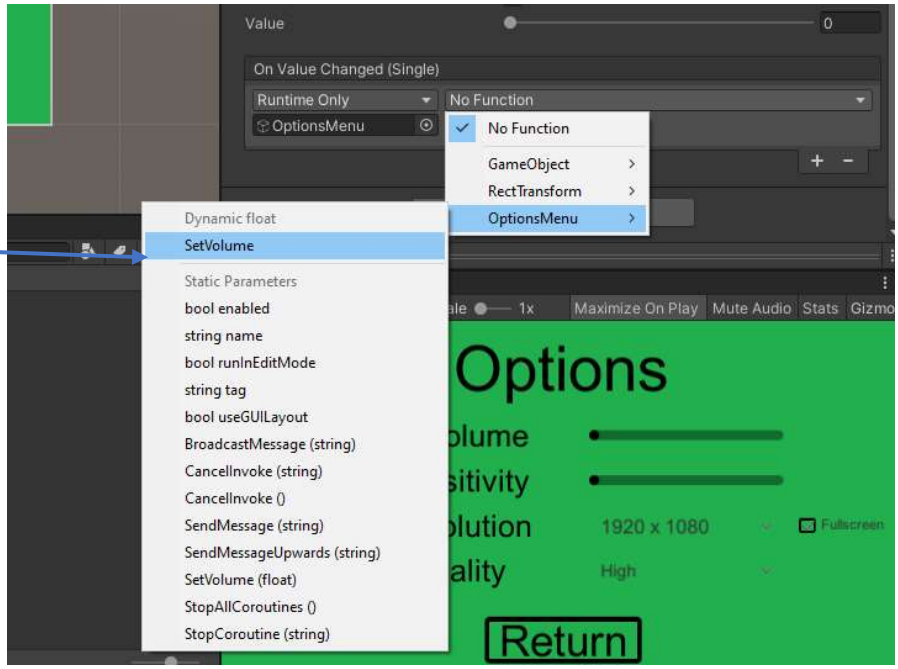
Date: 22/09/22

From here I could begin to add functionality to this menu. I started with the volume slider, creating a new function for changing the volume. Then I linked this function to the volume slider, and had to create a new audio mixer, which I named “MainMixer”. To make this audio mixer accessible from scripts, I had to “expose” it’s volume variable. Once the function was complete, and the audio mixer was linked up to the script, the slider functioned. However, it was only changing the volume by one. I had to match the slider units to the audio mixer units, which were from 0 to -80. Now this was working, I moved on to the quality drop down. The first step here was to align my drop-down options to the Unity quality options. From here, I could create a function that received the index of the game quality selected from the drop down, and then set this as the game’s quality in the function. Finally, I just had to link up the drop-down menu and the script. From here, I began to work on the “Fullscreen” check box. I created a new function within the “OptionsMenu” script for this. I then had to link this function to the check box. Now this was working correctly, I could begin to work on the resolution drop down menu. The values that I entered into the resolution drop down menu were just placeholders, since I need to get Unity to find out the local machine’s compatible resolutions, then load them into the resolution drop down menu. This worked very well, but it did not automatically load the correct resolution into the drop-down menu. After this was fixed, I loaded the selected resolution with a separate function. Once this was completed, I linked the “ResolutionDropdown” menu to the function. Now test 5, 7, 8, and 9 from the design section should be passed. This process is documented below.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class OptionsMenu : MonoBehaviour
6   {
7       // function to change the game volume
8       public void SetVolume(float volume)
9       {
10
11      }
12  }
13
```

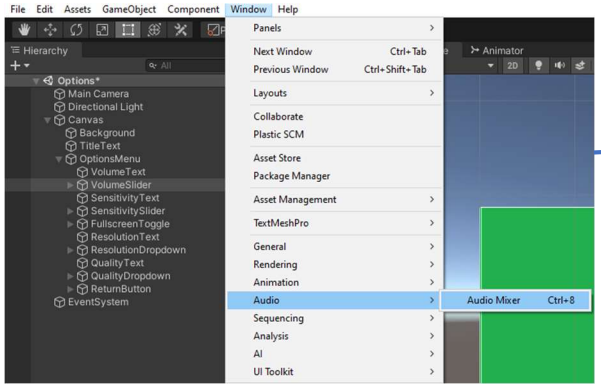
Created an empty “SetVolume” function in the “OptionsMenu” script.

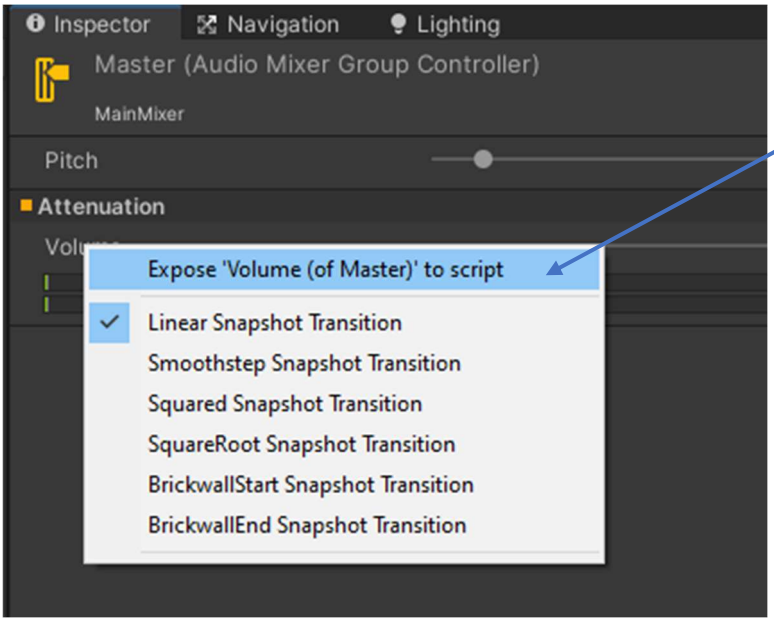




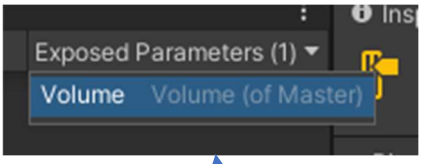
Linked the "VolumeSlider" object to the "SetVolume" function.

Created new "Audio Mixer" and called it "MainMixer"





Exposed the “Volume” variable to scripts so that I can edit it through the “OptionsMenu” program.



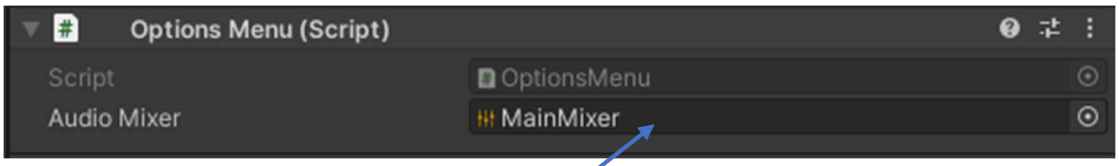
I then renamed this exposed parameter to “Volume”

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Audio;
5
6  public class OptionsMenu : MonoBehaviour
7  {
8      // reference to audio mixer
9      public AudioMixer audioMixer;
10
11     // function to change the game volume
12     public void SetVolume(float volume)
13     {
14         // sets the audio mixer volume to slider value
15         audioMixer.SetFloat("Volume", volume)
16     }
17 }
18
```

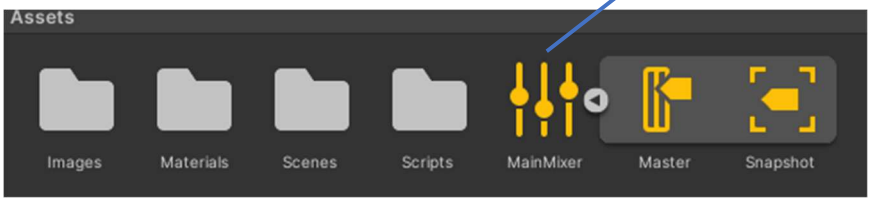
Required to reference the audio mixer.

Value of slider

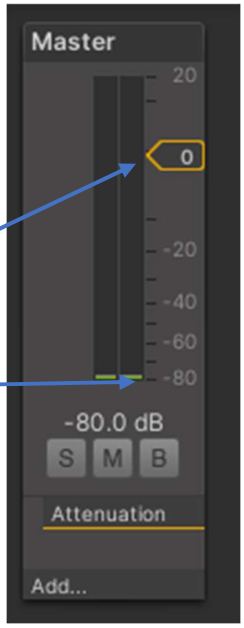
Sets value of audio mixer to the value in the volume slider.

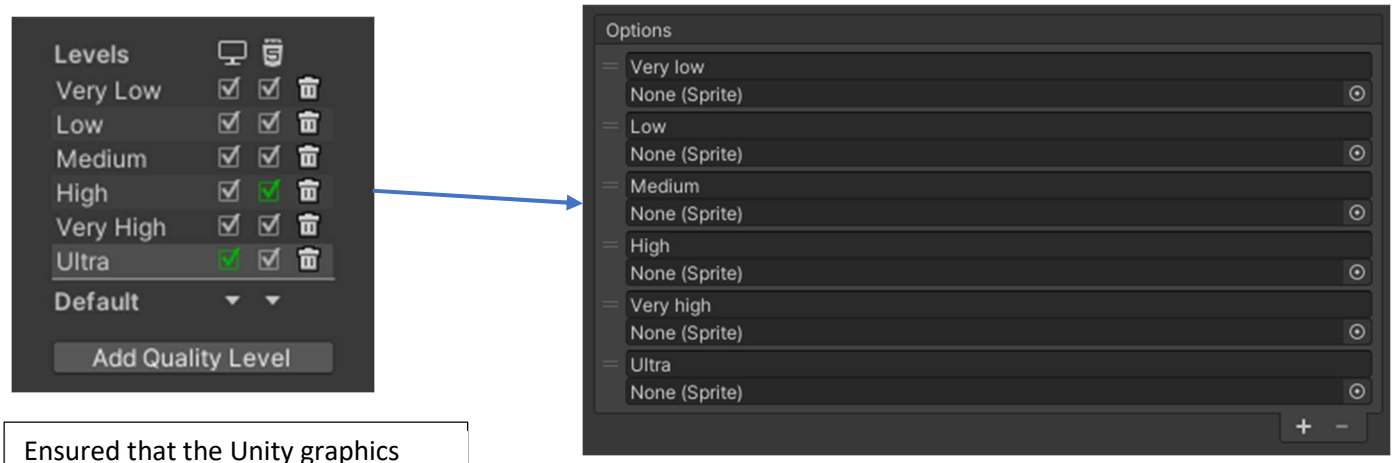


Link the “MainMixer” to the “OptionsMenu” script.



Here I aligned the max/min values of the volume slider to the max/min values of the audio mixer.

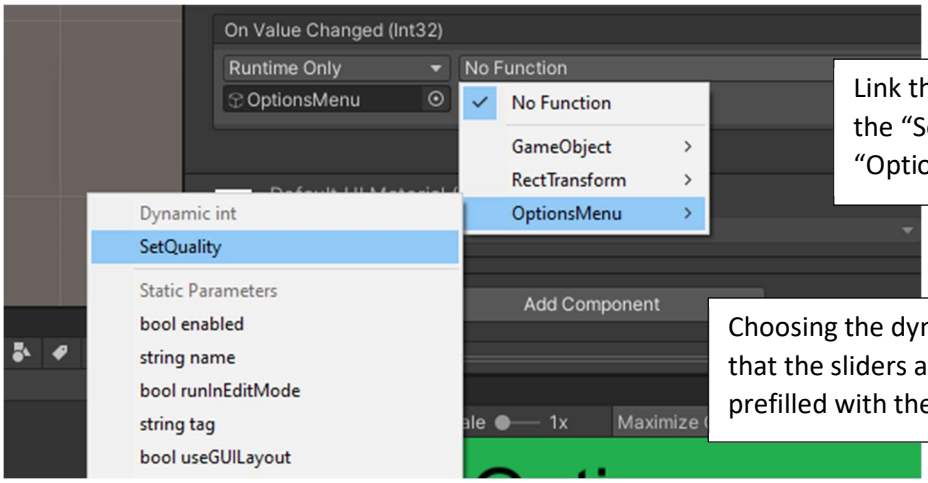




Ensured that the Unity graphics options, and the drop-down graphics options were the same.

```
public class OptionsMenu : MonoBehaviour
{
    // reference to audio mixer
    public AudioMixer audioMixer;

    // function to change the game volume
    public void SetVolume(float volume)
    {
        // sets the audio mixer volume to slider value
        audioMixer.SetFloat("Volume", volume);
    }

    // function to change the game quality
    public void SetQuality(int qualityIndex)
    {
        // sets the game quality to desired level
        QualitySettings.SetQualityLevel(qualityIndex);
    }
}
```

Index of the element chosen from the drop-down menu.



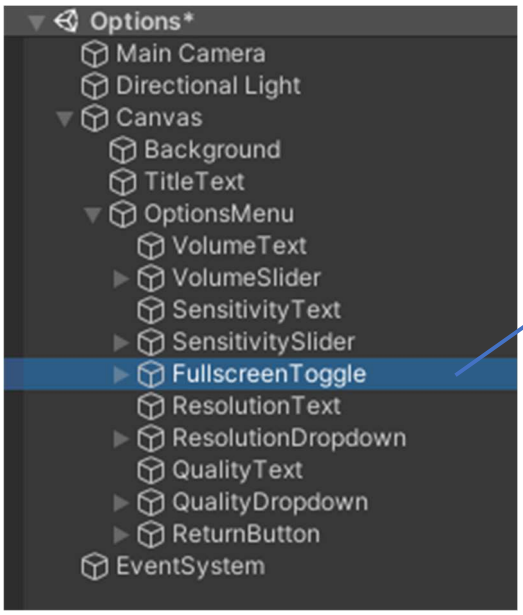
Link the “QualityDropdown” to the “SetQuality” function in the “OptionsMenu” script.

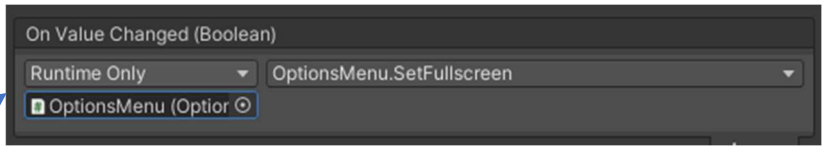
Choosing the dynamic options ensures that the sliders and dropdowns are prefilled with the variables.

```
public class OptionsMenu : MonoBehaviour
{
    // reference to audio mixer
    public AudioMixer audioMixer;

    // function to change the game volume
    public void SetVolume(float volume)
    {
        // sets the audio mixer volume to slider value
        audioMixer.SetFloat("Volume", volume);
    }

    // function to change the game quality
    public void SetQuality(int qualityIndex)
    {
        // sets the game quality to desired level
        QualitySettings.SetQualityLevel(qualityIndex);
    }

    // function to toggle fullscreen mode
    public void SetFullscreen(bool isFullscreen)
    {
        Screen.fullScreen = isFullscreen
    }
}
```

Value stored in toggleable check box.

Sets display to full screen according to value stored in check box.





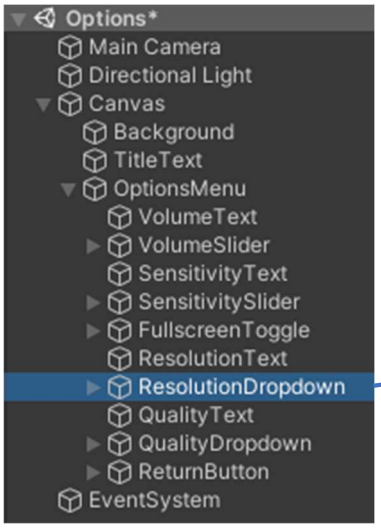
Linked the “FullscreenToggle” object to the “SetFullscreen” function within the “OptionsMenu” script.

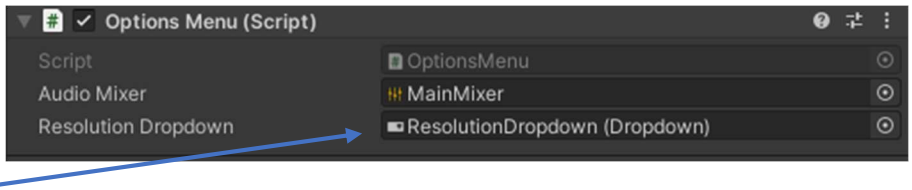
```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Audio;
5  using UnityEngine.UI;
6
7  public class OptionsMenu : MonoBehaviour
8  {
9      // reference to audio mixer
10     public AudioMixer audioMixer;
11
12     // reference to resolution dropdown
13     public Dropdown resolutionDropdown;
14
15     // array to store the resolutions
16     Resolution[] resolutions;
17
18     // function that runs at beginning
19     void Start()
20     {
21         // fill array with resolutions
22         resolutions = Screen.resolutions;
23         // remove any options already in dropdown
24         resolutionDropdown.ClearOptions();
25         // list to store string versions of resolutions
26         List<string> options = new List<string>();
27
28         // loops through all the resolutions
29         for (int i = 0; i < resolutions.Length; i++)
30         {
31             // must be in format: "width" + " x " + "height"
32             string option = resolutions[i].width + " x " + resolutions[i].height;
33             // add to options list
34             options.Add(option);
35         }
36
37         resolutionDropdown.AddOptions(options);
38     }
39
```

Required to reference UI elements.

This code within the “Start()” function removes all the options in the resolution drop down menu, then fills it back up with all the compatible resolutions for the host machine.

Accepts a list of strings, not an array of resolutions.



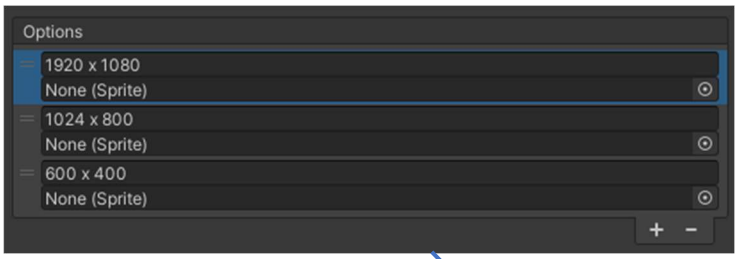


Reference the “ResolutionDropdown” object for the “OptionsMenu” script.

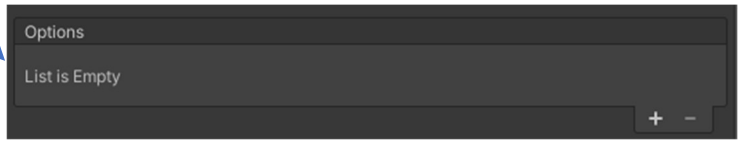
```
// function that runs at beginning
void Start()
{
    // fill array with resolutions
    resolutions = Screen.resolutions;
    // remove any options already in dropdown
    resolutionDropdown.ClearOptions();
    // list to store string versions of resolutions
    List<string> options = new List<string>();

    // integer to store the user's current resolution (indexed)
    int currentResolutionIndex = 0

    // loops through all the resolutions
    for (int i = 0; i < resolutions.Length; i++)
    {
        // must be in format: "width" + " x " + "height"
        string option = resolutions[i].width + " x " + resolutions[i].height;
        // add to options list
        options.Add(option);

        // check whether current resolution is equal to resolutions[i]
        if (resolutions[i].width == Screen.currentResolution.width &&
            resolutions[i].height == Screen.currentResolution.height)
        {
            // if so, save it as the current resolution index
            currentResolutionIndex = i;
        }
    }

    // add the options to the dropdown menu
    resolutionDropdown.AddOptions(options);
    // set the default options to the current resolution
    resolutionDropdown.value = currentResolutionIndex;
    // refresh the dropdown menu
    resolutionDropdown.RefreshShownValue();
}
```

This code now checks for the current resolution, and if it is available, it will set it as the default value in the resolution dropdown menu.

In C# you cannot compare the resolution variables directly, but instead have to split them up into width and height and then compare them.



Since the dropdown menu is now populated automatically, I removed the placeholder values from the menu.

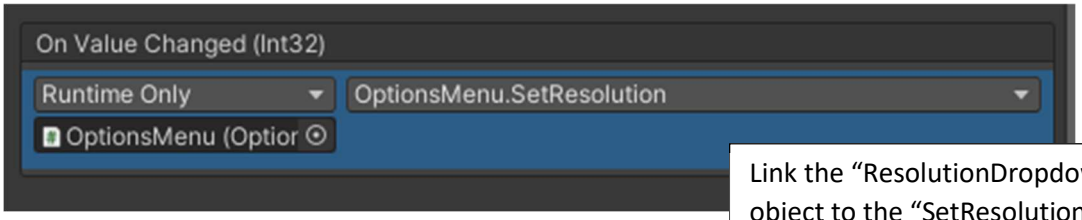


```
public void SetResolution(int resolutionIndex)
{
    Resolution resolution = resolutions[resolutionIndex];
    // accepts in format: width (int), height (int), fullscreen (bool)
    Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
}
```

Selected resolution from dropdown menu but indexed.

Creates a “resolution” variable and makes the value equal to the selected resolution.

Finally sets the screen resolution to the resolution selected by the player.



Link the “ResolutionDropdown” object to the “SetResolution” function within the “OptionsMenu” script.

Date: 23/09/22

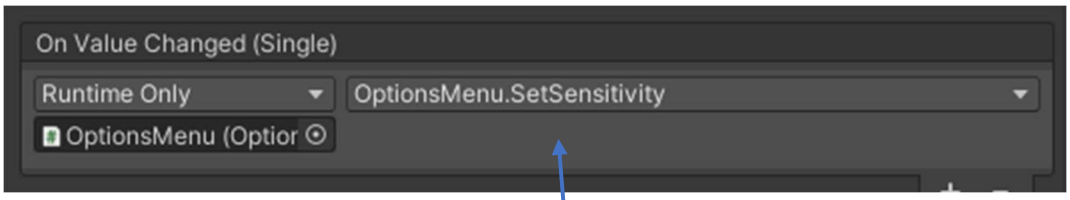
Finally, now I could implement the sensitivity setting, which I will do by directly changing the sensitivity variable from the “MouseLook” program. I had to set the min/max values of the slider to be minimum 100 and maximum 1000, which I thought worked well as a range for the sensitivity after testing it inside the editor. Now, I could make a new function to directly edit the value within the MouseLook script. From here, I linked the function with the sensitivity slider. Then I made the “OptionsButton” object within the “Start” scene point to the “Options” scene, and the “ReturnButton” within the “Options” scene point to the “Start” scene. This would let the user easily go between the menus. I built the game, and tried to change the sensitivity slider, which had no effect on my mouse sensitivity within the “Game” scene. I think it was due to the fact that the “Game” scene had not been loaded yet. With this in mind, I tried to initialise and store the “mouseSensitivity” variable within the “MainMenu” script instead of the “MouseLook” script, accessing it through the other scripts. This way it would be created as soon as the user loads the game, since the “Start” script automatically loads the “MainMenu” script. Unfortunately, this did not work either. My final attempt included trying to create a new function to set the “mouseSensitivity” variable within the “MouseLook” script and called it from the “OptionsMenu” script. This approach did not work either, and I decided that the sensitivity slider was not that important, considering it only effected and only will affect the “MouseLook” script. I might come back to this feature later in development, but for now I am going to leave it. Now test 4 and 10 should be passed from the design section. This process is documented below, in the form of annotated screenshots.

```
// reference to MouseLook script
public MouseLook mouseLook;
```

Here, I attempted to create a reference to the "MouseLook" script within the "Game" scene. I named this reference "mouseLook".

```
public void SetSensitivity(float sensitivity)
{
    mouseLook.mouseSensitivity = sensitivity;
}
}
```

Here I tried to set the mouse sensitivity to whatever value the slider returns.



At this point, I linked up the "SensitivitySlider" object to the "SetSensitivity" function within the "OptionsMenu" script.

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

Here I edited the "OptionsMenu" script.

```
public void Return()
{
    SceneManager.LoadScene("Start");
}
```

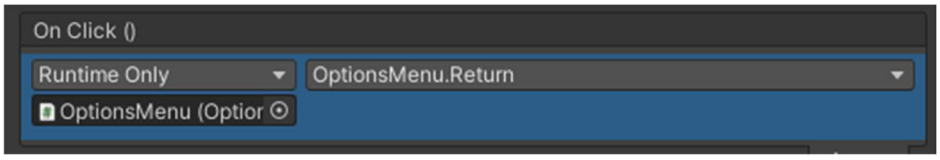
Adding this line allowed me to change scenes within the script.

For now, I just made the "Return" button in the options menu point back to the start menu, but later I will have to change this so that it points to the previous scene.

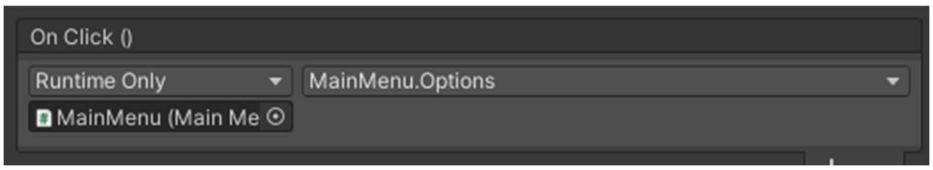
Here, I edited the "MainMenu" script so that the "Options" button points to the "Options" scene.

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

The last step here was to link the buttons to their respective functions within their scripts.



Linked the "ReturnButton" object with the "Return" function within the "OptionsMenu" script.



Linked the "OptionsButton" object with the "Options" function within the "MainMenu" script.

The solution that I had implemented previously did not work. I believe it was due to the fact that I was trying to edit a variable within a scene that had not been loaded yet, since the options menu was opened and used before the “Game” scene was loaded.

Here, I tried to create and initialise the “mouseSensitivity” variable within the “MainMenu” script, and then reference it from both the “OptionsMenu” script and the “MouseLook” script. This did not work either and I am not completely sure why.

Initialised the “mouseSensitivity” variable within the “MainMenu” script.

```
public class MainMenu : MonoBehaviour
{
    // set up sensitivity variable
    public float mouseSensitivity = 1000f;
```

```
public class OptionsMenu : MonoBehaviour
{
    // reference to audio mixer
    public AudioMixer audioMixer;

    // reference to resolution dropdown
    public Dropdown resolutionDropdown;

    // reference to sensitivity variable
    public MainMenu mainMenu;
```

Referenced the “mouseSensitivity” script and then changed the variable to the slider value.

```
public void SetSensitivity(float sensitivity)
{
    mainMenu.mouseSensitivity = sensitivity;
}
```

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

public class MouseLook : MonoBehaviour
{
    // create variable for mouse sensitivity
    public static float mouseSensitivity = 200f;
    // create transform for the player body
    public Transform playerBody;
    // create a variable for the x-rotation
    float xRotation = 0f;

    // start is called before the first frame update
    void Start()
    {
        // lock the cursor to the centre of the screen
        Cursor.lockState = CursorLockMode.Locked;
    }

    public void sliderSensitivity(float sliderIndex)
    {
        mouseSensitivity = sliderIndex;
    }

    // update is called once per frame
    void Update()
    {
        // creates variables to store the mouse axis
        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;

        // every frame increase xRotation by mouseY
        xRotation -= mouseY;
        // clamp the rotation
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);


        // rotate camera around the x-axis
        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        // rotate the player body around the y-axis
        playerBody.Rotate(Vector3.up * mouseX);
    }
}
```

At this point I decided to try a slightly different method, in which I created the "mouseSensitivity" variable in the "MouseLook" script, and ensured it was static so that it remained constant across all the other scripts.

Then I created a function, which takes an input of a float value, and sets the mouseSensitivity variable to this value.

In the "OptionsMenu" script, I referenced the "MouseLook" script.

```
// reference to MouseLook script
public MouseLook mouseLook;
```

Used the new function and gave it an input of the sensitivity slider value.

```
public void SetSensitivity(float sensitivity)
{
    mouseLook.sliderSensitivity(sensitivity);
    Debug.Log(sensitivity);
}
```

This approach did not yield any results

Sources

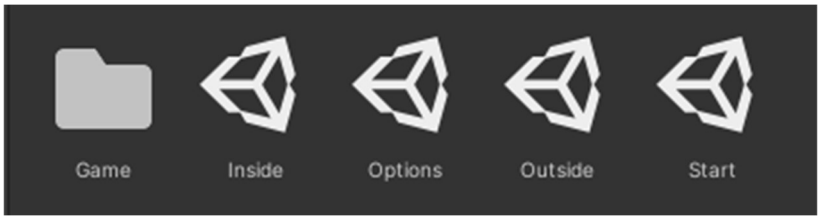
For this section of development, I used a YouTube video[18] as a guide and reference to creating a settings menu.

[18] (Thirslund, SETTINGS MENU in Unity 2017)

[3] Player house

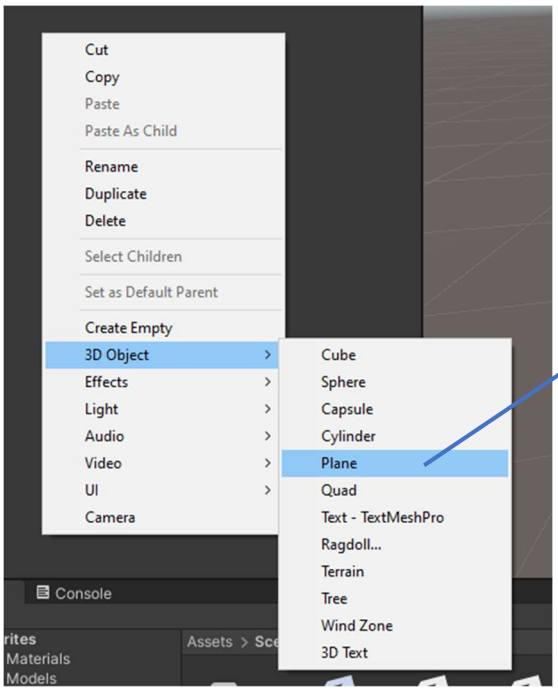
Date: 02/10/22

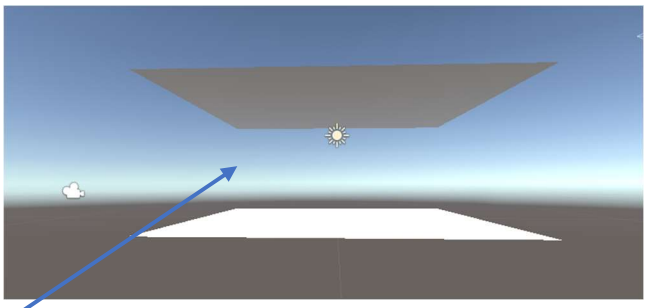
Now that the basic user interface was complete, I thought this was a good time to develop the actual game further. This meant that I had to create the player house and ensure that the player could go in between “Inside” and “Outside” states, following my game loop design. The first step here was to create a new scene called “Inside” and rename the “Game” scene to “Outside”. There were no direct references to the “Game” scene, so I did not have to change any scripts. However, I did have to change the “Game” folder, which includes the baked lighting for the now “Outside” scene, to be called “Outside”. I then loaded the “Inside” scene, and began constructing the interior of the player house. For now, I was only concerned about adding a door, floor, walls and ceiling. I added a “Plane” object to act as the floor, and also to act as a ceiling. I renamed them to “Floor” and “Ceiling”. The “Ceiling” plane is 4 units higher on the y-axis than the “Floor” plane. I then rotated the “Ceiling” object 180 degrees about the x-axis, so that the opaque side was facing inwards. Then I created 4 more planes, to act as the walls. They had to be rotated, scaled and positioned accordingly to fit. They were renamed to “WallLeft”, “WallBack”, “WallRight”, “WallFront”. I put all these objects into an empty parent object called “House”. Then, I copied over the “Character” object from the “Outside” scene into this scene. Then I deleted the other “Main camera” object. I had to give all the objects within the “House” parent object the “Terrain” layer. I didn’t like how small and square the inside was, so I extended it to double the size. This included re-scaling and re-positioning all the objects. The inside of the room was far too bright, so I created a new materials called “Inside” and gave it an “Albedo” colour. I then assigned it to all the “House” objects. Finally, I rotated the character object around so it was facing the back wall when the player loaded the scene. This process is documented below as annotated screenshots.



Created new “Inside” scene and renamed “Game” scene to “Outside”.

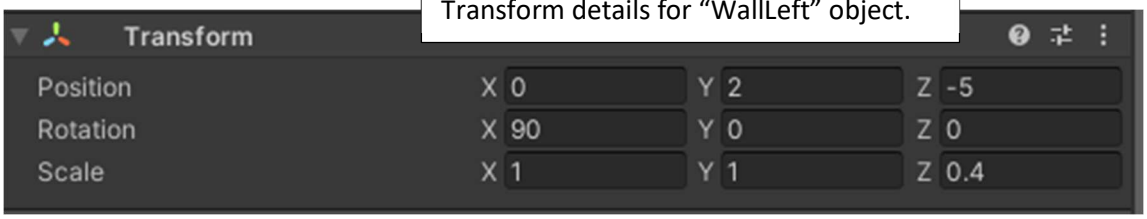
Renamed “Game” lighting folder to “Outside” so it could function.



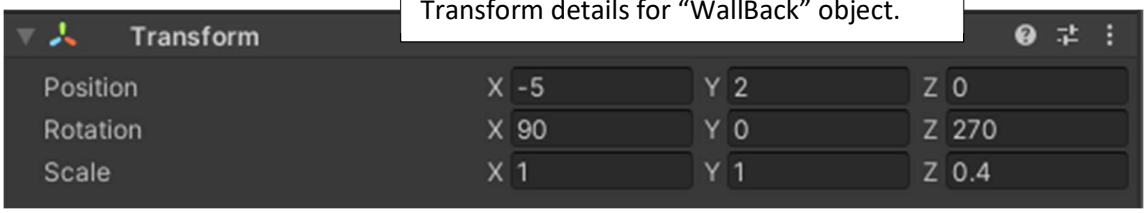


Here I created two new plane objects and ensured one plane had a higher y-axis position than the other. These would act as the room ceiling and floor. I renamed these plane objects to “Floor and Ceiling”.

Transform details for “WallLeft” object.

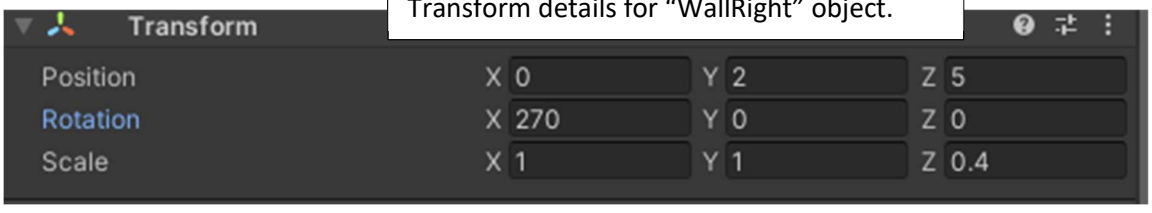


Transform details for “WallBack” object.

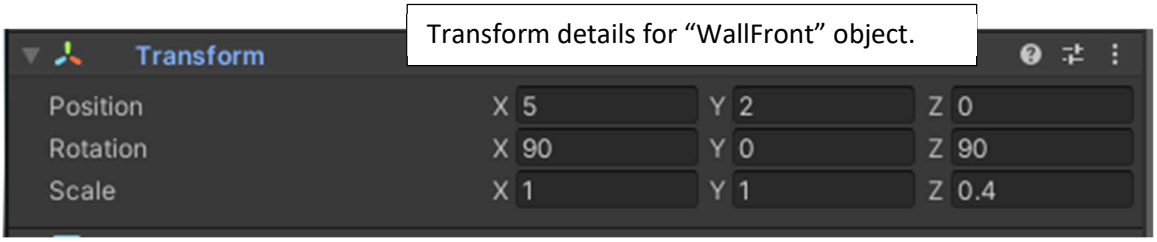


Here I added more plane objects, ensuring they were positioned, rotated and scaled correctly so that the planes formed a room. I renamed the planes to “WallLeft”, “WallBack”, “WallRight” and “WallFront”.

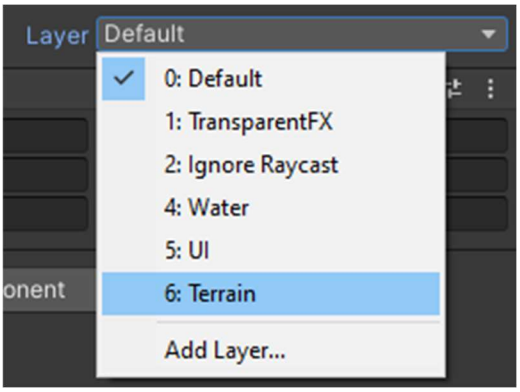
Transform details for “WallRight” object.

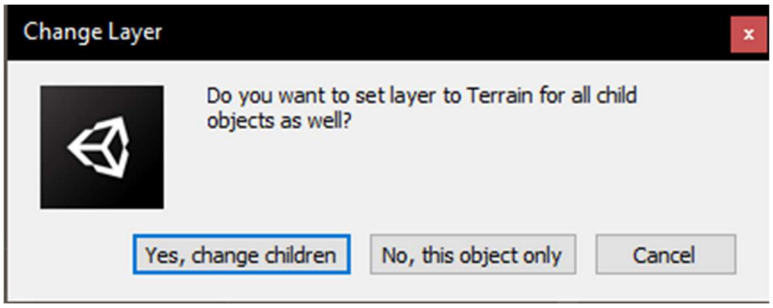


Transform details for “WallFront” object.

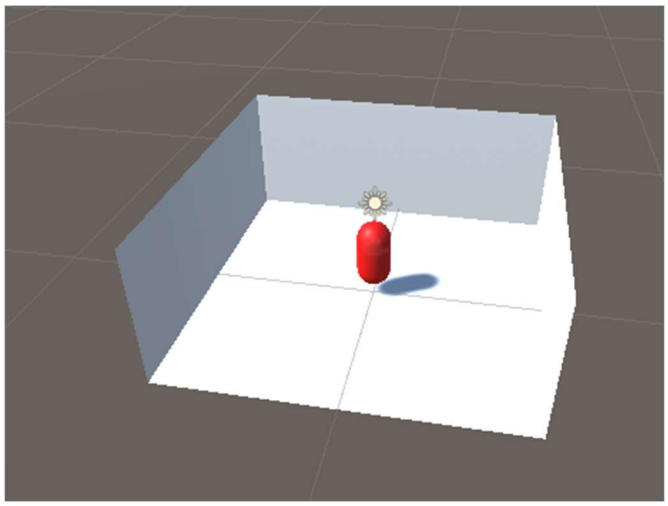


From here, I made a new empty object called “House” and made it a parent to all the components. This includes the “Ceiling”, “Floor”, “WallLeft”, “WallRight”, “WallBack” and “WallFront” objects.

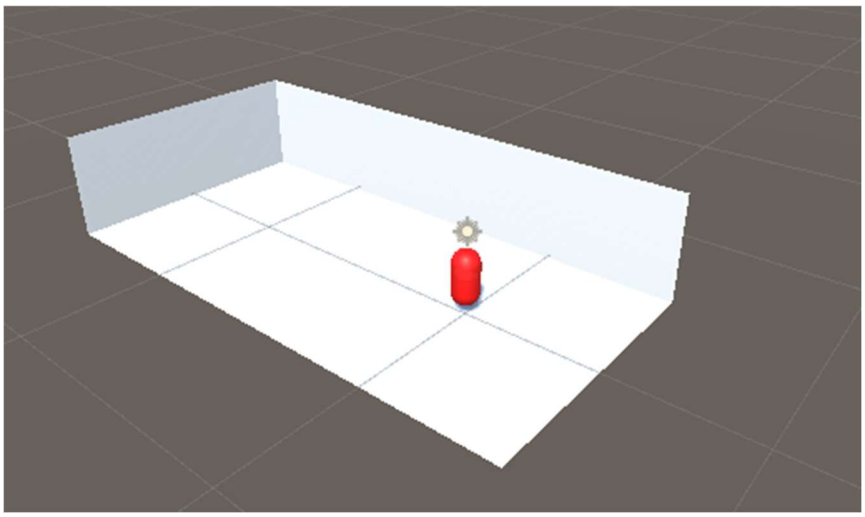




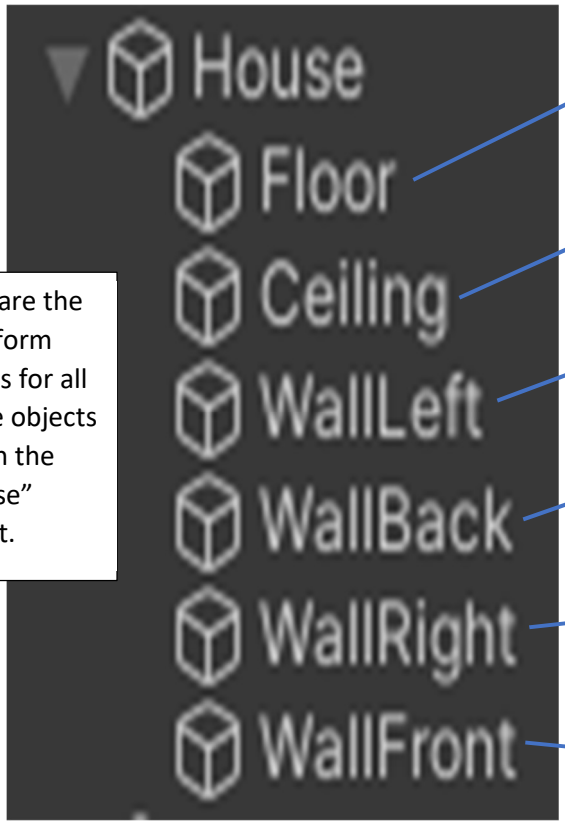
After copying over the "Character" object from the "Outside" scene, I ran into an issue. None of the objects within the "House" object had the "Terrain" layer, which was required to ground the character and allow it to jump. All I had to do was to change the layer of the "House" object to "Terrain" and thus change all the children objects. This fixed the issue.

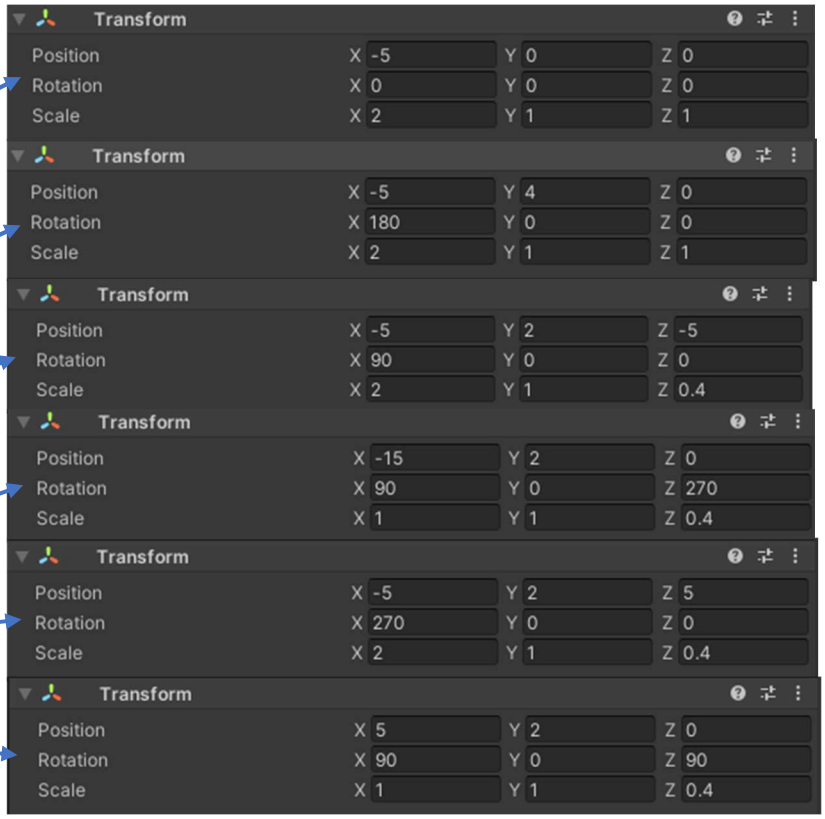


Here, I changed the position, rotation and scale values of the plane objects to double the length of the house, since I thought it was too cramped.

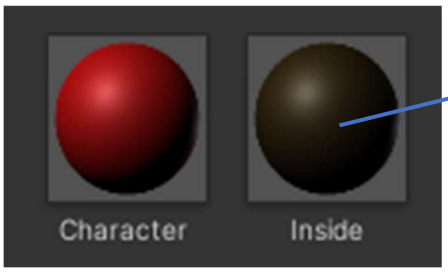


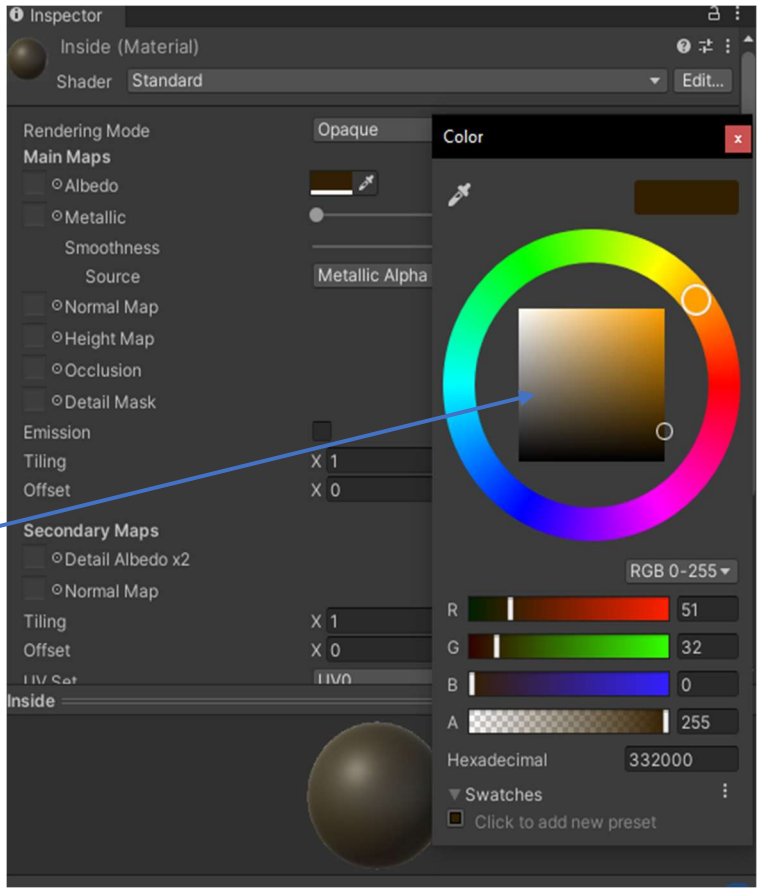
Here are the transform details for all of the objects within the "House" object.





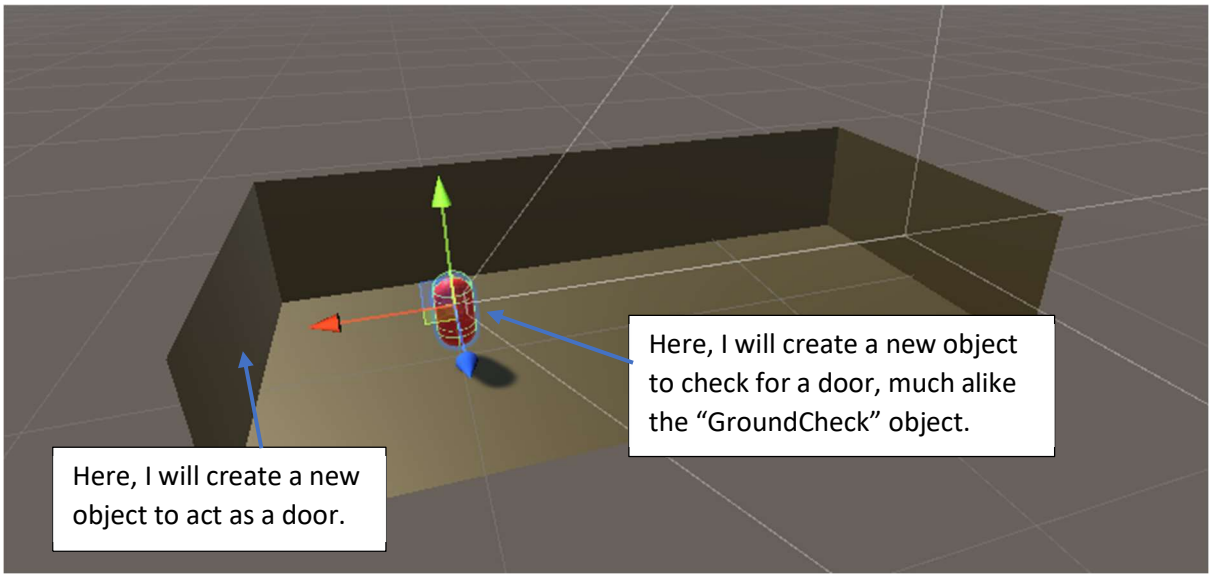
Here, I created a new material called "Inside" and gave it a brown albedo colour, to represent wooden walls.



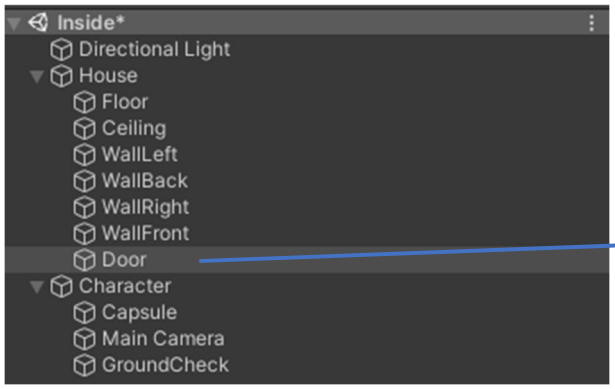


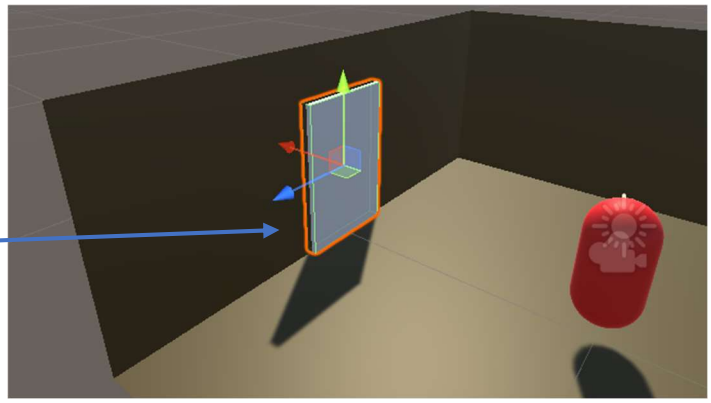
Date: 13/11/22

In this section of development, I wanted to implement a mechanic which would switch the scene from “Inside” to “Outside” and vice-versa whenever the character walked into a door. The first step here was to add a new daughter object in the “House” object, to act as a door. I used a cube object and shaped it to my liking. Furthermore, I created a new material and set the albedo to slightly darker than the rest of the room. Once this was completed, I created a new layer called “Door”, similar to the “Terrain” layer that I used in my character controller section to detect when the player touches the floor. I assigned this layer to the “Door” object. The next step at this point was to create a new script, which I called “DoorDetect”. This script will detect objects in the character’s environment and change the scene accordingly. In this case, it is detecting doors. I could have implemented a solution within the “PlayerMovement” script, but detecting doors is not necessarily linked to moving the character, so I thought it would be best to create a new script. I used the “PlayerMovement” script as a guide however when creating the new “DoorDetect” script, since the code for checking when the character touches the floor is very similar and can be reused. I ran into some syntactical issues when dealing with the scene operations, and resorted to the Unity manuals, one function to return the scene[19], and another to return its name[20]. I also found the pseudocode and diagrams from my design section very helpful when creating this script. Once it was completed, I had to create a “DoorCheck” object which will be at the front of the character object, like in the diagram I drew in the design section. From here I was able to link the script to the character object and reference all the other objects to the script. Once this was complete, I was able to test out this solution. It worked great, with no problems. However, there is only a door on the “Inside” scene, so I went to the “Outside” scene, created a cube to act as the cabin, created another cube to act as a door and coloured them accordingly with their respective materials. I then updated the “Character” object in the “Outside” scene so that the door system could function. I linked and referenced the final parts, and it worked great. The player can now walk between scenes, which marks the end of this sub-section of development. Now test 2 from the design section should be passed. This process can be seen below in the form of annotated screenshots.

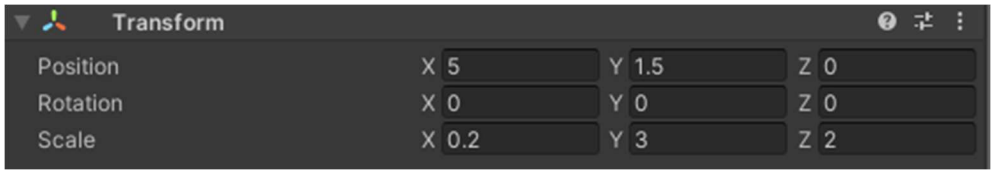


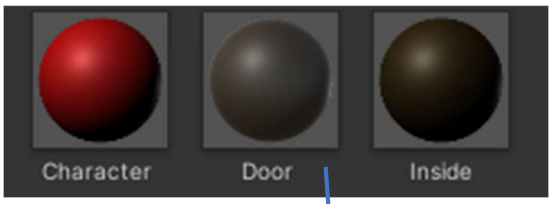
[19] (Unity, SceneManager.GetActiveScene 2022)
[20] (Unity, Scene.name 2022)



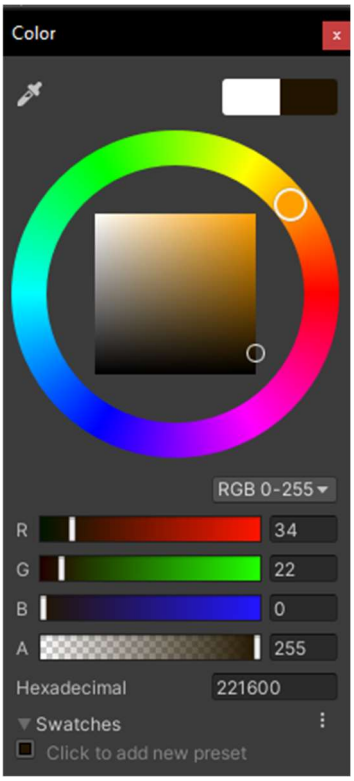


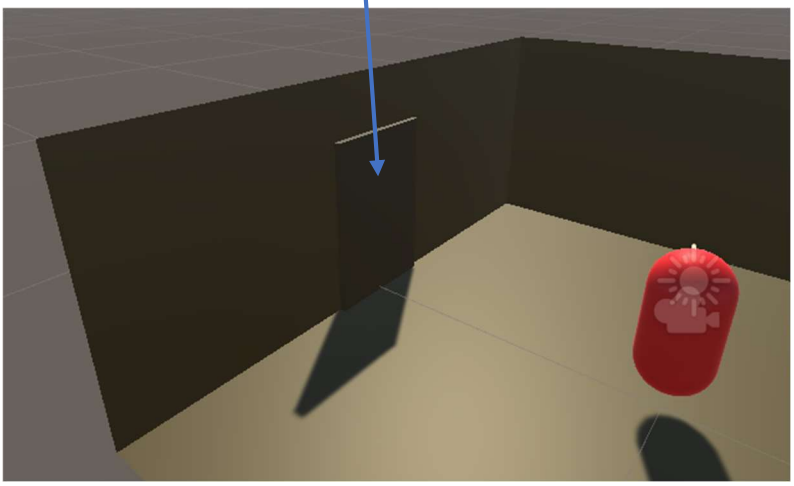
This shows the process of creating the "Door" object within the "Inside" scene. Below are the transform values that I used to create a door shape.

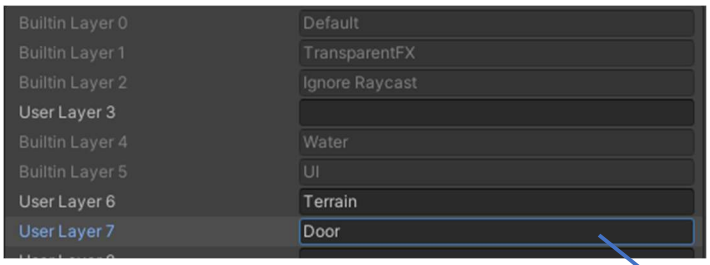




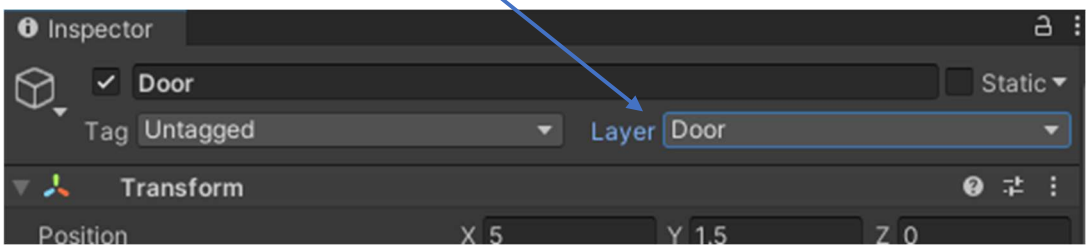
Here, I created a new "Door" material, which I applied to the "Door" object. It is slightly darker than the rest of the room, so that it stands out for the player.







Here, I created a new "Door" layer, and assigned it to the "Door" object.



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

// script to change the scene when
// the character walks into a door

public class DoorDetect : MonoBehaviour
{
    // reference to the DoorCheck object
    public Transform doorCheck;
    // radius of sphere to project
    public float doorDistance = 0.4f;
    // new mask which links to layer
    public LayerMask doorMask;

    bool touchingDoor;

    // Update is called once per frame
    void Update()
    {
        // creates sphere at:
        // player position
        // radius doorDistance
        // mask doorMask
        touchingDoor = Physics.CheckSphere(doorCheck.position, doorDistance, doorMask);

        // changes scene between "Inside" and "Outside" if touching door
        if (touchingDoor)
        {
            string name = SceneManager.GetActiveScene().name;
            switch (name)
            {
                case "Inside":
                    SceneManager.LoadScene("Outside");
                    break;
                case "Outside":
                    SceneManager.LoadScene("Inside");
                    break;
                default:
                    Debug.Log("Error occurred");
                    break;
            }
        }
    }
}
```

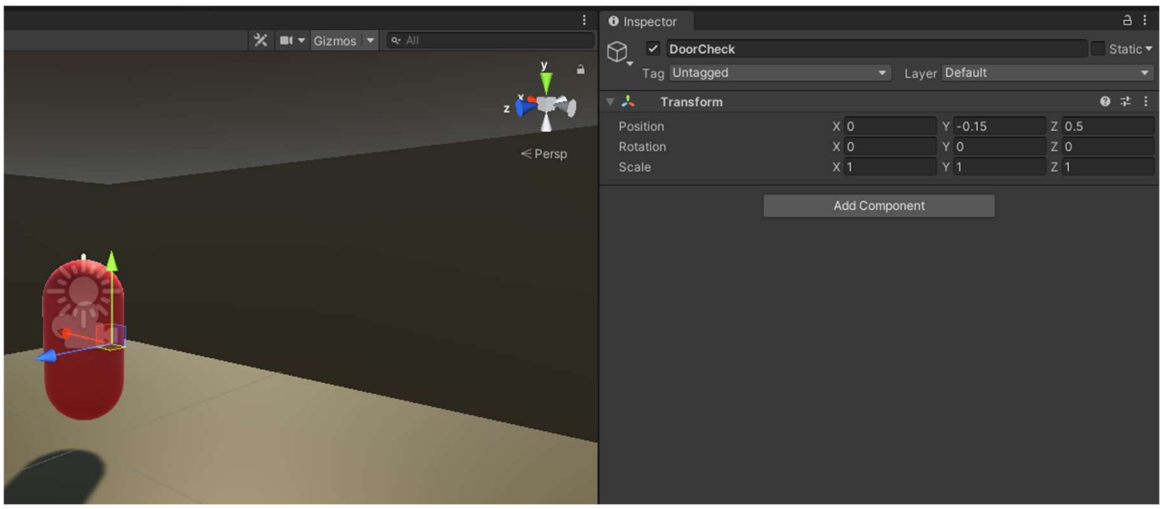
I had to use the "UnityEngine.SceneManagement" library since I will be changing scenes within this script.

I deleted the "Start" function, since it was not necessary here.

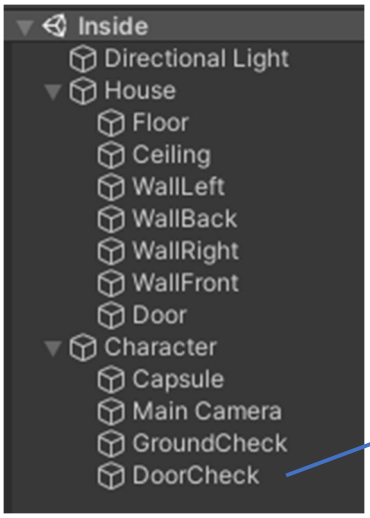
Here, a sphere is projected at the "DoorCheck" object, to detect for any doors that the player is standing next to.

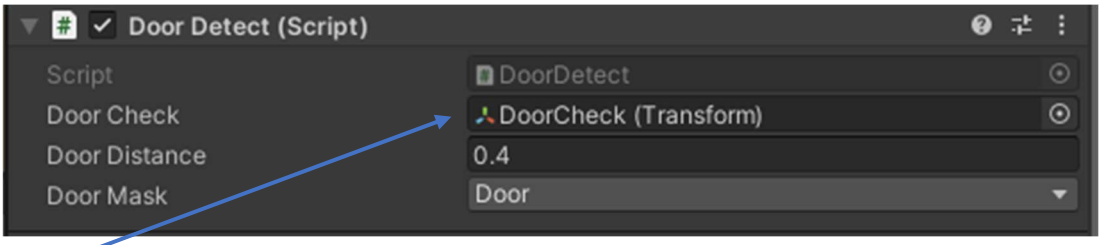
Returns a string of the scene's name.

This switch statement checks whether the player is in the "Inside" or "Outside" scene and swaps them over if the character is in contact with a door object.

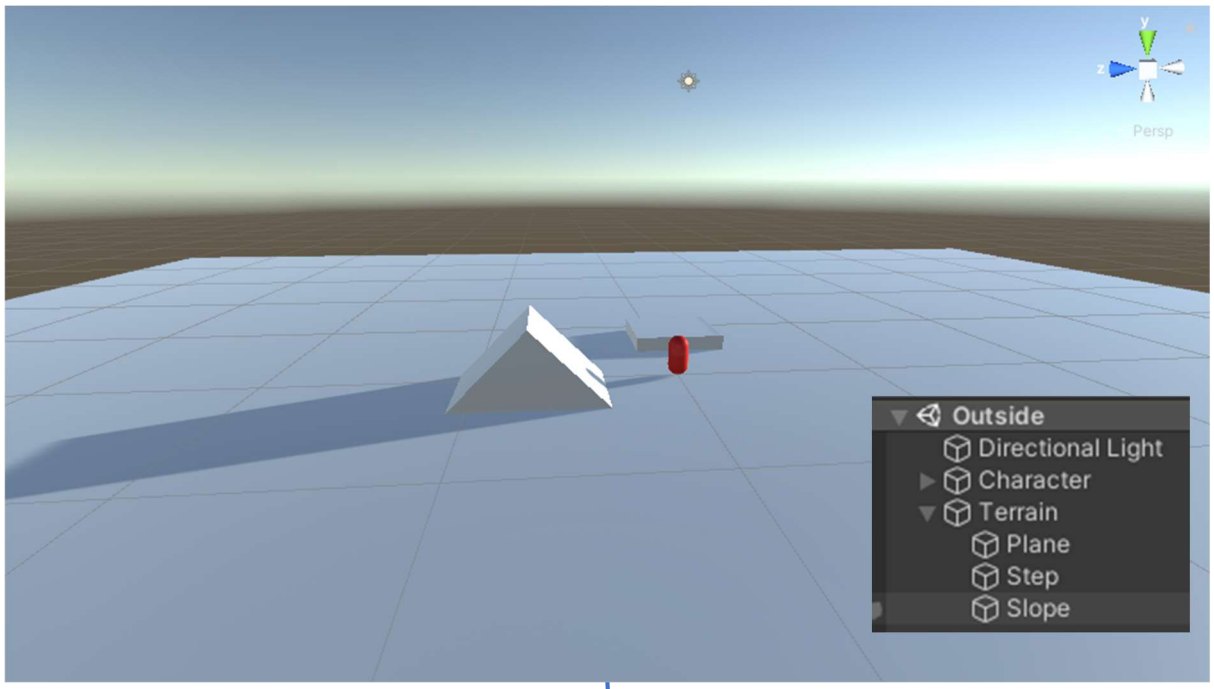


This screenshot shows the location of the "DoorCheck" object, and its respective transform values.

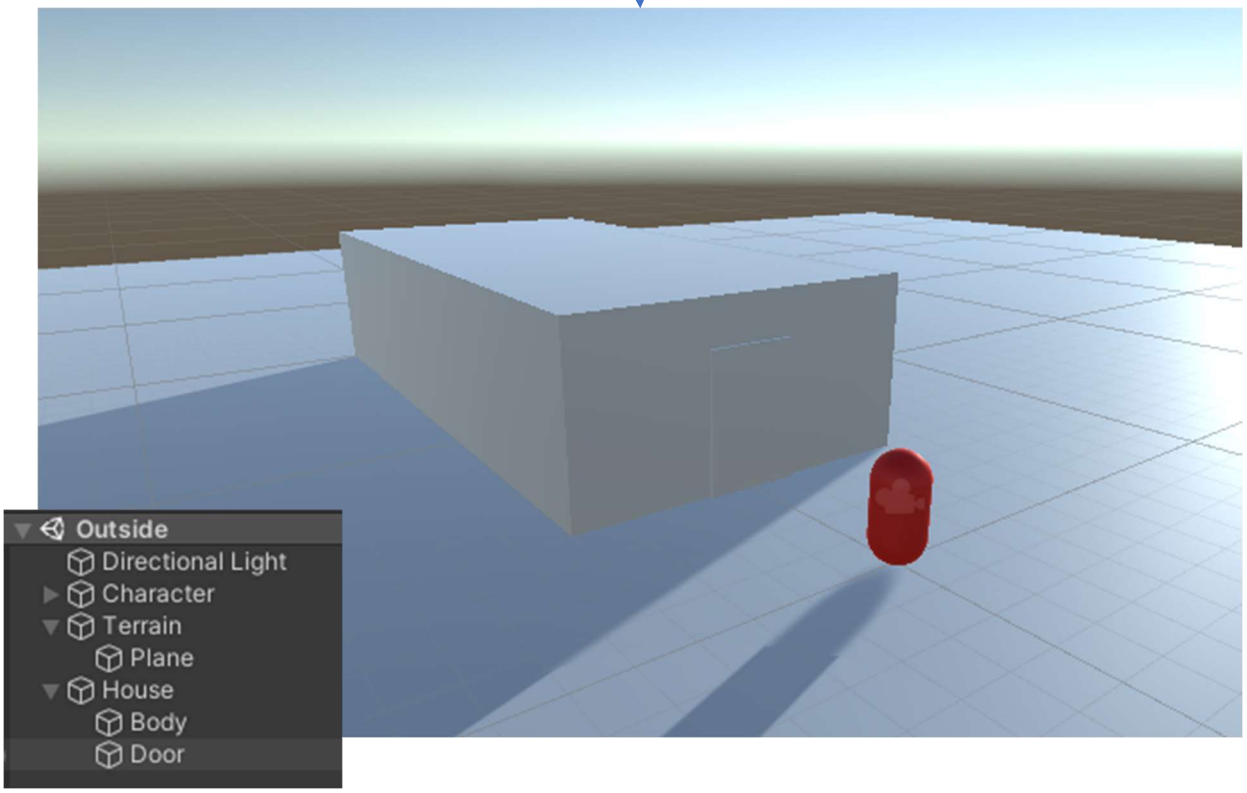


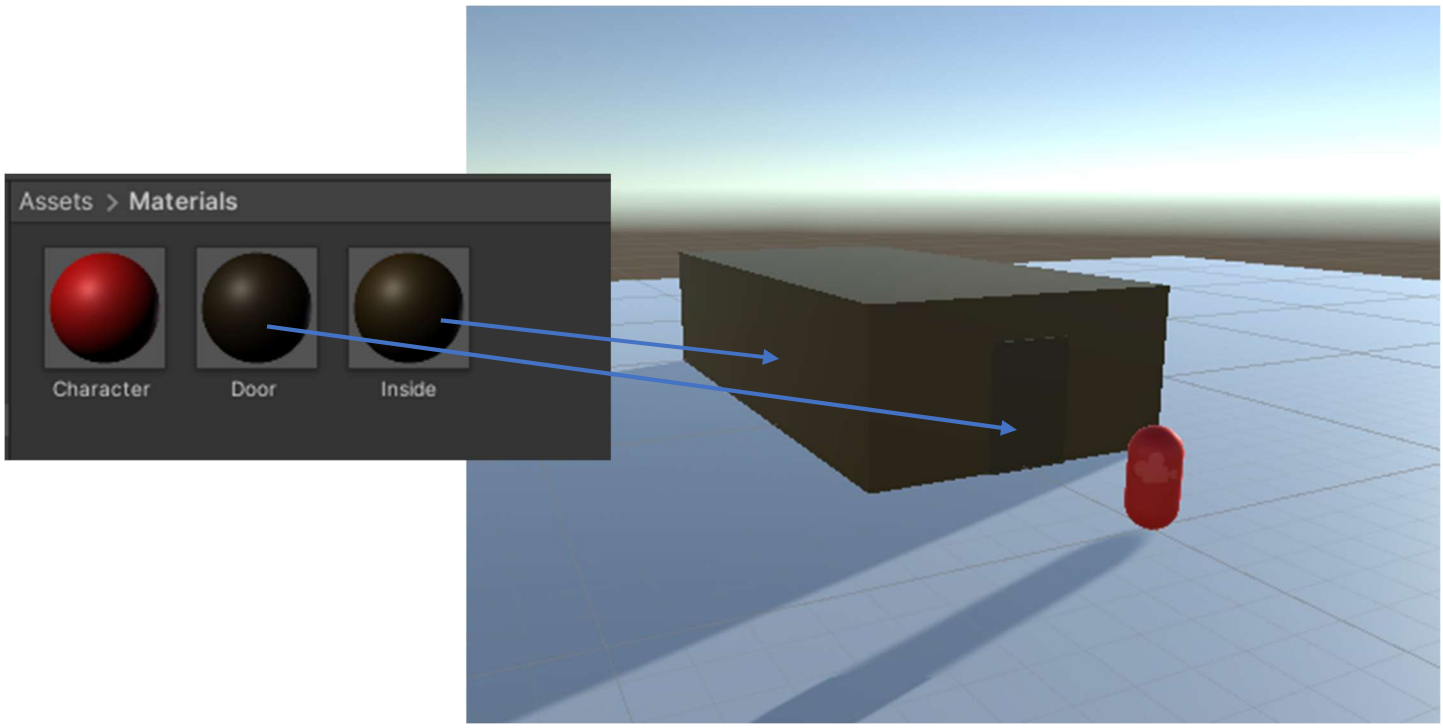


Here, I referenced the "DoorCheck" object in the "Character" object, where I linked the "DoorDetect" script to. I also had to select the "Door" layer as the "DoorMask".

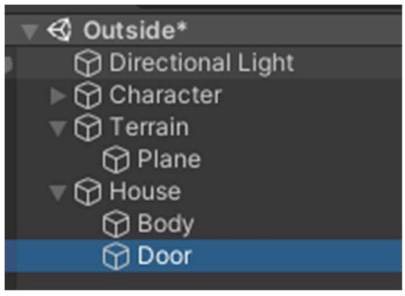


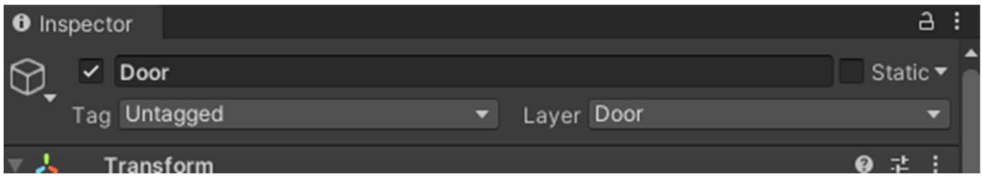
I deleted the test objects, being the “Slope” and “Step” objects and replaced them with two cubes that I resized to resemble a house and a door. I named them “Body” and “Door”, both under a parent object “House”.



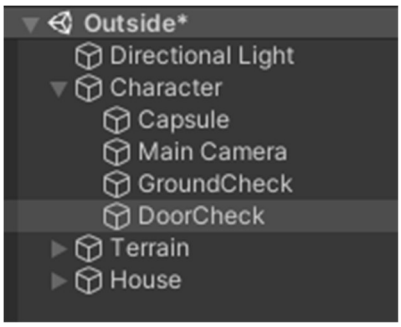


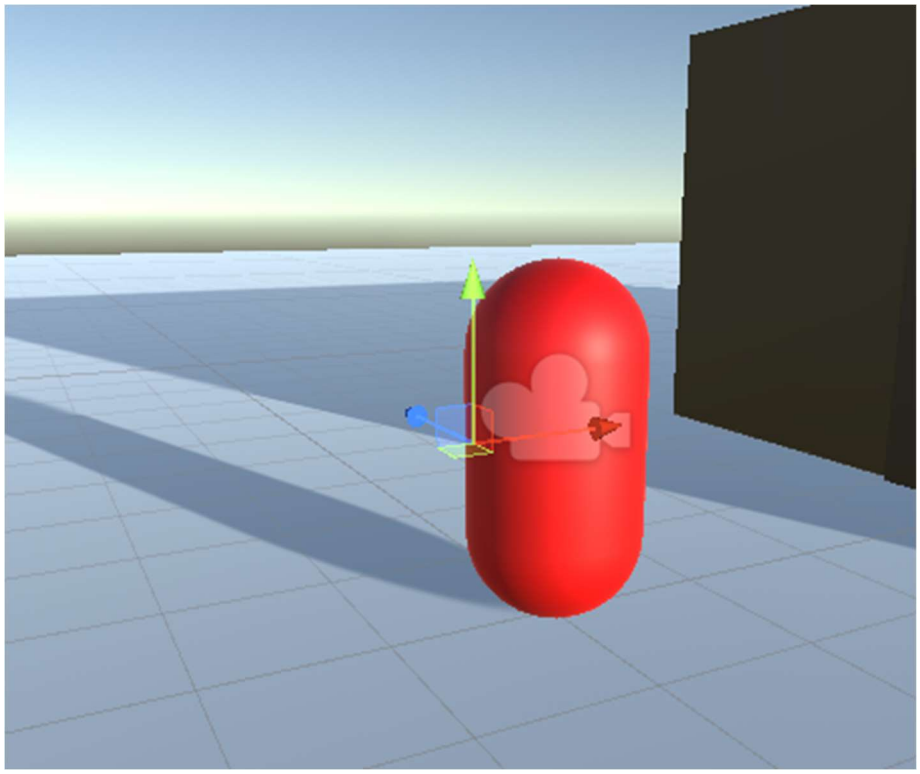
I used the “Inside” and “Door” materials to colour the house for now. Later in development I might find a wood log texture to put on the outside.



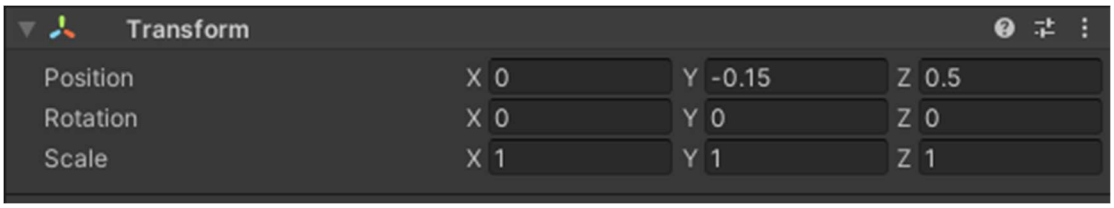


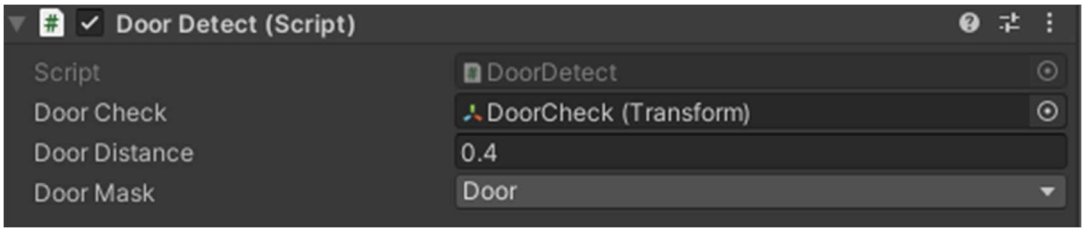
Here, I gave the new “Door” object the “Door” layer so that the “Character” object will be able to detect it as a door, and change scenes.





Here, I created the "DoorCheck" object, and positioned it correctly on the character. Below are the respective transform values.



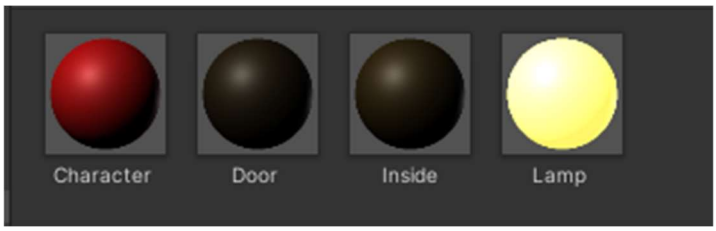


Here, I edited the "Character" object to update it with the new changes, adding the "DoorDetect" script and referencing the respective objects.

Date: 14/11/22

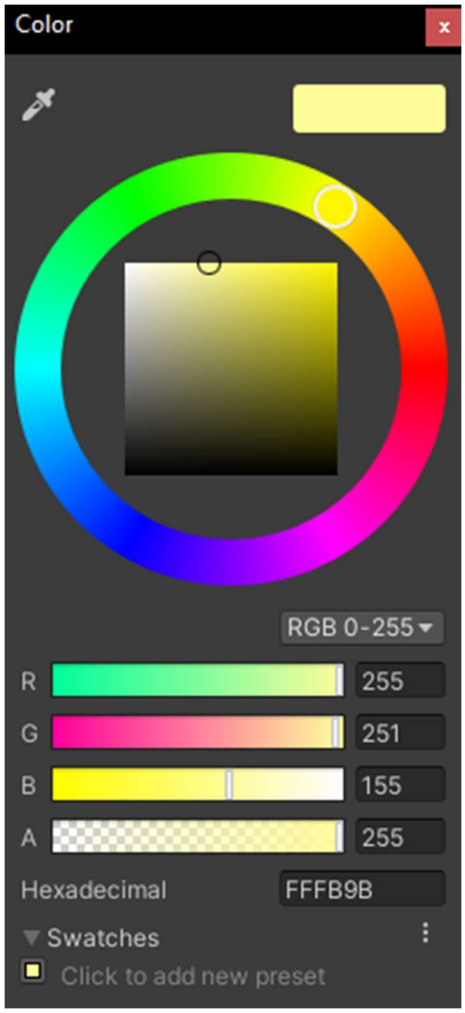
In this section of development, I wanted to fix a bug I ran into, whilst also improving the lighting and adding some objects into the “Inside” scene. When going from the “Inside” scene to the “Outside” scene, everything runs smoothly. However, when going the other way round, an error occurred since there is no lighting files for the “Inside” scene. I needed to bake the lighting and save them locally like I did for the “Outside” scene. Furthering this, I wanted to use a different type of lighting for the “Inside” scene, since “Directional” lighting doesn’t really work for a room. I decided to use “Point” lighting and created a couple lamps in the room. The first step I took here was to create two spheres and call them “Lamp1” and “Lamp2”. I scaled them down slightly and positioned them in such a way that only half of the sphere was visible. I then created a new material called “Lamp”, and then applied it to these objects. I checked the “Emission” box on this material, which gives the material a bright look. This made the lamps look much better. I then deleted the “Directional Light” object and added two “Point” light objects. I positioned them at the same positions as the lamps, and also set their lighting to “Realtime”. I was now happy with this lighting and had to bake it. I baked the “Outside” scene lighting again too since the objects have changed since I last baked it. This solution worked great, and now the player could seamlessly walk between scenes, and the lighting worked. At this point, I also wanted to add in some new objects to act as the planters. I created four new cube objects, then resized and positioned them. I was not at a point in development to turn them into plant storage units, so for now, I left them as placeholders. Test 1 from the design section should now be passed, and the process can be seen below as annotated screenshots.

Here, I created the “Lamp1” and “Lamp2” objects, then position and scaled them accordingly. The transform values can be seen below.

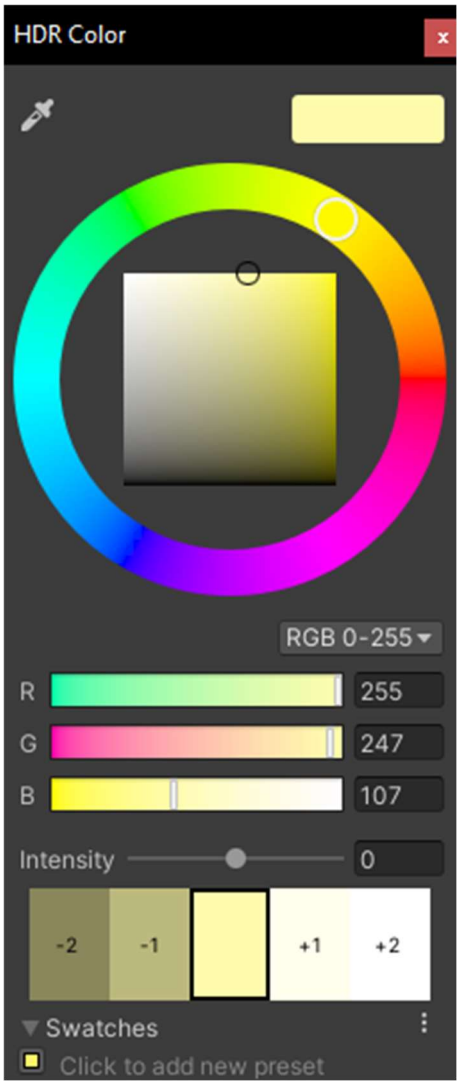


Here, I created the new material called “Lamp”.

Below is the “Albedo” colour settings.

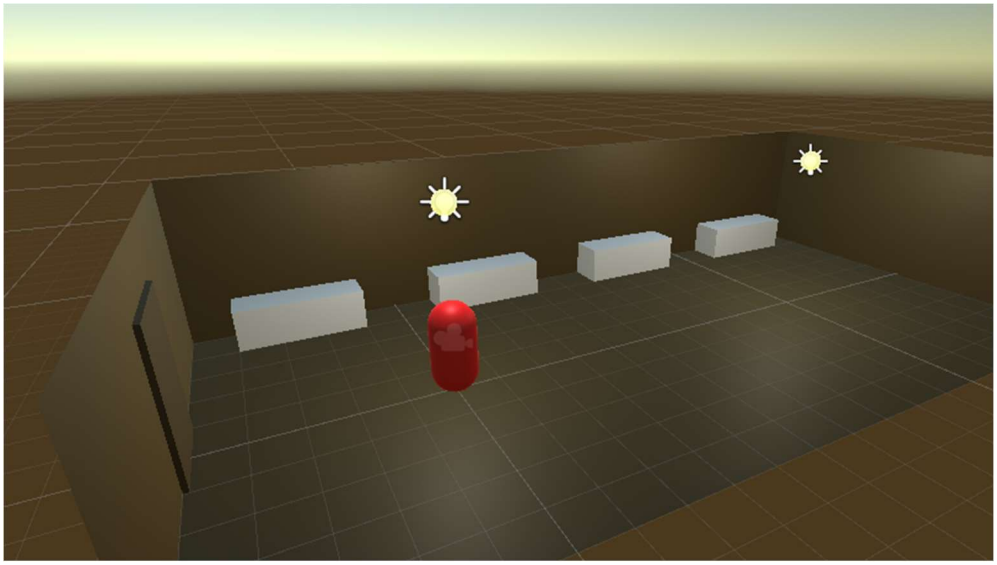


Below is the “HDR” colour settings.



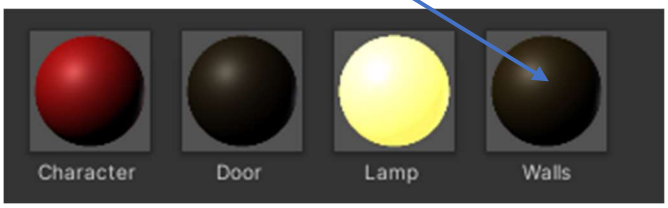
This is the end result of the lighting change.

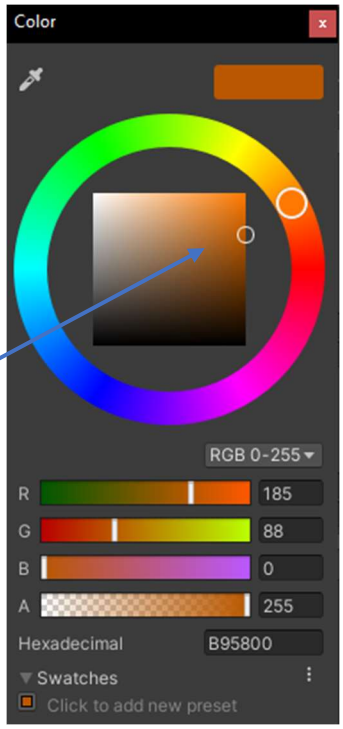
After baking the lighting for both the “Inside” and “Outside” scenes, I was left with two folders containing the lighting files.



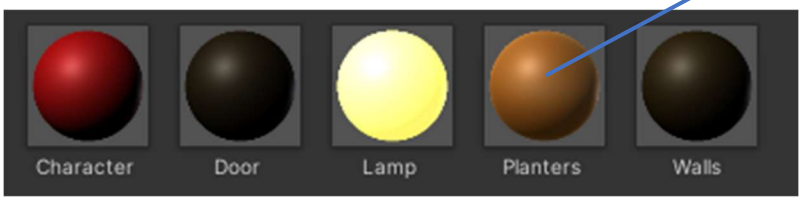
Added in four new cube objects, which I then reshaped and repositioned them. I named them “Planter1”, “Planter2”, “Planter3”, “Planter4”.

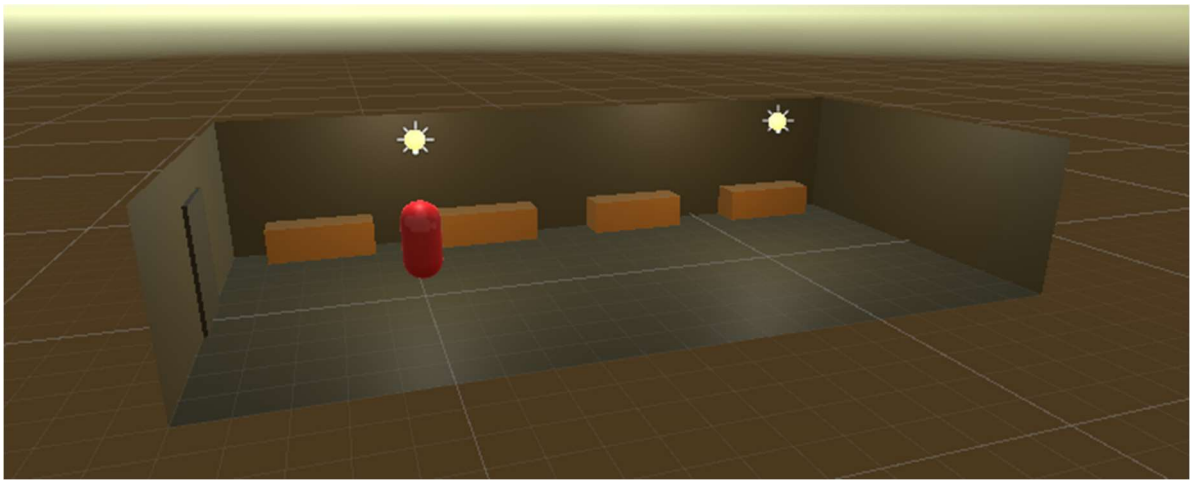
Changed “Inside” material to “Walls” material.





Added new “Planters” material.





Date: 26/11/22

I wanted to try another method to fix the broken sensitivity bar in the “Options” scene. I created a new script and called it “Global”. I may use this script in the future to instantiate any variables which I will use in multiple scenes, or global variables. I created the “sensitivity” variable and gave it a default value. I then edited the “MouseLook” script and removed the “mouseSensitivity” variable. I replaced it with a pointer to the “Global” script, using “Global.sensitivity”. Finally, I edited the “OptionsMenu” script, and ensured that the sensitivity slider edited the “sensitivity” variable in the “Global” script. This solution worked. The user could edit their mouse sensitivity in the options menu, and if left untouched, a default value would be used. I was helped out by a YouTube video[21] when tackling this bug. This process can be seen below.

Created “Global” script and instantiated the “sensitivity” float variable. I gave it the value of “1000” which will act as the default sensitivity value if the user does not change it in the “Options” scene.

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

// script to store the sensitivity variable

public class Global : MonoBehaviour
{
    public static float sensitivity = 1000f;
}
```

```
public class MouseLook : MonoBehaviour
{
    // create variable for mouse sensitivity
    public float mouseSensitivity = 1000f;
    // create transform for the player body
    public Transform playerBody;
    // create a variable for the x-rotation
    float xRotation = 0f;
```

Removed the instantiation of the “mouseSensitivity” variable within the “mouseLook” script, since it was no longer of use to me.

```
public class MouseLook : MonoBehaviour
{
    // create transform for the player body
    public Transform playerBody;
    // create a variable for the x-rotation
    float xRotation = 0f;
```

[21] (Zając 2020)

```
// update is called once per frame
void Update()
{
    // creates variables to store the mouse axis
    float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
    float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;
```

Removed the "mouseSensitivity" variable within the "mouseLook" script and replaced it with the "Global.sensitivity" reference. This points the script to the "Global" script and uses the "sensitivity" variable value.

```
// update is called once per frame
void Update()
{
    // creates variables to store the mouse axis
    float mouseX = Input.GetAxis("Mouse X") * Global.sensitivity * Time.deltaTime;
    float mouseY = Input.GetAxis("Mouse Y") * Global.sensitivity * Time.deltaTime;
```

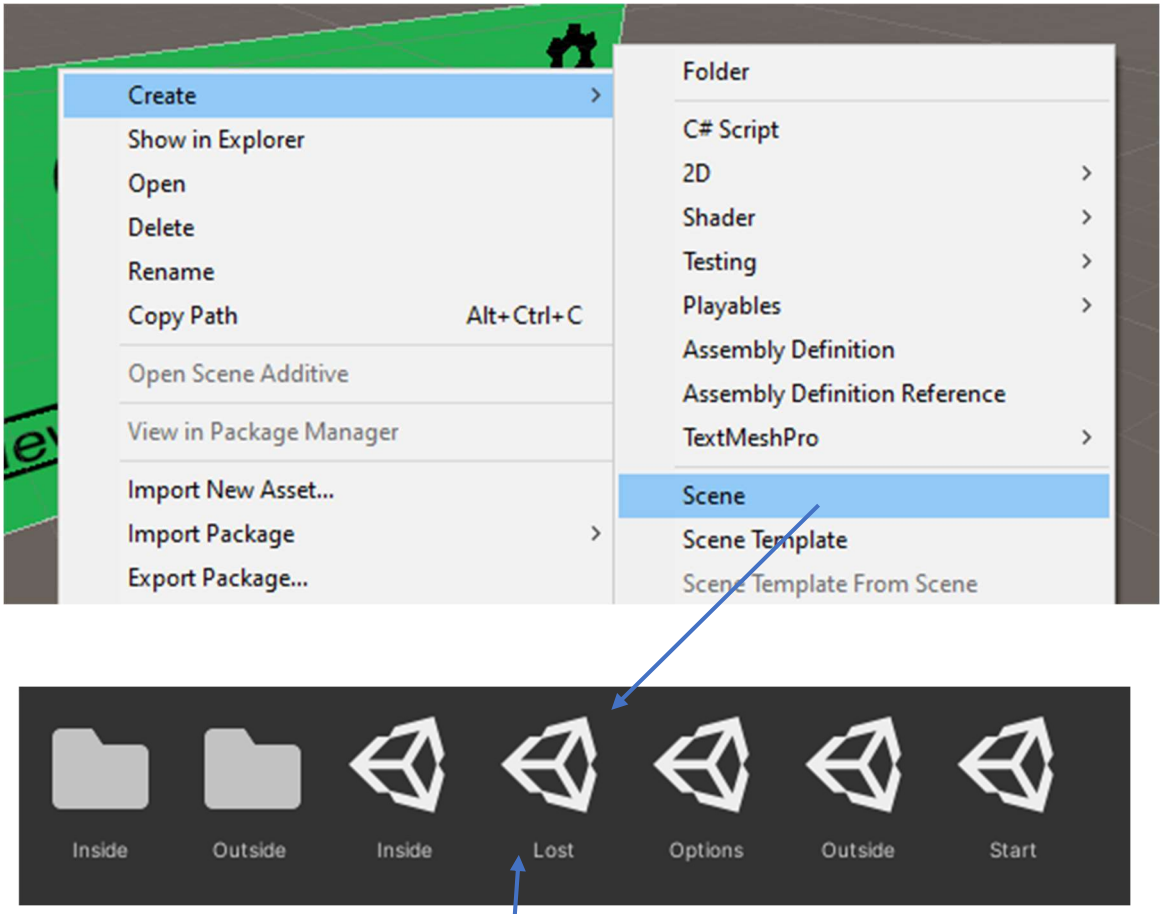
Ensures that the "sensitivity" variable within the "Global" script is updated with a new float value when the sensitivity slider is moved in the options menu.

```
public void SetSensitivity(float sensitivity)
{
    Global.sensitivity = sensitivity;
}
```

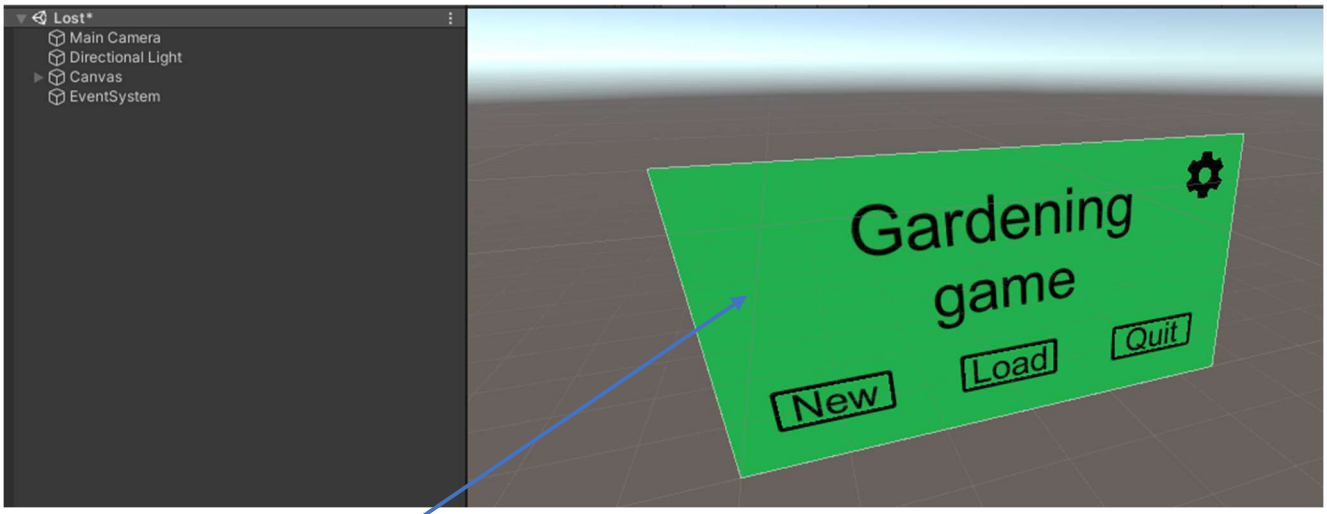
[3] Game over

Date: 27/11/22

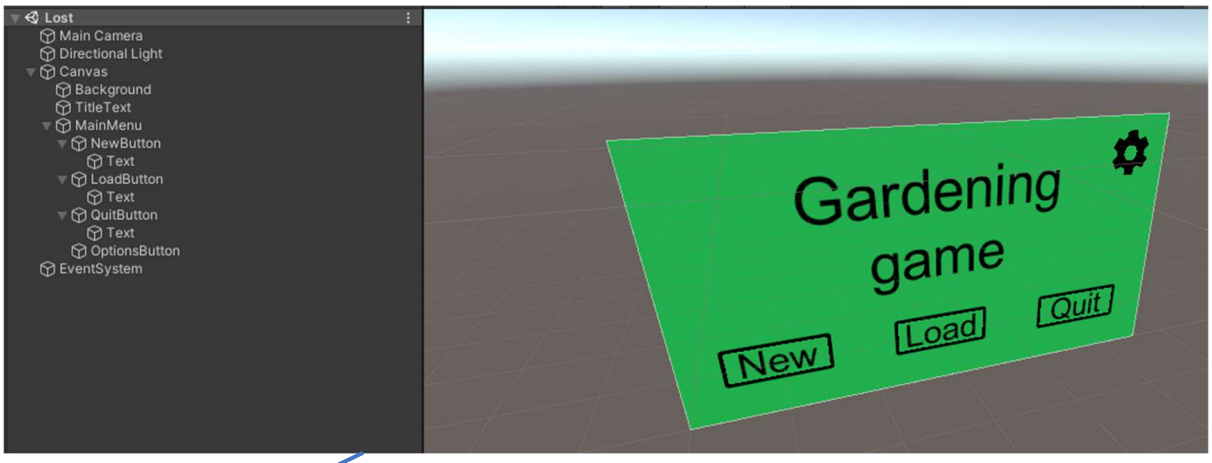
In this stage of development, I need to create a new scene called “Lost” and link up the buttons to their associated functions. The first thing to do was to create a new scene called “Lost”. I thought the best thing to do here was to copy over all the objects in the “Start” scene to this new “Lost” scene. This would save me a lot of time and allows me to reuse assets from earlier in the project. I chose the “Start” scene since it looks quite similar to my game over sketch up from the design section. All I had to do from here was change the text, remove two buttons and move the objects and their anchors around. From here, I could create a new script called “LostMenu”. This would handle the button presses made on this menu. For now, I will only add functionality to the “Quit” button and will handle the “Load” button once the save and load process has been developed. Before this however, I changed the “Quit” button to a “Return” button, since I thought the word “Quit” was misleading when all the button did was return the player to the start screen. Within the script I added a “Return” function, which placed the user back into the “Start” scene. This marked this development subsection as complete. There was now a game over screen, from where the user can return to the main menu. This process can be seen below in the form of annotated screenshots.



Created a new scene and renamed it “Lost”.



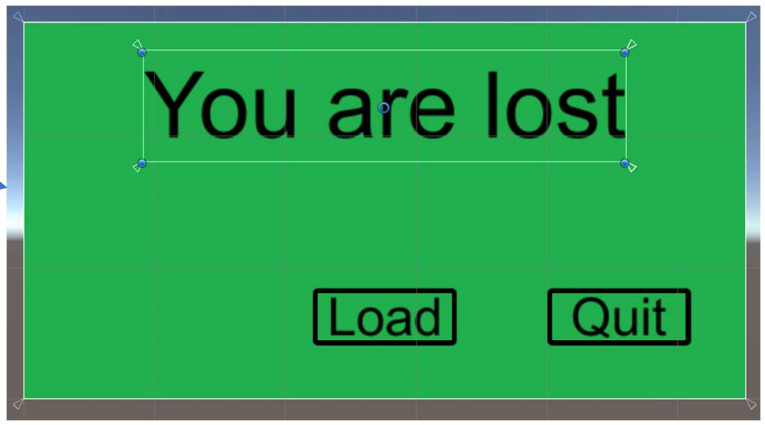
Copied over all the objects from the “Start” scene.



Deleted the “NewButton” object.

Deleted the “OptionsButton” object.

Edited the text attribute within the “TitleText” object from “Gardening game” to “You are lost”.



Renamed the “MainMenu” object to “LostMenu”.

Moved the buttons around to fit the screen properly, adjusted their anchors also.

Renamed the "Quit" button to a "Return" button.

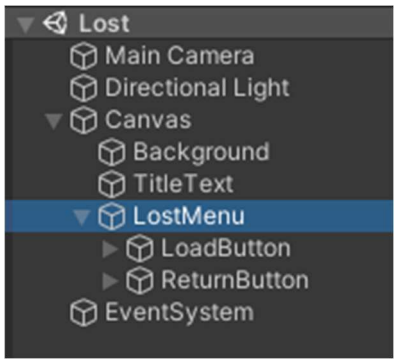
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LostMenu : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
}
```

Removed the "Start" and "Update" functions since they are not required.

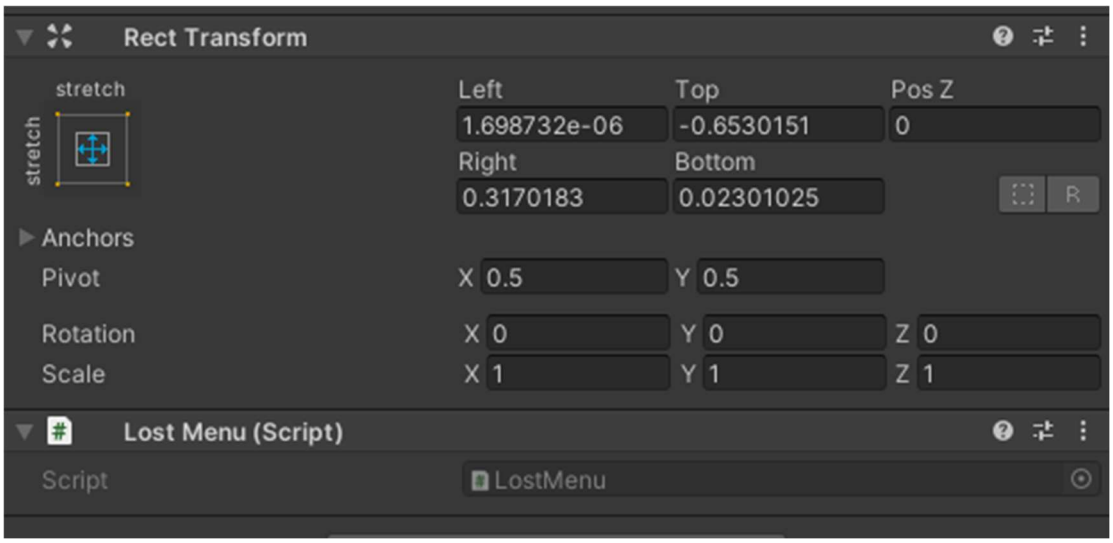
Had to include the "SceneManagement" since the script changes the scene.

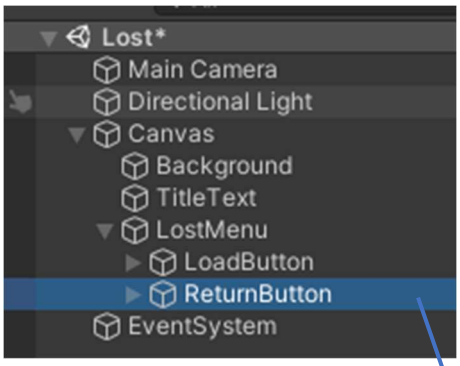
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LostMenu : MonoBehaviour
{
    public void Return()
    {
        SceneManager.LoadScene("Start");
    }
}
```

Changes the scene to "Start" if the "Return" button is pressed.

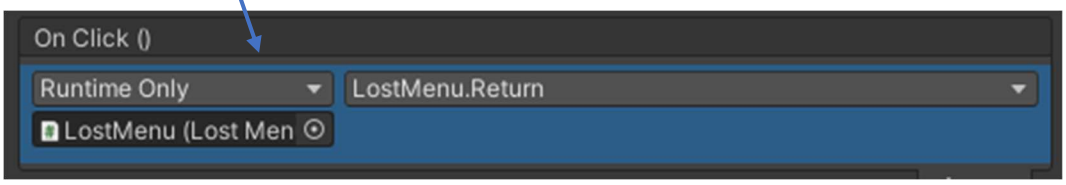


Added the “LostMenu” script to the “LostMenu” game object.





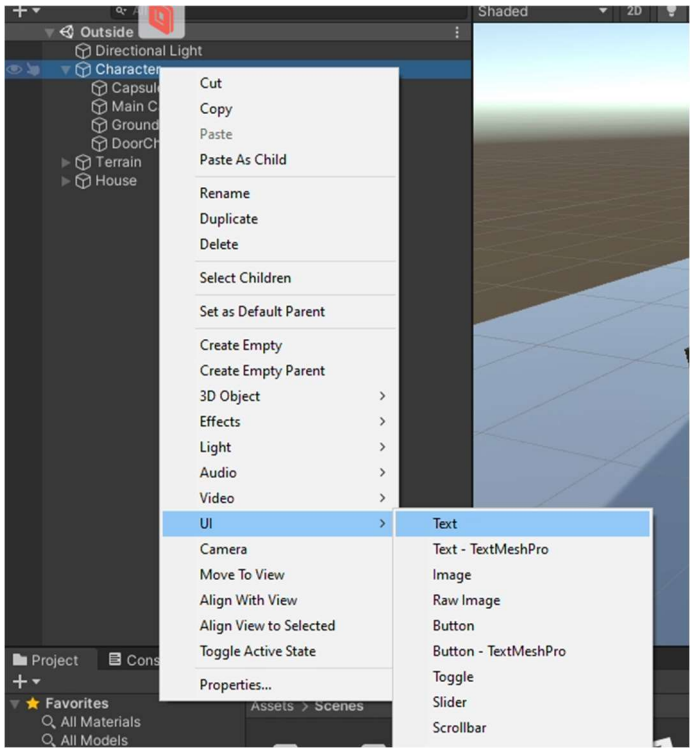
Referenced the “Return” function within the “ReturnButton” object.



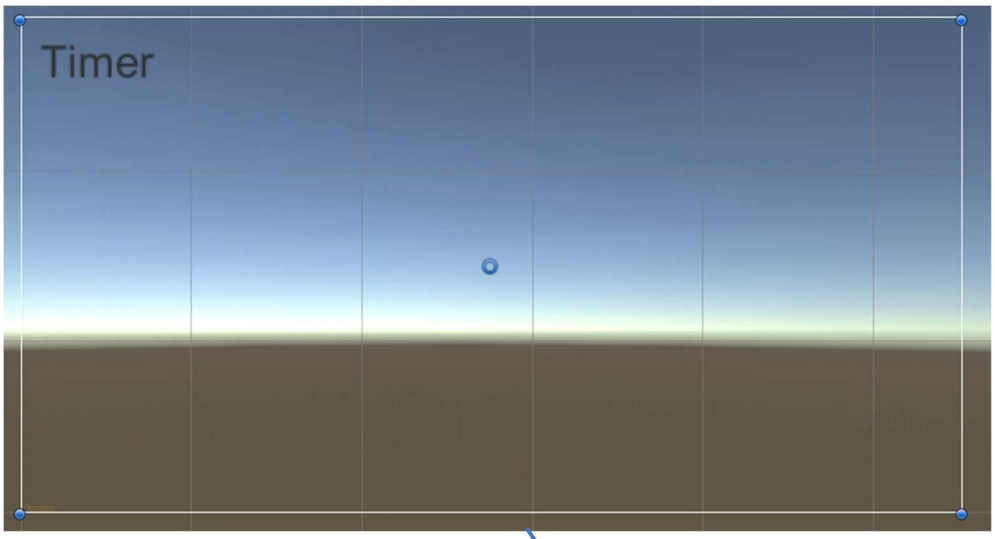
[3] Explore mechanic

Date: 28/11/22

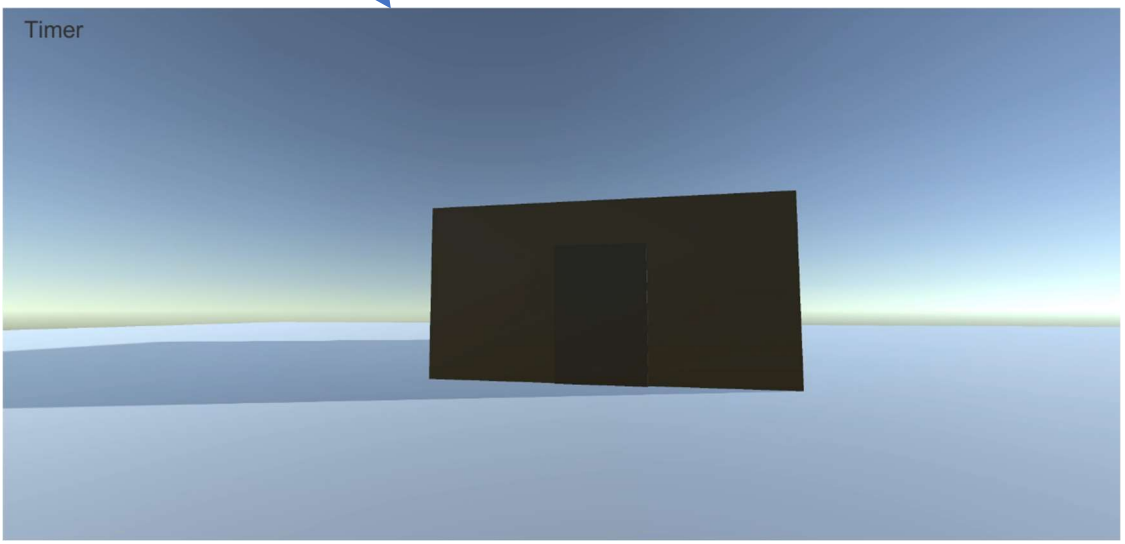
In this subsection, I wanted to add a timer into the player UI when they are exploring. To start, I created a new “Text” object within the UI section to the “Character” object, in the “Outside” scene. I renamed the “Canvas” object to “UI”, since it will act as the player UI when they are exploring. I positioned the “Text” object within the “UI” object to the top left of the screen, changed the text size so it was a bit larger, and changed the text attribute from “New Text” to “Timer”, for now. I also had to move the anchors, so they matched the size of the text box. I checked “Best fit” so that the text size would scale up with resolution. I also created a new object called “Timer” within the “UI” object, to make it easier for myself to manage in the future. I then had to redo the anchors. From here, I could create a new script, which I decided to name “Timer”. I used my design section pseudocode here, which I found to be helpful. I then attached this script to the “Timer” object. I referenced the “timerText” variable to the “Text” object within the “Timer” object. Once this was done, the solution worked great, and the player was moved to the “Lost” scene after the time ran out. Walking in the house, and back out, reset the timer. Test 5, 6 and 8 from my design section can now be passed. This process can be seen below as annotated screenshots.



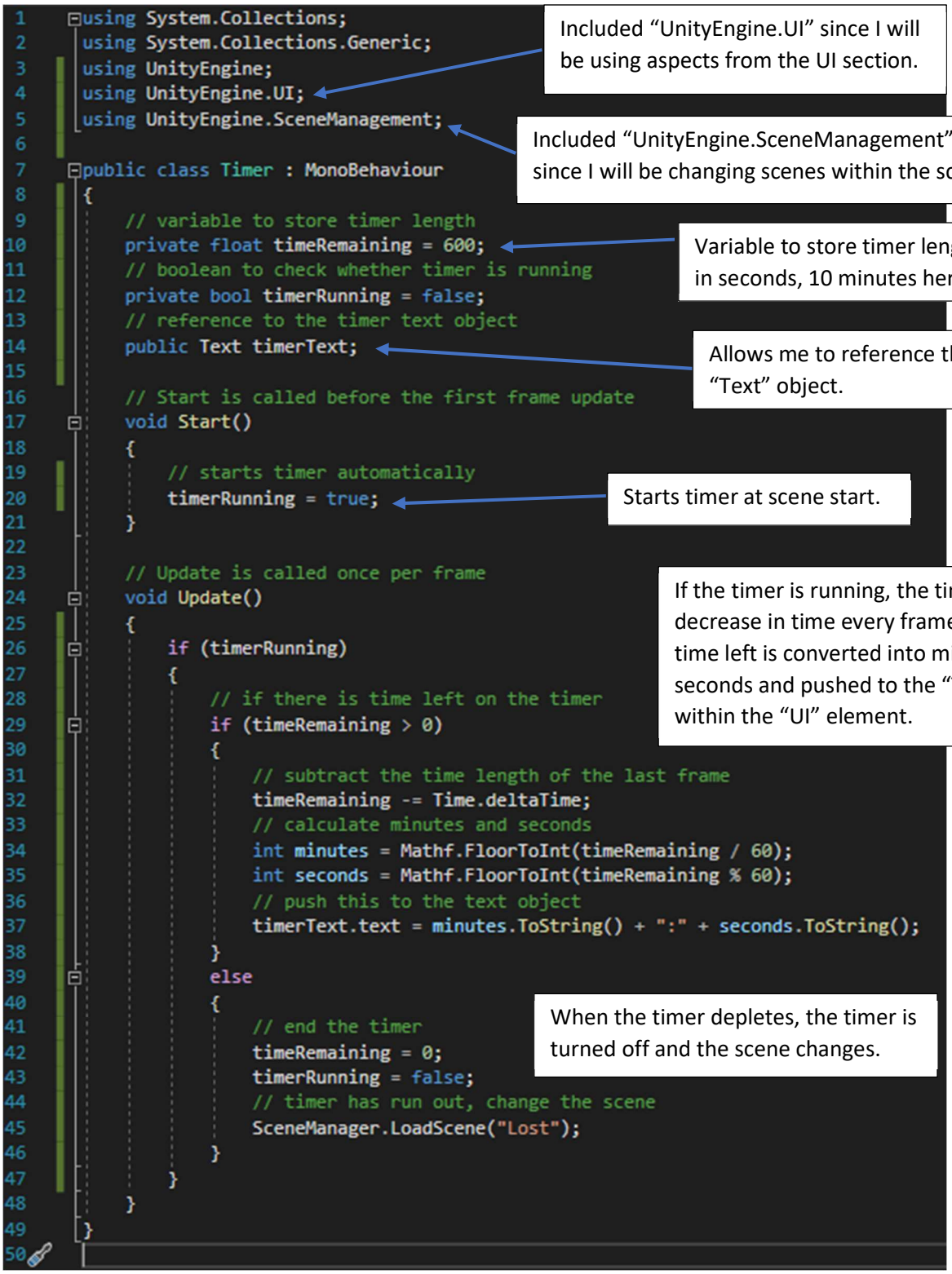
Created a new “Text” object from the UI section.



New “Text” UI object after being edited and placed onto the character object.



Added a new “Timer” parent object for the “Text” object.



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

public class Timer : MonoBehaviour
{
    // variable to store timer length
    private float timeRemaining = 600;
    // boolean to check whether timer is running
    private bool timerRunning = false;
    // reference to the timer text object
    public Text timerText;

    // Start is called before the first frame update
    void Start()
    {
        // starts timer automatically
        timerRunning = true;
    }

    // Update is called once per frame
    void Update()
    {
        if (timerRunning)
        {
            // if there is time left on the timer
            if (timeRemaining > 0)
            {
                // subtract the time length of the last frame
                timeRemaining -= Time.deltaTime;
                // calculate minutes and seconds
                int minutes = Mathf.FloorToInt(timeRemaining / 60);
                int seconds = Mathf.FloorToInt(timeRemaining % 60);
                // push this to the text object
                timerText.text = minutes.ToString() + ":" + seconds.ToString();
            }
            else
            {
                // end the timer
                timeRemaining = 0;
                timerRunning = false;
                // timer has run out, change the scene
                SceneManager.LoadScene("Lost");
            }
        }
    }
}
```

Included “UnityEngine.UI” since I will be using aspects from the UI section.

Included “UnityEngine.SceneManagement” since I will be changing scenes within the script.

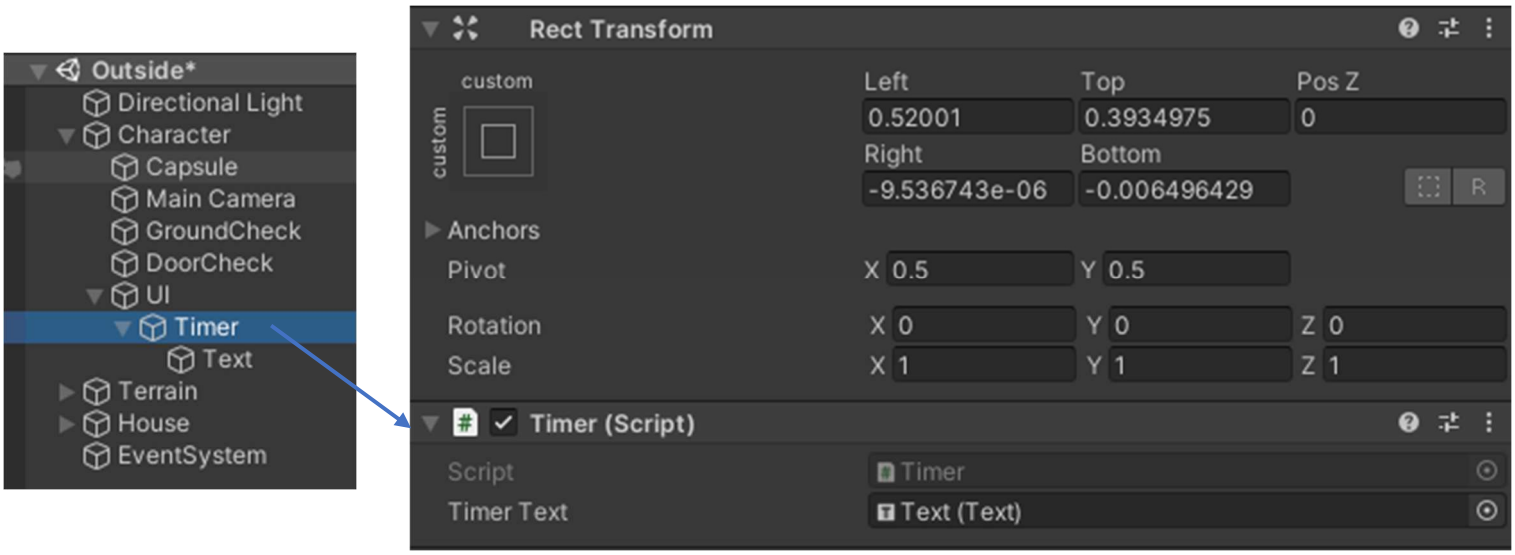
Variable to store timer length in seconds, 10 minutes here.

Allows me to reference the “Text” object.

Starts timer at scene start.

If the timer is running, the timer will decrease in time every frame. The current time left is converted into minutes and seconds and pushed to the “Text” object within the “UI” element.

When the timer depletes, the timer is turned off and the scene changes.



Added the “Timer” script as a component of the “Timer” object.

Date: 28/11/22

When testing this implementation of the timer, I ran into an issue that I had not considered before. Once the timer ends, the player is moved to the “Lost” screen. However, they cannot interact with the menu. I think this is because in the “Outside” scene, the cursor is locked. At no point is it unlocked. So, to fix this, I will add “Cursor.lockState = CursorLockMode.None;” within the “Start” function of all the menu scenes; “Start”, “Options”, “Lost”. This solved the problem, and the process can be seen below.

```
public class MainMenu : MonoBehaviour
{
    // function that runs at beginning
    void Start()
    {
        // ensures that the cursor is unlocked
        Cursor.lockState = CursorLockMode.None;
    }
```

Ensured the cursor was unlocked in the “Start” function of the “MainMenu” script.

```
public class LostMenu : MonoBehaviour
{
    // function that runs at beginning
    void Start()
    {
        // ensures that the cursor is unlocked
        Cursor.lockState = CursorLockMode.None;
    }
```

Ensured the cursor was unlocked in the “Start” function of the “LostMenu” script.

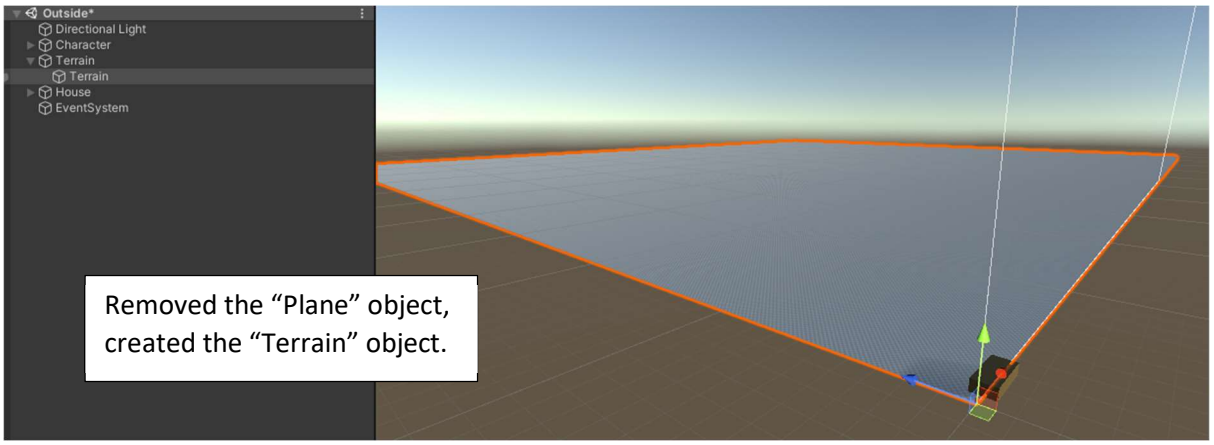
```
// function that runs at beginning
void Start()
{
    // fill array with resolutions
    resolutions = Screen.resolutions;
    // remove any options already in dropdown
    resolutionDropdown.ClearOptions();
    // list to store string versions of resolutions
    List<string> options = new List<string>();
```

Added a line of code to ensure that the cursor is not locked when the scene is loaded.

```
// function that runs at beginning
void Start()
{
    // ensures that the cursor is unlocked
    Cursor.lockState = CursorLockMode.None;

    // fill array with resolutions
    resolutions = Screen.resolutions;
    // remove any options already in dropdown
    resolutionDropdown.ClearOptions();
    // list to store string versions of resolutions
    List<string> options = new List<string>();
```

Date: 15/12/22

At this point in development, I wanted to tackle one of the most complex problems in this project, being the terrain generation with Perlin noise. I decided to use a Unity “Terrain” object, instead of using a mesh to create the terrain. This was due to two reasons: a terrain object already has a collider, and Unity allows for automatic tree distribution across a terrain object. This simplifies the problem for me and is why I chose it. So, the first step for me here was to remove the “Plane” object that I have had as a placeholder for a while, and replace it with a new “Terrain” object. I then had to give it the “Terrain” layer, so the character controller would work well on it. Now, I created a new script, and called it “TerrainGeneration”. I could remove the default “Update()” function, since the terrain will only change at the start of the script, and will not be updated during. I had to set up some variables, such as the “height”, “width” and “depth” of the terrain. I also had to set up special variables relating to Perlin noise; the scale of the terrain, number of octaves, the lacunarity and the persistence. For now, I chose basic values which I can fine tune later. I also had to define offsets for the x and z axis, and giving these random values will give me random terrains. After this, I was able to begin on the “Start()” function. Here, I had to give random values to the offset variables, reference the “Terrain” object, then call the function that will generate the new terrain. Next, I had to create a function that could calculate a height when given an x and z coordinate. I called it “CalculateHeight”. With this done, I could create a function called “GenerateHeights”, which returns a ”heightmap”, an array containing all the heights for every coordinate on the terrain. This will utilise the ”CalculateHeight” function. Now, I can create a function called “GenerateTerrain”, which will push the heightmap onto the terrain object. With the script complete, I could add the script as a component to the “Terrain” object. Now, every time I ran the script, a new terrain was generated. There was a border around the terrain, so I had to ensure that the size of the terrain was a power of 2. This fixed this issue. This whole process can be seen below in the form of annotated screenshots.



Removed the “Plane” object, created the “Terrain” object.

```
public class TerrainGeneration : MonoBehaviour
{
    // declare terrain size variables
    public int xSize = 1000; // x axis
    public int zSize = 1000; // z axis
    public int ySize = 200; // y axis

    // declare perlin noise variables
    public float scale = 4f;
    public float offsetX; // random value
    public float offsetZ; // random value
    public int octaves = 6;
    public float lacunarity = 2f; // frequency multiplier
    public float persistence = 0.5f; // amplitude multiplier
```

Here, I set up the size of the terrain to be generated.

At this point, I set up the Perlin noise variables.

This is how much the amplitude is multiplied by each octave.

This is how much the frequency is multiplied by each octave.

For now, I have set them as “public” variables. This is so I can fine tune the terrain generation if needs be. They will be set to private once that is done.

```
// Start is called before the first frame update
void Start()
{
    // calculate random offsets
    offsetX = Random.Range(0f, 9999f);
    offsetZ = Random.Range(0f, 9999f);

    // reference terrain component to change data
    Terrain terrain = GetComponent<Terrain>();
    // create new terrain based off of current terrain
    terrain.terrainData = GenerateTerrain(terrain.terrainData);
}
```

Here, I give a random float value between 0 and 9999 to the offset variables. This ensures that it is a random terrain each time the scene is loaded.

Here, I reference the "Terrain" object, so that I can edit it within the script. I assign it to a variable called "terrain".

At this point, I set the "terrainData" to the output of the "GenerateTerrain" function. This will be developed soon.

All of this is within the "Start" function since I need this to run when the player leaves the house. Then, a random landscape is generated every time they explore.

Input of an "x" and "z" value.

```
float CalculateHeight(int x, int z)
{
    float xCoord = (float)x / xSize * scale + xOffset;
    float zCoord = (float)z / zSize * scale + zOffset;
    float frequency = 1f;
    float amplitude = 0.5f;
    float y = 0f;
    for (int i = 0; i < octaves; i++)
    {
        y = y + (amplitude * Mathf.PerlinNoise(xCoord * frequency, zCoord * frequency));
        frequency = frequency * lacunarity;
        amplitude = amplitude * persistence;
    }
    return y;
}
```

Here, I need to convert these "x" and "z" numbers to coordinates on the terrain. I divide them by the size of the terrain, multiply them by the scale, and then add the offset.

Here, I set the initial frequency and amplitude values.

Finally, I create a for loop which is conditional on the number of octaves to generate a "y" value for this coordinate on the terrain. I use the "Mathf.PerlinNoise" function and apply the current amplitude and frequency values. Then, I generate new frequency and amplitude variables by multiplying them by lacunarity and persistence respectively. Then, I can return the "y" value after all the octaves have been completed.

Will return an array of floats.

New two-dimensional array of the same size as the terrain.

```
float[,] GenerateHeights()
{
    // create a grid of floats for heights of terrain
    float[,] heights = new float[xSize, zSize];
    // use perlin noise
    for (int x = 0; x < xSize; x++)
    {
        for (int z = 0; z < zSize; z++)
        {
            heights[x, z] = CalculateHeight(x, z);
        }
    }
    // return the heights array
    return heights;
}
```

Here, I run through every item in the array, assigning a height, produced by the "CalculateHeight" function.

Finally, I can return the "heights" array. This will be used as a heightmap for the terrain.

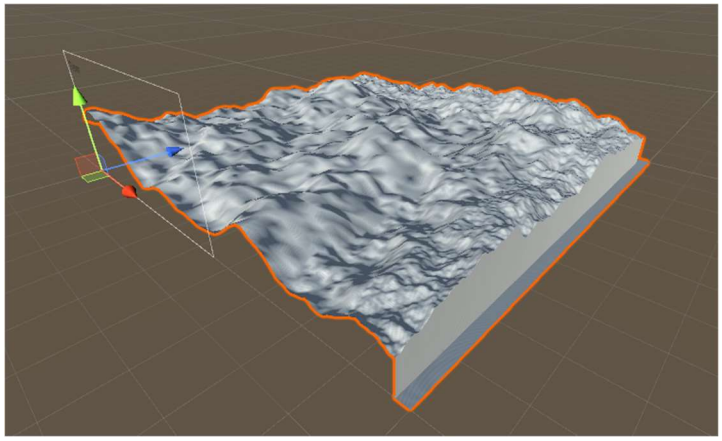
Has an input of the initial terrain's "terrainData".

```
TerrainData GenerateTerrain(TerrainData terrainData)
{
    terrainData.heightmapResolution = xSize + 1;
    // set up size of terrain
    terrainData.size = new Vector3(xSize, ySize, zSize);
    // modify heights of floats in terrain
    terrainData.SetHeights(0, 0, GenerateHeights()); // from start point 0, 0
    return terrainData;
}
```

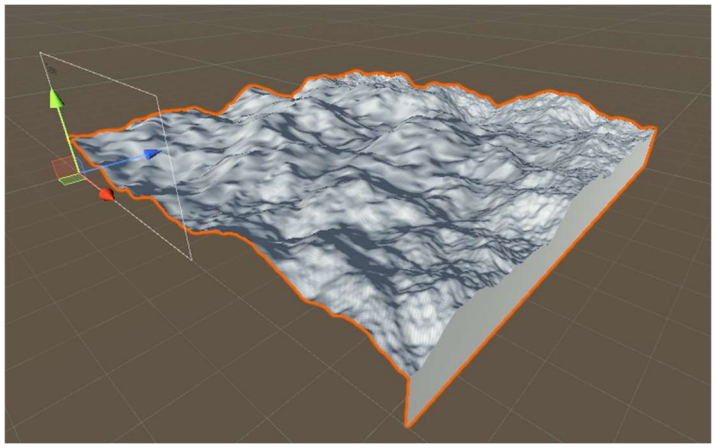
Sets the amount of "texels" that the terrain takes up. I add 1 to ensure that there are enough.

Finally, I set the terrain's size using the variables set up earlier and can push the heights to the terrainData using the "SetHeights" method, with an input of my heightmap. I then return the new "terrainData".

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

public class TerrainGeneration : MonoBehaviour
{
    // declare terrain size variables
    public int xSize = 1000; // x axis
    public int zSize = 1000; // z axis
    public int ySize = 200; // y axis

    // declare perlin noise variables
    public float scale = 4f;
    public float xOffset; // random value
    public float zOffset; // random value
    public int octaves = 6;
    public float lacunarity = 2f; // frequency multiplier
    public float persistence = 0.5f; // amplitude multiplier

    // Start is called before the first frame update
    void Start()
    {
        // calculate random offsets
        offsetX = Random.Range(0f, 9999f);
        offsetZ = Random.Range(0f, 9999f);

        // reference terrain component to change data
        Terrain terrain = GetComponent<Terrain>();
        // create new terrain based off of current terrain
        terrain.terrainData = GenerateTerrain(terrain.terrainData);
    }

    float CalculateHeight(int x, int z)
    {
        float xCoord = (float)x / xSize * scale + xOffset;
        float zCoord = (float)z / zSize * scale + zOffset;
        float frequency = 1f;
        float amplitude = 0.5f;
        float y = 0f;
        for (int i = 0; i < octaves; i++)
        {
            y = y + (amplitude * Mathf.PerlinNoise(xCoord * frequency, zCoord * frequency));
            frequency = frequency * lacunarity;
            amplitude = amplitude * persistence;
        }
        return y;
    }

    float[,] GenerateHeights()
    {
        // create a grid of floats for heights of terrain
        float[,] heights = new float[xSize, zSize];
        // use perlin noise
        for (int x = 0; x < xSize; x++)
        {
            for (int z = 0; z < zSize; z++)
            {
                heights[x, z] = CalculateHeight(x, z);
            }
        }
        // return the heights array
        return heights;
    }

    TerrainData GenerateTerrain(TerrainData terrainData)
    {
        terrainData.heightmapResolution = xSize + 1;
        // set up size of terrain
        terrainData.size = new Vector3(xSize, ySize, zSize);
        // modify heights of floats in terrain
        terrainData.SetHeights(0, 0, GenerateHeights()); // from start point 0, 0
        return terrainData;
    }
}
```



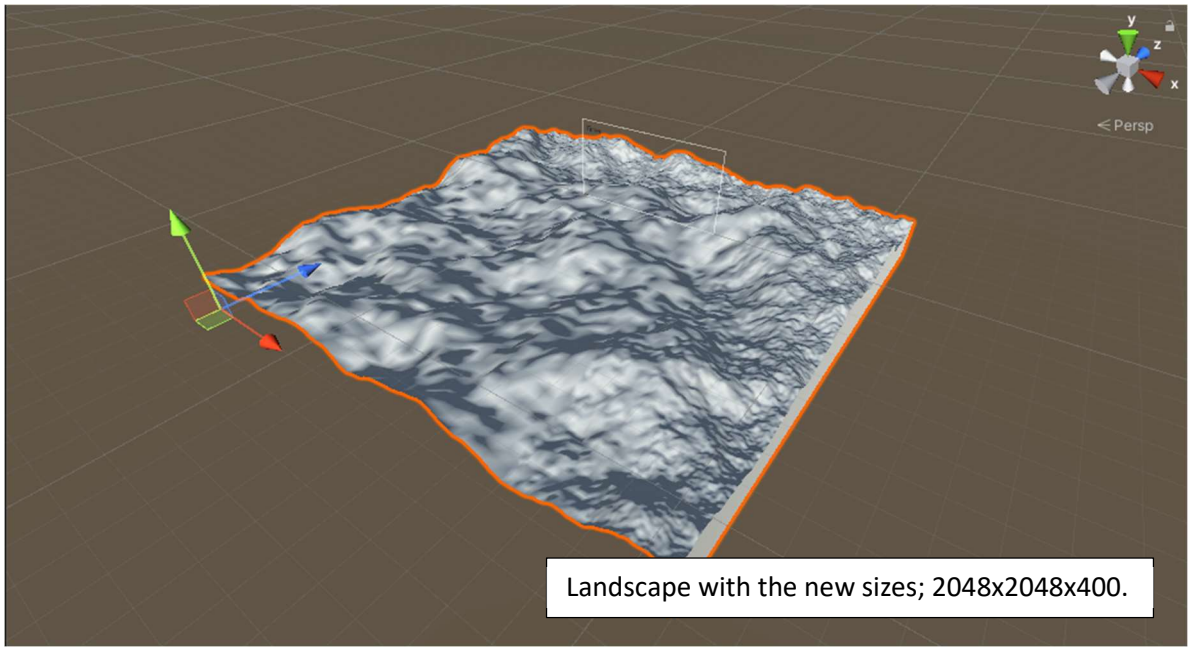
```
// declare terrain size variables
public int xSize = 1000; // x axis
public int zSize = 1000; // z axis
public int ySize = 200; // y axis
```



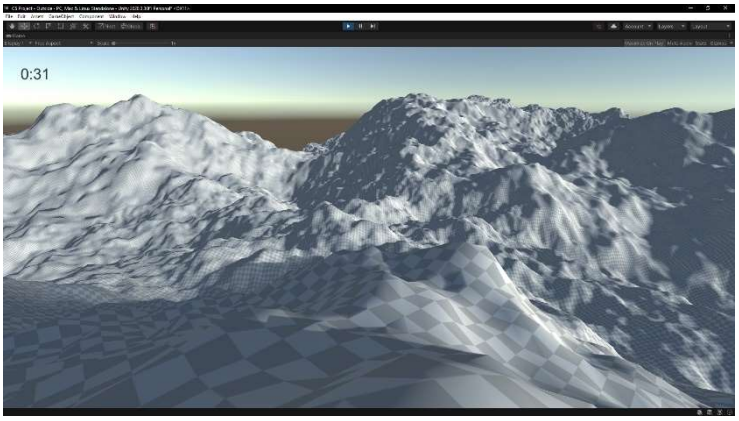
```
// declare terrain size variables
public int xSize = 1024; // x axis
public int zSize = 1024; // z axis
public int ySize = 200; // y axis
```

Date: 16/12/22

In this section, I wanted to take some time getting the size and variables right for the landscape. The first thing I did was increase the “xSize” from 1024 to 2048, “zSize” from 1024 to 2048 and “ySize” from 200 to 400. This increase in size will allow me to apply gradients to the landscape, but also have a large area for the player to explore. I have to be careful when increasing sizes, since it will increase the loading times between scenes. I thought that the “persistence” and “lacunarity” values I had initially picked were working great, so I did not change them. However, I wanted to find a good number of octaves to use. This will also increase loading times if I pick a high amount. I tested using higher octaves, which produced very detailed landscapes but made the character controller feel jumpy due to the jagged edges. Lower octaves produced landscapes that lacked details. In the end, I chose to stick with 6 octaves, since the landscape was not jagged, it produced interesting features, and had reasonable loading times. Now, I wanted to give the landscape a basic green colour. I created a new material called “Landscape” and gave it a green albedo. At this point, I was happy with how the landscape looked. This process can be seen below.

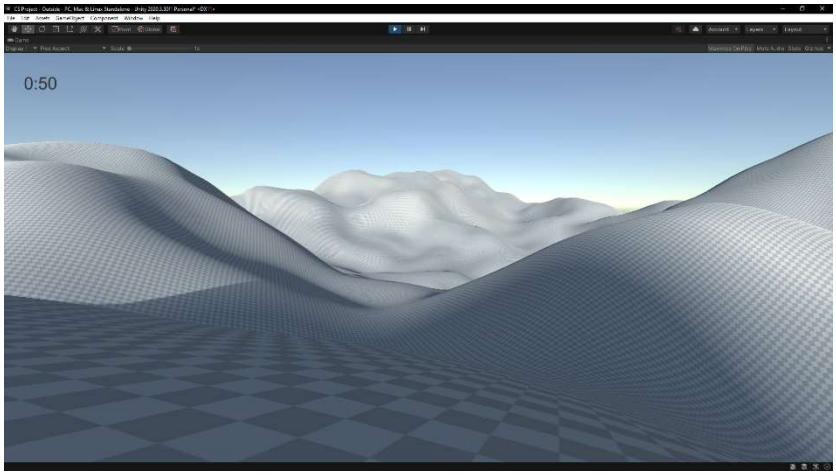


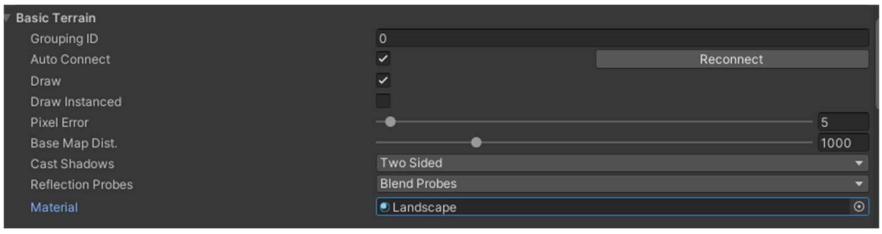
Landscape with the new sizes; 2048x2048x400.



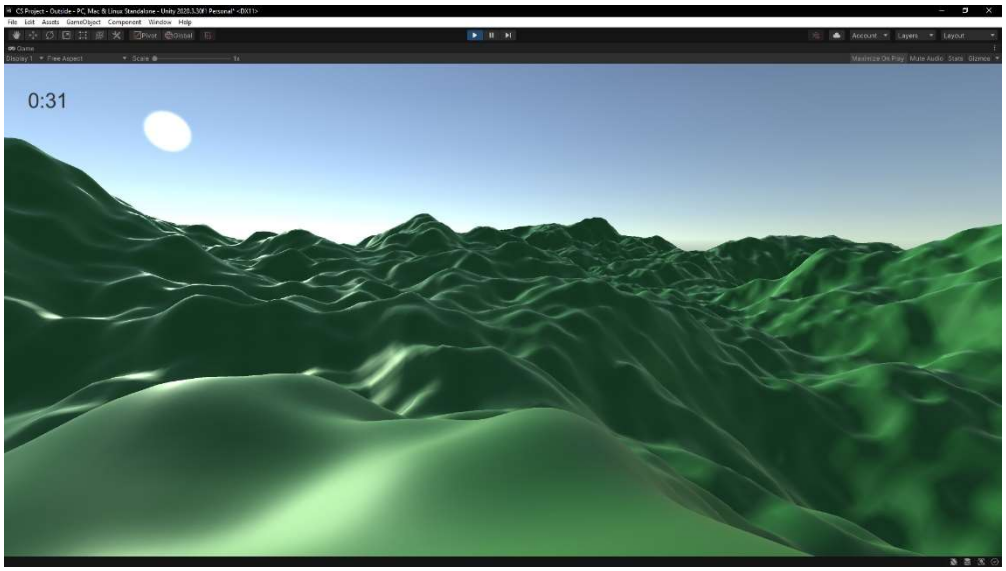
Here, I used 18 octaves. The landscape is very detailed, but quite jagged. This made the character controller feel less smooth.

Here, I used 4 octaves. The landscape is very smooth but is quite boring.



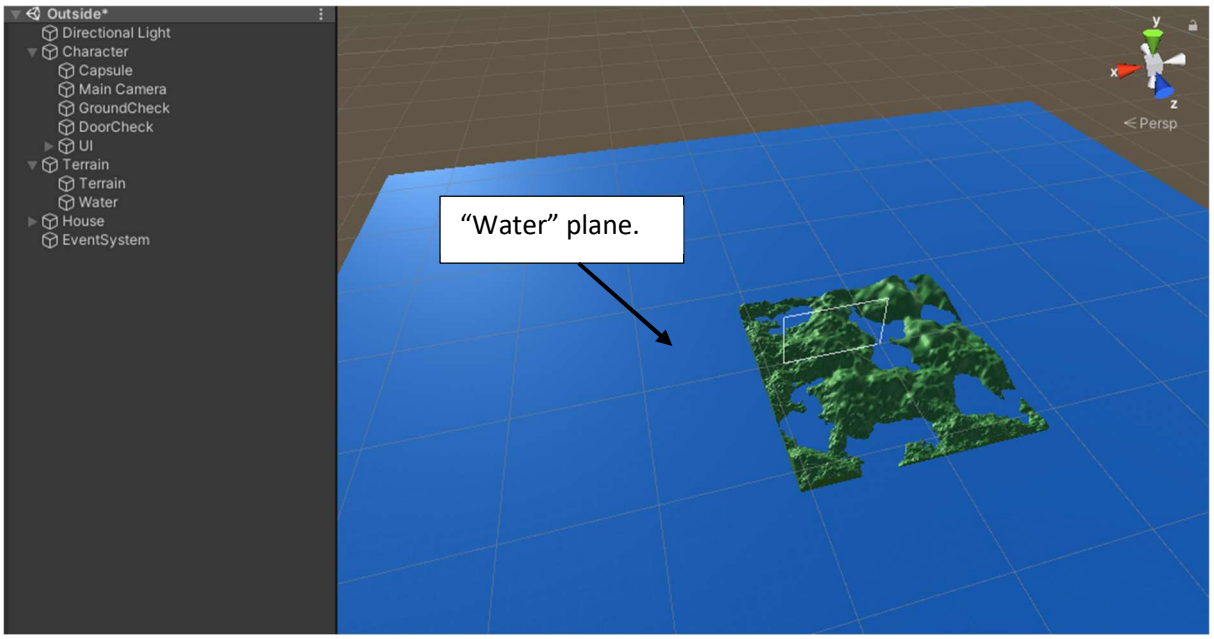


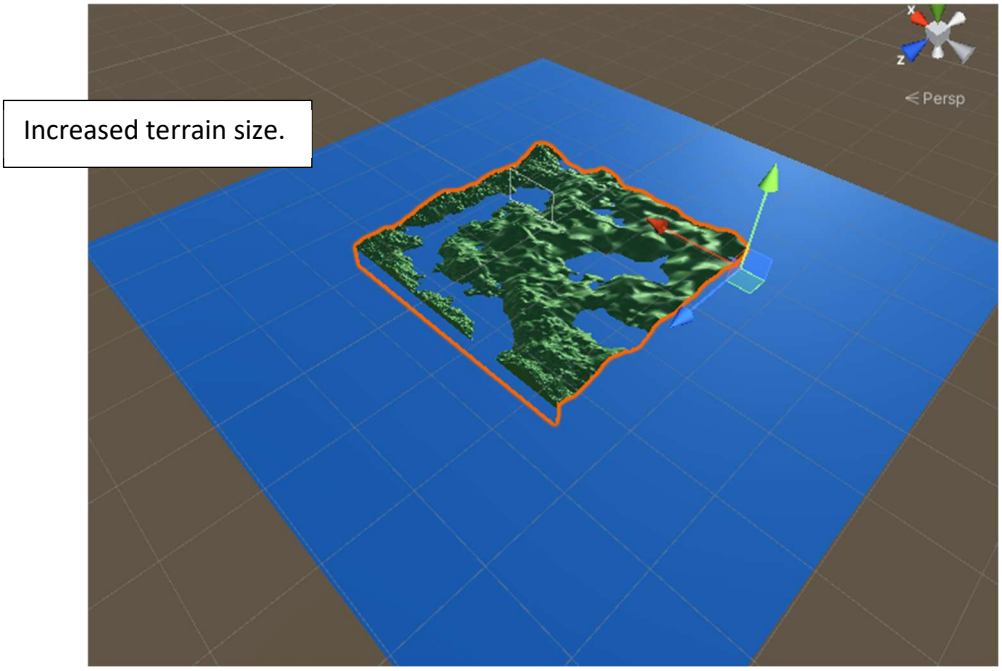
Here, I apply the "Landscape" material to the terrain. For now, it is just a basic green.



Date: 16/12/22

Here, I quickly wanted to add some water into the landscape. I created a new plane, called “Water”. I then gave it a blue colour, with the “Water” material that I made. The “Water” plane does not have the “Terrain” layer, so the player cannot jump when standing on the water. I think this helps the player differentiate between standing on water and land. In the future I would like to decrease movement speed when standing on water. At this point, I realised that the landscape was going to be once again too small, so increased the size to 4096x4096x800. With this larger size, I increased the octaves from 6 to 8 because there was a lack of detail. There was now a noticeable loading time, but I think it is worth it for the added size. This process can be seen below.





Date: 16/12/22

Now, I wanted to apply a gradient over the square terrain. This will make it circular, and the height will trail off towards the end of the terrain. I edited the "TerrainGeneration" script, and initially made two new variables called "xMid" and "zMid". These will store their respective coordinates for the midpoint of the terrain. I also removed the "public" from all the variables since I do not need to change them anymore. I had to make "xSize", "zSize" and "ySize" static variables so that I could perform calculations on them for the midpoints. I also reduced the octaves back down to 6 since the loading time was too long with the new gradient calculations. Next, I edited the "CalculateHeight" function, and created two new "xDist" and "zDist" variables. These store the x and z distances between the current point and the midpoint of the terrain. Then, I used Pythagoras to determine the real distance from the point and the midpoint and stored it in the "dist" variable. Next, I created a "cutoff" variable, to store the maximum distance before a gradient is applied. I used an if statement to check whether the current distance is larger than the "cutoff". If it is, then I calculated the "extra" distance, or in other words how much over the cutoff the current point is. Then, I reduced the gradient value, which is originally 1, by 0.0005 for every unit over the cutoff the current point is. Then, I multiply the "y" value for the current point by the newly calculated "gradient" value. This produces a circular terrain which trails off at the end. Test 7 from the design section will now be passed. This process can be seen below in the form of annotated screenshots.

```
// declare terrain size variables
static int xSize = 4096; // x axis
static int zSize = 4096; // z axis
static int ySize = 800; // y axis

// calculate midpoint
int xMid = xSize / 2;
int zMid = zSize / 2;

// declare perlin noise variables
float scale = 4f;
float xOffset; // random value
float zOffset; // random value
int octaves = 6;
float lacunarity = 2f; // frequency multiplier
float persistence = 0.5f; // amplitude multiplier
```

Calculate the terrain midpoint in terms of x and z.

```
// calculate distance from centre
int xDist = Mathf.Abs(xMid - x);
int zDist = Mathf.Abs(zMid - z);
float dist = Mathf.Sqrt(Mathf.Pow(xDist, 2) + Mathf.Pow(zDist, 2));
```

Calculate the distance from midpoint to current point in terms of x and z.

Calculate true distance using Pythagoras.

$a^2+b^2=c^2$

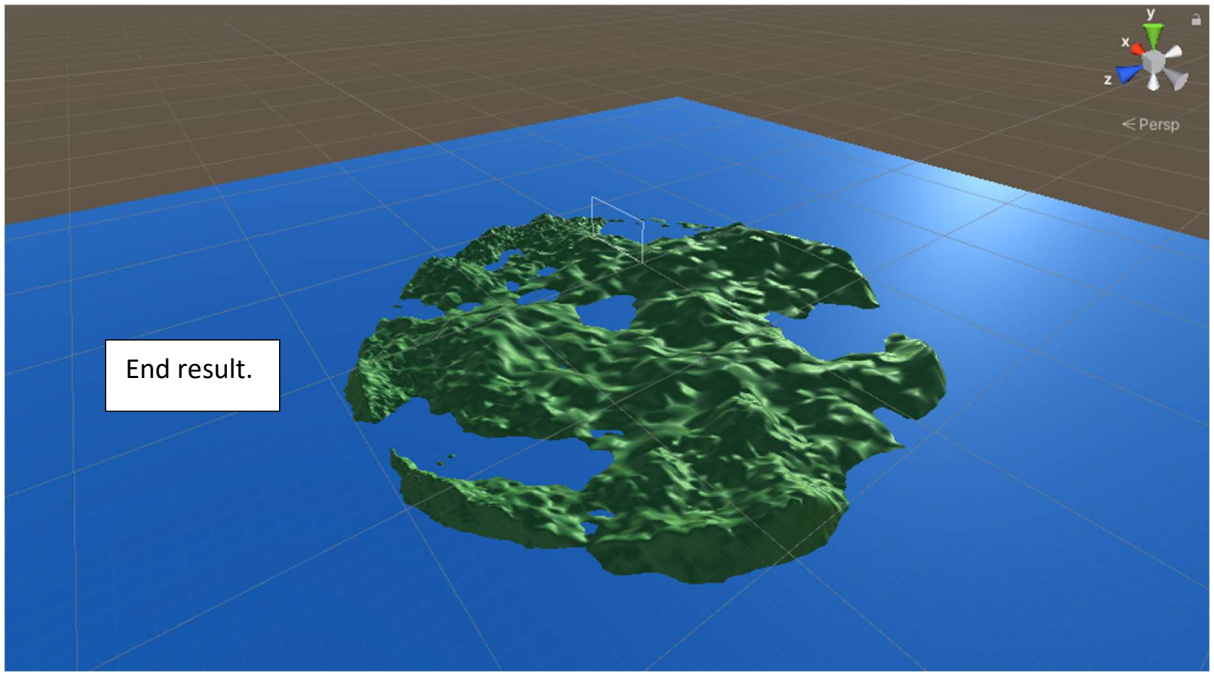
```
// calculate the gradient multiplier
int cutoff = xMid - 200;
float gradient = 1f;
if (dist > cutoff)
{
    float extra = dist - cutoff;
    gradient = gradient - (extra * 0.0005f);
}
```

Calculate the max distance before gradient is applied.

Reduce the gradient multiplier for every unit the current point is above the cut off.

```
for (int i = 0; i < octaves; i++)
{
    y = y + (amplitude * Mathf.PerlinNoise(xCoord * frequency, zCoord * frequency));
    y = y * gradient;
    frequency = frequency * lacunarity;
    amplitude = amplitude * persistence;
}
```

Multiply the “y” value by the gradient multiplier.



End result.

Date: 16/12/22

At the moment, I have set the house and character positioning to be up in the sky, so when the player loads in, they fall into the environment. This is no use since they cannot get back to their house. I wanted to find a quick fix to this and decided to just move the character and house object to the same y-value as the height given to the point at (0, 0). I had to edit the "TerrainGeneration" script, so that it referenced the "house" and "character" objects. Then, within the "CalculateHeight" function I added a new if statement. It checks whether the current point was (0, 0) and if it is, then the house and character objects are given the point's height as their height. After referencing the objects in the editor, the solution worked. In short, this solution keeps the "Character" and "House" objects at (0, 0, 0) until the centre point of the terrain is processed. Once this is done, the height value for these objects is updated with that of the Perlin noise function. Test 3 and 4 will now be passed from the design section. This process can be seen below.

```
// reference gameobjects
public GameObject house;
public GameObject character;
```

Reference to the "House" and "Character" objects. Stored them in variables called "house" and "character" respectively.

```
// check if centre, move gameobjects
if (dist == 0)
{
    float height = (y * 800) - 300;
    // check if it is below water level
    if (height < 0f)
    {
        height = 0f;
    }
    // move the house to new height
    house.transform.position = new Vector3(0, height, 0);
    // move the character to new height
    character.transform.position = new Vector3(0, (height + 1.08f), 0);
}
```

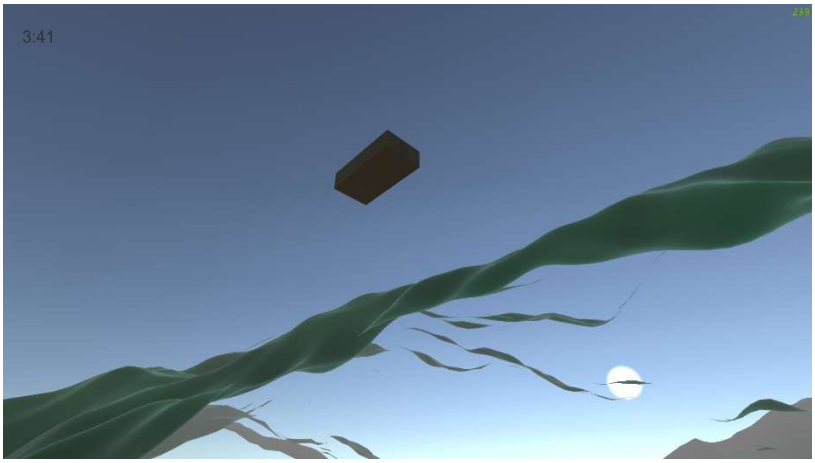
Here I convert the y-value to a coordinate and store it in the "height" variable.

Checks whether it is under water level, and if it is, changes height to 0.

Moves the objects to their new heights.

Date: 17/12/22

Here, I noticed some bugs, and also wanted to improve the lighting in the scene. When loading into the outside scene, sometimes the player will fall through the terrain, usually when it is spawned onto a steep edge. To fix this, I just increased the height that the player is moved to slightly. Due to the loading time, the player can't even notice the character fall. I also moved the character object slightly further away from where the door is moved to, to avoid the player accidentally going back into their house. Unfortunately, this did not solve the issue. I tried another approach using two if statements to check whether the loop was on a certain x and z combination, and then move the "House" and "Character" objects respectively with their assigned x and z combinations. This also yielded no better results. After testing a little longer, I realised this actually had nothing to do with the height of the player, but in fact sometimes the player just wasn't being moved. This occurred whenever I changed scenes and tried again. What was peculiar is that the house was being moved. After searching the internet, I found that the character controller does not pair well with the "transform.position" function. So, to fix the issue, I had to disable the character controller, move the character, then enable it again. With this finished, I could improve the lighting. Firstly, I added fog in linear mode, with a grey-blue colour, based off of the skybox. This added more depth into the game. Next, I increased the "Clipping planes" from 1000 to 5800, since I was finding it very annoying not being able to see the whole landscape. I picked this value since it is the rough diagonal length across the landscape. This ensures that at any point on the terrain, the player can see the other end of it. Here, I started to notice weird lighting bugs when exploring the landscape. I believe this is because I was using baked lighting, which is obviously no use when the landscape is changing every time. However, it is fine for me to continue using this method for the "Inside" scene. To achieve this, I enabled "Realtime lighting", which created a settings file for the "Outside" scene. This worked much nicer. This process can be seen below.



Here, the character was not being moved above the terrain.

```
// move the house to new height
house.transform.position = new Vector3(0, height, 0);
// move the character to new height
character.transform.position = new Vector3(-2, (height + 6f), 0);
```

At first, I thought the character was clipping through the terrain object, so tried to just increase the height it was spawned in at.

When this didn't work, I tried to assign x and z combinations to move the character and house object individually, to get the perfect heights. This also didn't

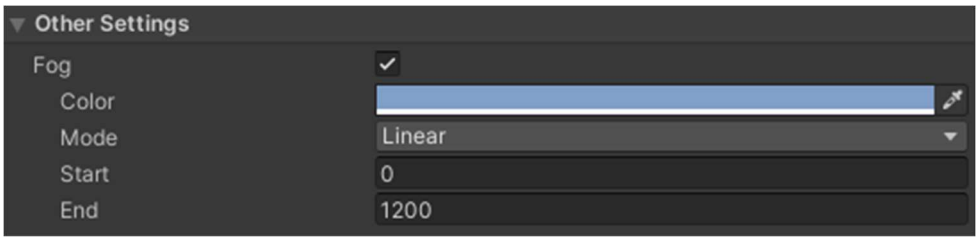
```
// check if x and z combination, move gameobjects
if (x == xMid && z == zMid)
{
    // convert terrain y value to height coordinate
    float height = (y * 800) - 300;
    // check if it is below water level
    if (height < 0f)
    {
        height = 0f;
    }
    // move the house to new height
    house.transform.position = new Vector3(0, height, 0);
}
else if (x == xMid - 2 && z == zMid)
{
    // convert terrain y value to height coordinate
    float height = (y * 800) - 300;
    // check if it is below water level
    if (height < 0f)
    {
        height = 0f;
    }
    // move the character to new height
    character.transform.position = new Vector3(-2, (height + 1.08f), 0);
}
```

```
// reference gameobjects
public GameObject house;
public GameObject character;
public CharacterController controller;
```

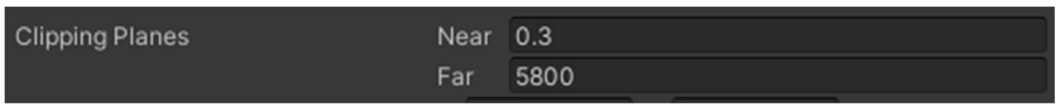
After some research, I found that the character controller was creating issues when paired with "transform.position".

```
// move the house to new height
house.transform.position = new Vector3(0, height, 0);
// move the character to new height
controller.enabled = false;
character.transform.position = new Vector3(-2, height + 4f, 0);
controller.enabled = true;
```

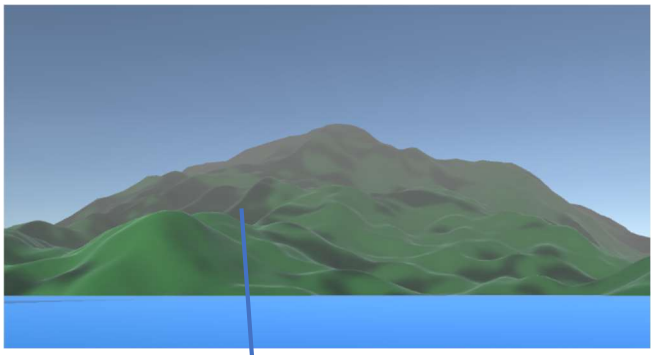
So, I disabled the character controller, moved the character, then enabled it again. This solved the

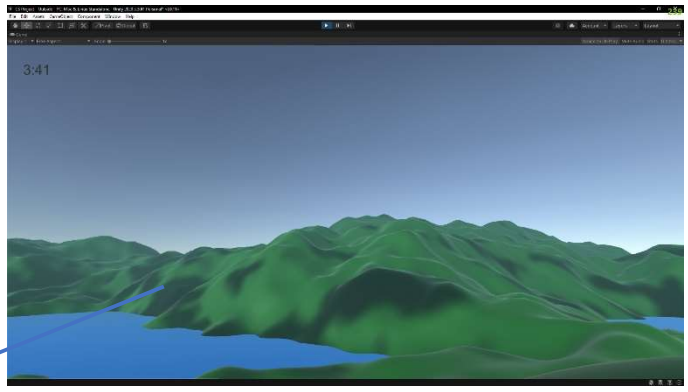


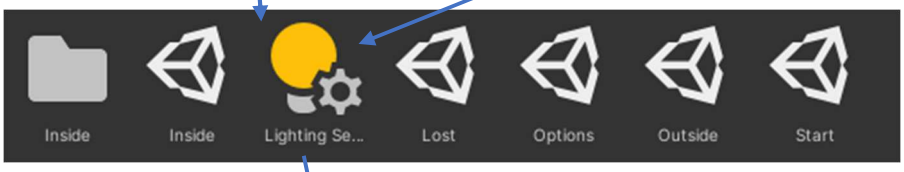
Here, I used the colour picker tool on the skybox to find the right fog colour. Then, I set the mode to linear and found start and end points that worked nicely with the game.

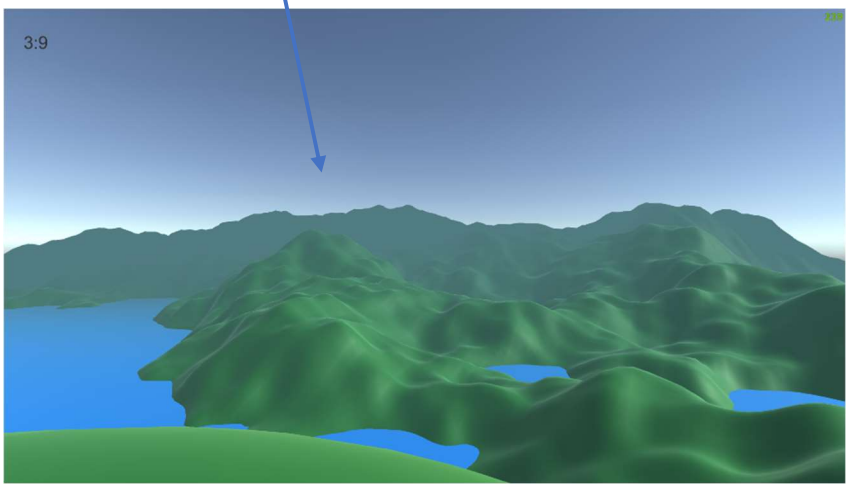


Here, I changed the "Far" value from 1000 to 5800. This ensures that the player can see any of the landscape from any point on the terrain. This is rewarding when they climb up a mountain to find their house for example. Far away landscape should not be "clipped" off.







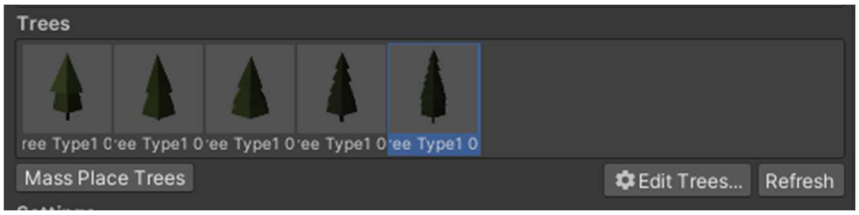


At this point, I was finding lighting bugs across the landscape. This was due to using baked lighting. I created a "Realtime lighting" settings file for this scene, which meant I used realtime lighting instead of baked lighting for the "Outside" scene. This solved the problem and looked much better.

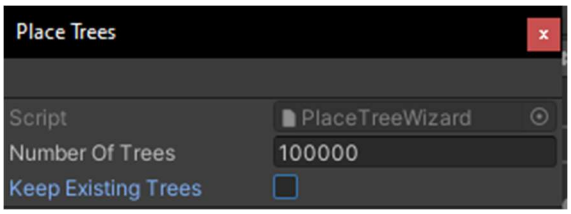
[4] Explore mechanic

Date: 17/12/22

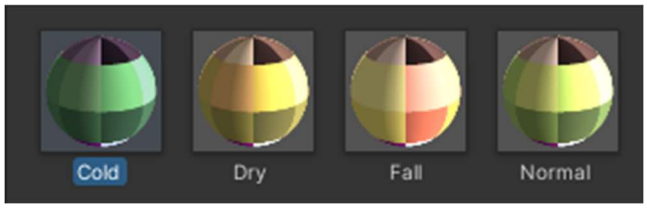
In this small section of development, I wanted to take advantage of an inbuilt feature of Unity, which is distributing trees across a terrain. Firstly, I found a package[22] on the Unity asset store that included many different types of low-poly trees. I downloaded and installed this package. I then navigated to my "Terrain" object and selected the "Paint trees" option. I had to import all the new trees into this menu. Under the "Mass place trees" heading, I entered "100000" trees, and deselected "Keep existing trees". This means that every time the scene is loaded, 100,000 trees will be randomly placed across the landscape. I cannot do this method with the plants and water items, since I need them to be individual objects that the player can interact with. I had to change all the package's materials to "Nature soft occlusion" to avoid an error. Test 1 should now be passed from the design section. This process can be seen below.

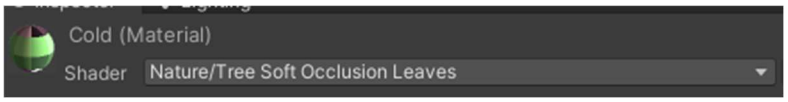


Added the package trees into the "Trees" section of the "Terrain" object settings.

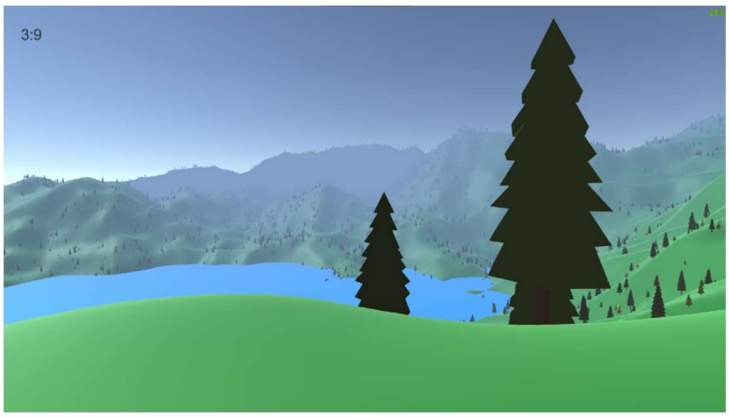


Assigned 100,000 of these 5 tree types to be randomly distributed across the landscape.





Here, I had to assign the package's materials the "soft occlusion" shader to avoid error.



This was the end result. Trees are randomly distributed across the terrain, and it looks great.

[22] (Studios 2022)

Date: 18/12/22

I realised that along the way, I had left some untidy code in the last couple milestones. Now I will take some time to clean it up. Initially, I thought that the "CalculateHeight" function within the "TerrainGeneration" script was getting a bit too large and could be split up. So, I move the section that was checking if the current point was at the centre to another function called "MoveObjects". The other issue was that the timer variable was being stored locally, and I could move the variable into the "Global" script that I have. This is because I need to access this variable across multiple scenes in the future, when dealing with plant growth. With these issues addressed, I am happy with the tidiness of my coded solutions. This process can be seen below.

```
// move objects if necessary
MoveObjects(dist, y);
```

```
void MoveObjects(float distance, float y)
{
    // check if current point is at centre
    if (distance == 0f)
    {
        // convert terrain y value to height coordinate
        float height = (y * 800) - 300;
        // check if it is below water level
        if (height < 0f)
        {
            height = 0f;
        }
        // move the house to new height
        house.transform.position = new Vector3(0, height, 0);
        // move the character to new height
        controller.enabled = false;
        character.transform.position = new Vector3(-2, height + 4f, 0);
        controller.enabled = true;
    }
}
```

Moved the code here into its own function, to ensure I do not make my functions too generic.

```
// variable to store timer length
private float timeRemaining = 240;
// boolean to check whether timer is running
private bool timerRunning = false;
// reference to the timer text object
public Text timerText;
```

```
public class Global : MonoBehaviour
{
    // variable to store mouse sensitivity
    public static float sensitivity = 1000f;
    // variable to store timer length
    public static float timeRemaining;
}
```

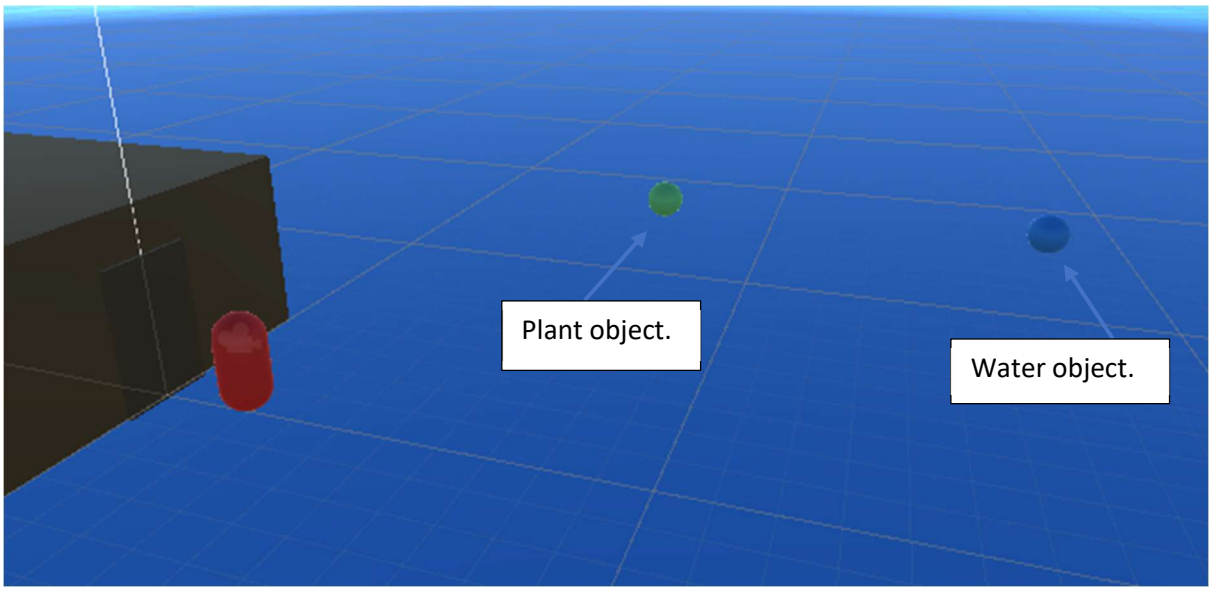
```
// Update is called once per frame
void Update()
{
    if (timerRunning)
    {
        // if there is time left on the timer
        if (Global.timeRemaining > 0)
        {
            // subtract the time length of the last frame
            Global.timeRemaining -= Time.deltaTime;
            // calculate minutes and seconds
            int minutes = Mathf.FloorToInt(Global.timeRemaining / 60);
            int seconds = Mathf.FloorToInt(Global.timeRemaining % 60);
            // push this to the text object
            timerText.text = minutes.ToString() + ":" + seconds.ToString();
        }
        else
        {
            // end the timer
            Global.timeRemaining = 0;
            timerRunning = false;
            // timer has run out, change the scene
            SceneManager.LoadScene("Lost");
        }
    }
}
```

```
// Start is called before the first frame update
void Start()
{
    // assign the timer length
    Global.timeRemaining = 240;
    // starts timer automatically
    timerRunning = true;
}
```

Here, I moved the creation of the "timeRemaining" variable over to the "Global" script. I had to set the length in the "Start" function so that it reset every time the player explored. Then, I had to change every reference of "timeRemaining" in the "Timer" script to "Global.timeRemaining". This worked great.

Date: 18/12/22

Here, I wanted to distribute the plant and water items. For now, I decided to keep it simple, and create spherical objects of green and blue to represent these items. From here, I will use random numbers to determine whether one will be placed at a certain point. The first thing I did was create two new “Sphere” objects, calling them “Plant” and “Water”. I then scaled the spheres down to 0.5 and created a new “Plant” material. I will reuse the “Water” material that I have already created. From here, I need to use the “Object.Instantiate” function to clone/create these objects across the landscape, randomly. Firstly, I created two new empty parent objects, called “Plants” and “Waters”. I know that waters does not make proper grammatical sense, but it aided my understanding. I moved the “Plant” and “Water” object into their respective parent objects. Now, I could head into the “TerrainGeneration” script and edit the “CalculateHeight” function. I added a call to a function called “CreateObjects” which I will now develop. I had to write some code to convert the x, y and z values in my script to that inside the unity editor. Now, I could generate a random integer between 0 and 99, using the “Random.Range” function. From here, I could check if the number was a certain value, and from this create a plant or water object in the current position. To do this, I had to create a reference earlier on in the script to the “Plant” and “Water” objects, since these are the objects I will be cloning. I also had to reference the parent objects “Plants” and “Waters”. Then I could use the “Instantiate” function to create new objects with parameters: “original”, “position”, “rotation”, “parent”. After referencing the objects, the script seemed to work. Unfortunately, this was far from the truth. The two main issues were as follows: there were way too many objects being produced, and the heights were completely wrong. To solve the first issue, I changed the probability of a plant or water spawning from 1 in 100, to 1 in 1000. To solve the second, I had to swap over the x and z values when assigning the “xCoord” and “zCoord” values. This solved the problem. There were still too many plant and water objects, so I cut it down to 1 in 10,000 instead. There were still too many, so I cut it down to 1 in 100,000. Now I was happy. Test 2 from the design section will now be passed. This process can be seen below in the form of annotated screenshots.



Here, I made parent objects for the plant and water objects, which will eventually be cloned. The clones will have the same parent object.

```
// reference gameobjects
public GameObject house;
public GameObject character;
public CharacterController controller;
public GameObject plant;
public GameObject water;
```

```
// reference transforms
public Transform plants;
public Transform waters;
```

```
// create objects if necessary
CreateObjects(x, z, y);
```

This function will use a random number generator to decide at any given point on the terrain whether to place a plant or water object. The "Plant" or "Water" object will be cloned and will be assigned the "Plants" or "Waters" parent, and a new location.

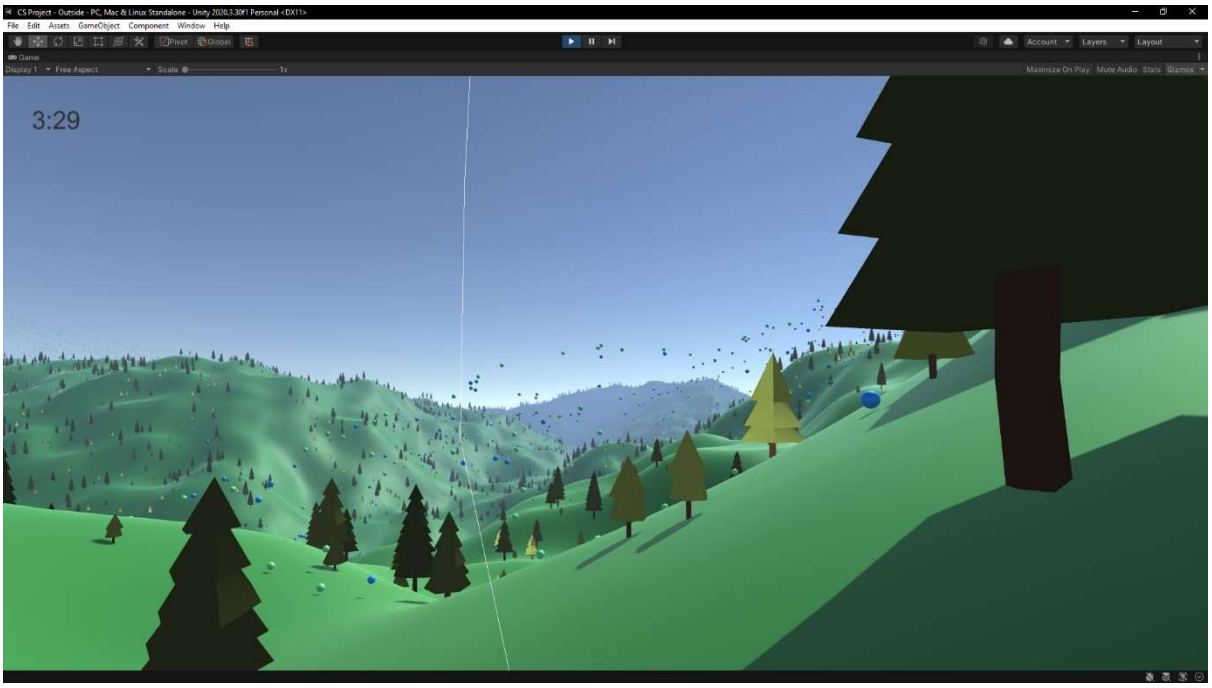
```
void CreateObjects(int x, int z, float y)
{
    // convert the x, z and y values to coordinates
    float yCoord = (y * ySize) - 300;
    float xCoord = (x - xMid);
    float zCoord = (z - zMid);

    // store these in a vector3 variable
    Vector3 position = new Vector3(xCoord, yCoord, zCoord);

    // generate a random number between 0 and 99999
    int num = Random.Range(0, 100000);
    // check number, perform necessary action
    switch (num)
    {
        case 0:
            // create a plant object
            Instantiate(plant, position, Quaternion.identity, plants);
            break;
        case 1:
            // create a water object
            Instantiate(water, position, Quaternion.identity, waters);
            break;
        default:
            break;
    }
}
```

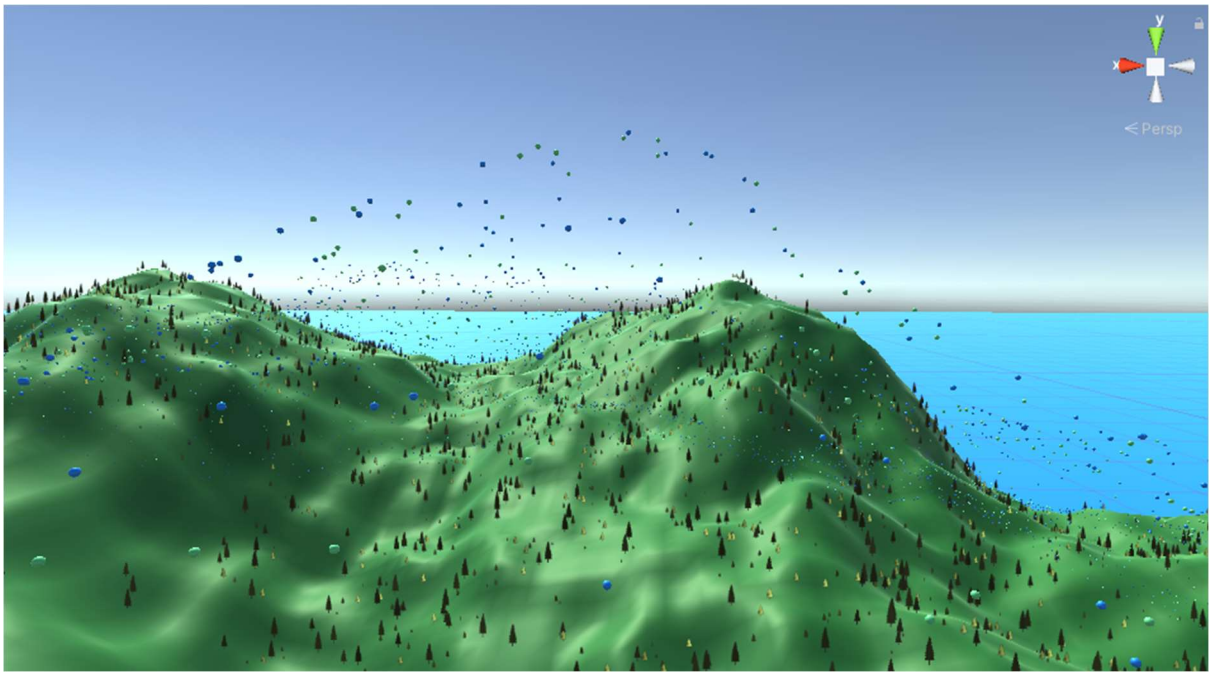
Instantiate(original, position, rotation, parent)

Represents an empty rotation, necessary because the function takes 4 arguments.

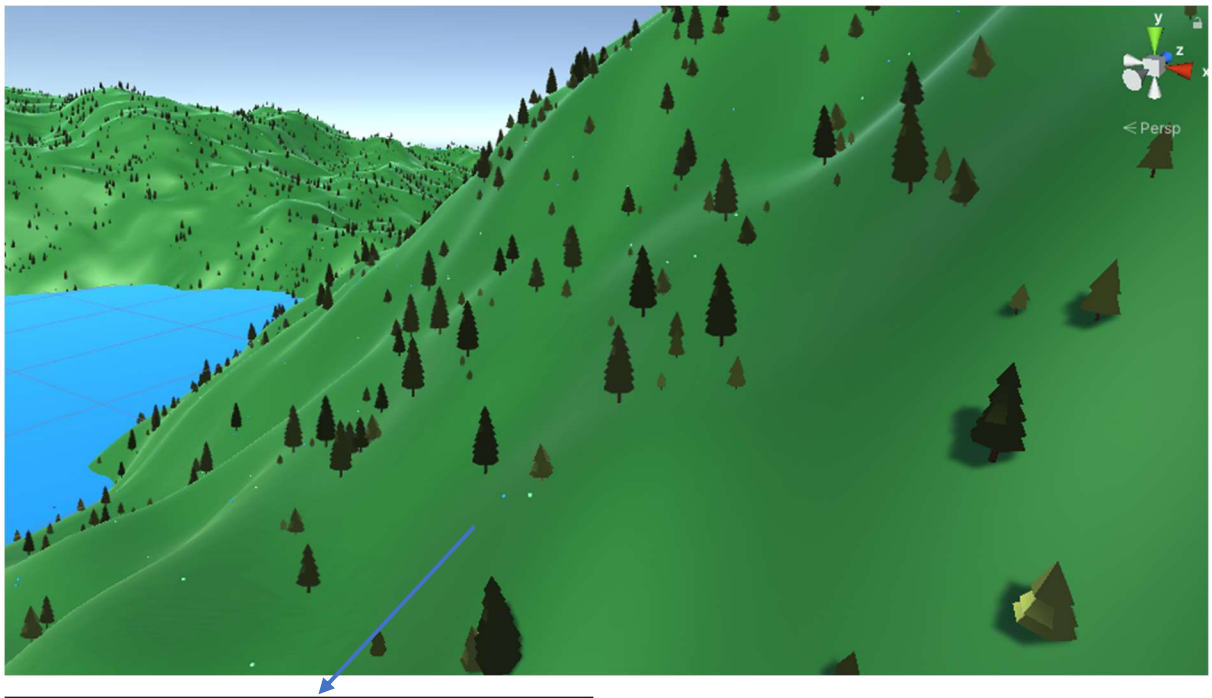
```
// convert the x, z and y values to coordinates
float yCoord = (y * ySize) - 300;
float xCoord = (z - xMid);
float zCoord = (x - zMid);
```

My for loop was not similar to the coordinate grid, had to swap x and z round.





This was the issue I was running into before swapping the x and z variables when converting them to coordinates.



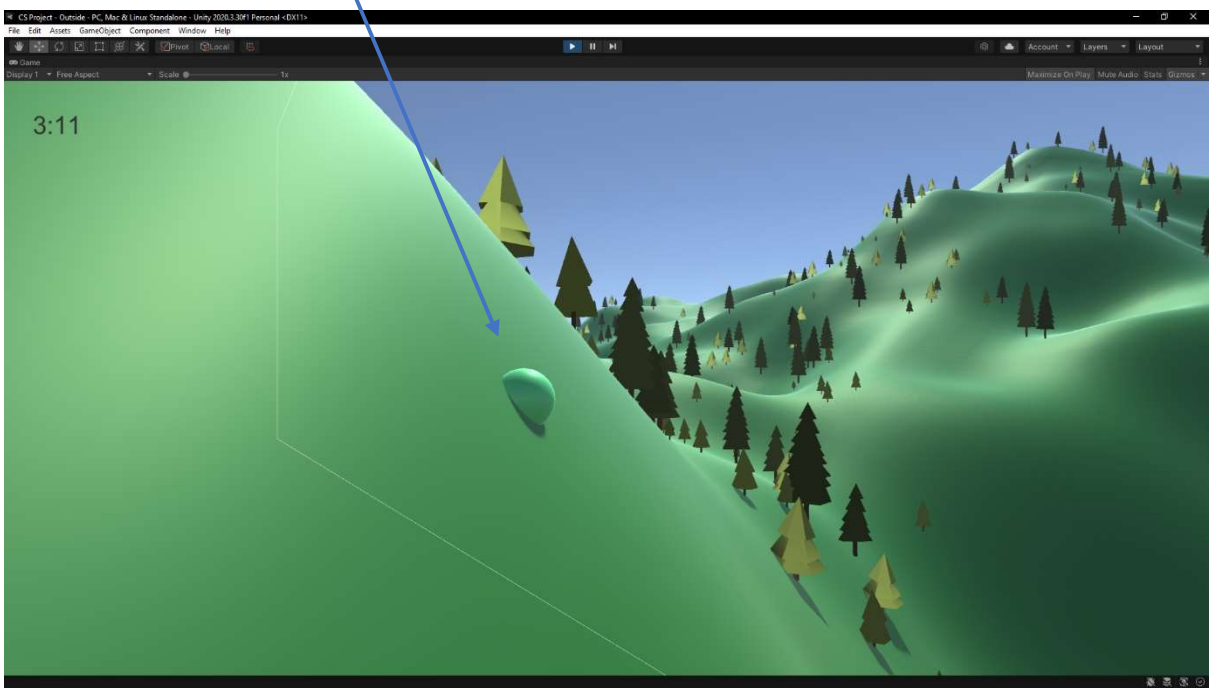

```
// generate a random number between 0 and 99
int num = Random.Range(0, 100);
```

```
// generate a random number between 0 and 999
int num = Random.Range(0, 1000);
```

```
// generate a random number between 0 and 9999
int num = Random.Range(0, 10000);
```

```
// generate a random number between 0 and 99999
int num = Random.Range(0, 100000);
```

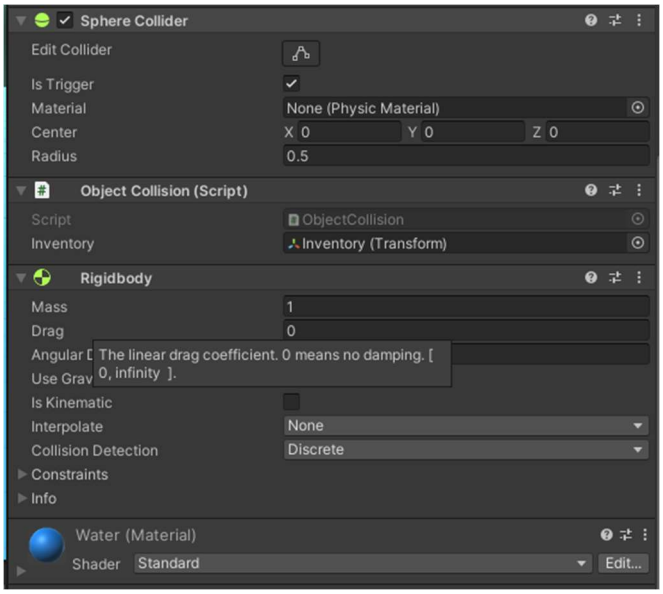
Here, I tested decreasing the chance of a plant/water object spawning. I thought it would be much better if they were scarce and hard to find, making the game harder and making it easier for the player to get lost.

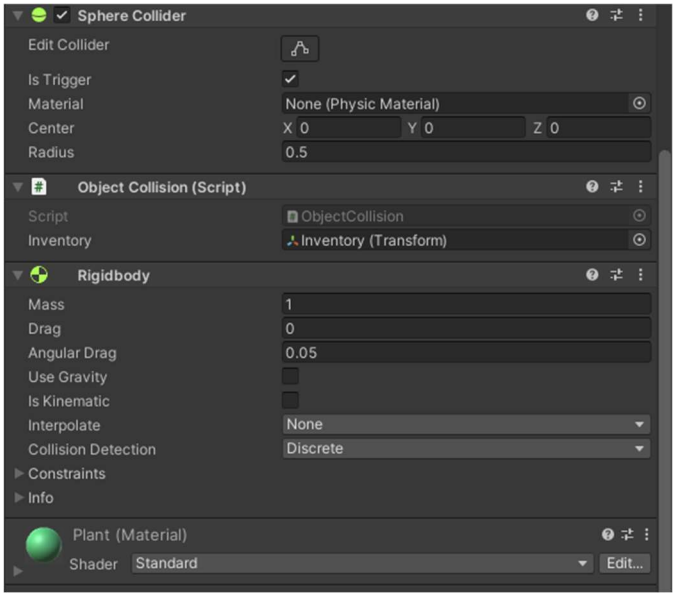


[4] Score system

Date: 19/12/22

Now, I needed to move the plant/water objects whenever a player collided with them. They should be moved into an "Inventory" parent transform, which I can take advantage of later when constructing an inventory system. The score will increase by 1 every time a player collects a plant/water object. So, to do this, I had to use object collisions, which I will use tags for. I will not use sphere projection like before, since I need to perform operations on the exact object that the player collided with. So, I created a new script called "ObjectCollision". I used the "OnTriggerEnter" function to check whether the plant/water objects are touched by the player. I then checked the tag, and if the tag is "Player", the plant/water object are moved to (-20, -5, 0), which is out of sight to the player. Then, I changed the plant/water parent object to "Inventory". I had to give all plant and water objects a "Rigidbody" for this mechanic to work. I then assigned the script to the "Plant" and "Water" objects which will be cloned. I then added a line to the script which adds a point to the global score variable, which I will add in later. This script was now complete. I had to give the "Capsule" object within the "Character" object the "Player" tag, since it was the object with the collider. Now, I created a new object within the "UI" parent object and called it "Score". I gave this a text component. I resized it and anchored it properly. Then, I instantiated the "score" variable within the "Global" script. Now I could create a new script, and I called it "Score". All this script had to do for now was set the score to 0 in the "Start" function and update the text in the "Update" function. Then I attached it to the "Score" object in the "UI" parent. After referencing the text component, the script worked well. Now, every time the player walks over an item, it is moved out of sight, into the "Inventory" parent item, and the score is increased by 1. This implementation was helped greatly by the official Unity documentation[23]. Test 4, 5 and 6 from the design section can now be passed. This process can be seen below.



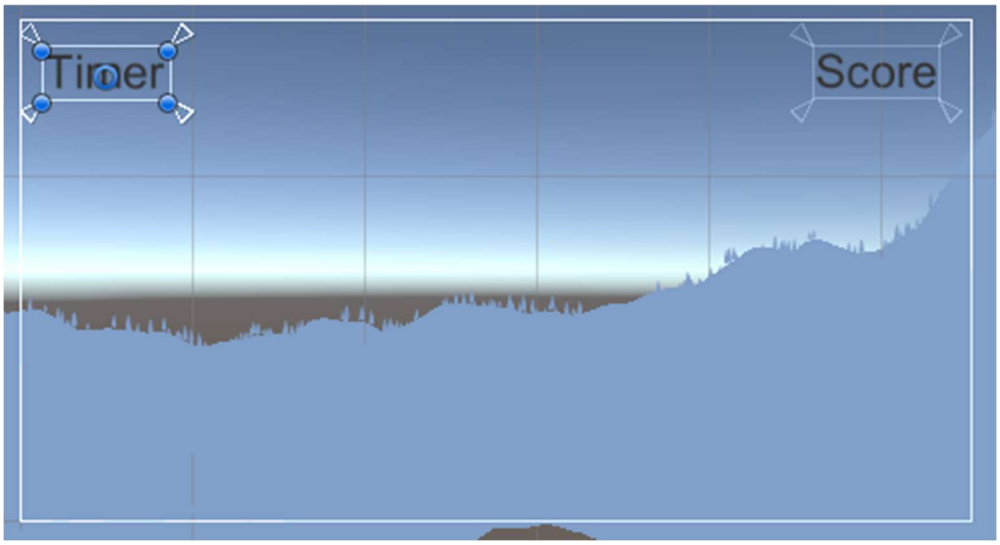


Here, I give the "Plant" and "Water" objects the "ObjectCollision" script as a component. I had to drag and drop the "Inventory" parent item into the reference point. I also had to select "Is Trigger" on these objects. Finally, I had to add a rigid body as per the Unity documentation.

[23] (Unity, Collider.OnTriggerEnter(Collider) 2022)

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

public class ObjectCollision : MonoBehaviour
{
    // reference the transforms
    public Transform inventory;

    // check for collisions
    void OnTriggerEnter(Collider col)
    {
        if (col.gameObject.tag == "Player")
        {
            // move the object out of view
            transform.position = new Vector3(-20, -5, 0);
            // change the parent to inventory
            transform.parent = inventory;
            // add one to the score
            Global.score += 1;
        }
    }
}
```

Here, I created the new "ObjectCollision" script, in which I put the "OnTriggerEnter" script. This then checks if the plant/water object collides with any object with the "Player" tag. This is the "Character" object. If so, the plant/water is moved underneath the map, and given the "Inventory" parent object. The score is then increased by 1.

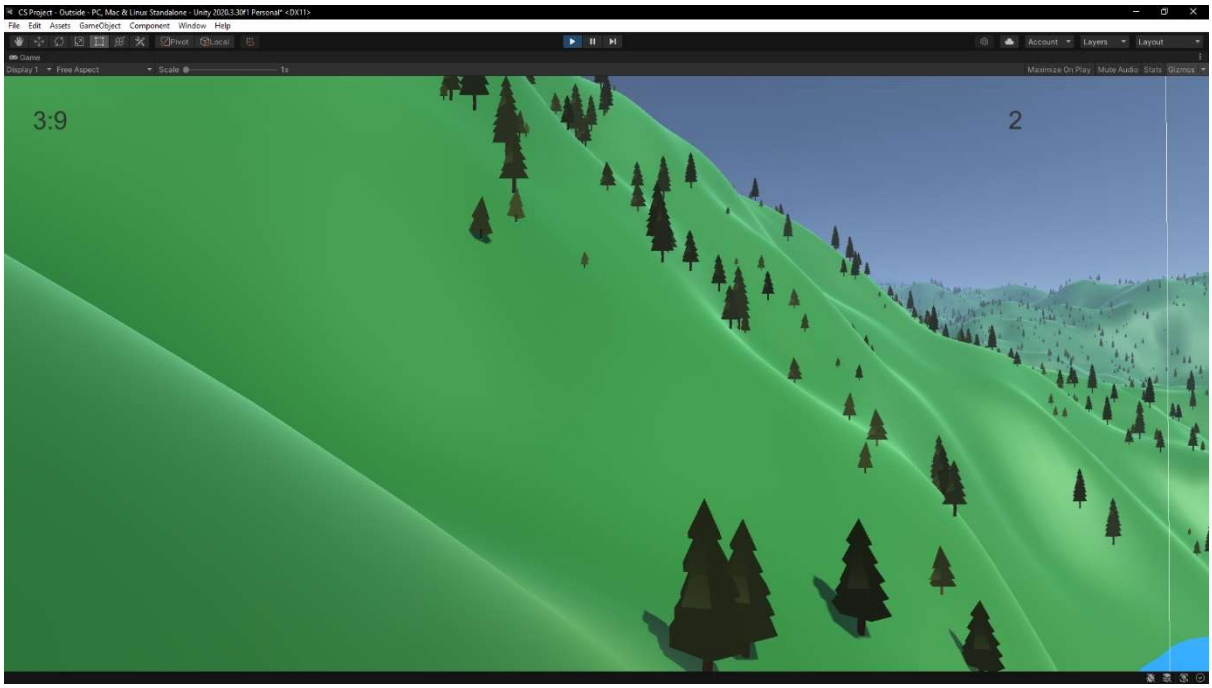


Here, I duplicated the "Timer" object, then renamed it to "Score. After this, I renamed the "Text" attribute to "Score". Finally, I moved it onto the other side of the screen and adjusted the anchors.

```
public class Global : MonoBehaviour
{
    // variable to store mouse sensitivity
    public static float sensitivity = 1000f;
    // variable to store timer length
    public static float timeRemaining;
    // variable to store player score
    public static int score;
}
```

Here, I instantiated the “score” variable within the “Global” script. Then, I created a new “Score” script, which sets the score to 0 at the start of the scene, and then updates the score every frame by checking the “Global.score” variable. The “Score” text attribute is then set to “Global.score”

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

public class Score : MonoBehaviour
{
    // reference to the score text object
    public Text scoreText;

    // Start is called before the first frame update
    void Start()
    {
        Global.score = 0;
    }

    // Update is called once per frame
    void Update()
    {
        scoreText.text = Global.score.ToString();
    }
}
```



[4] Explore mechanic

Date: 20/12/22

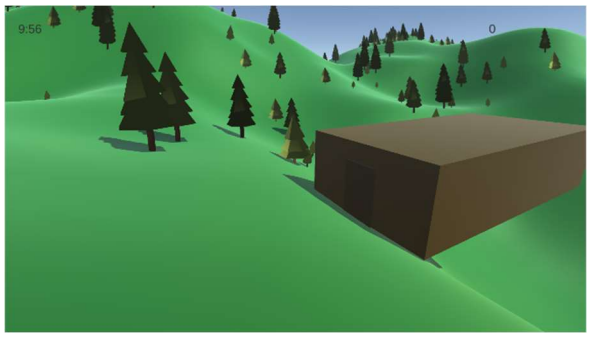
Here, I wanted to quickly add a new feature into the game. I thought it would be good to assign the fog distance to the time left on the timer in seconds. At first, it was a bit too close, so I multiplied the distance to be the time remaining multiplied by 2. This worked well. The process can be seen below.

```
// if there is time left on the timer
if (Global.timeRemaining > 0)
{
    // subtract the time length of the last frame
    Global.timeRemaining -= Time.deltaTime;
    // calculate minutes and seconds
    int minutes = Mathf.FloorToInt(Global.timeRemaining / 60);
    int seconds = Mathf.FloorToInt(Global.timeRemaining % 60);
    // push this to the text object
    timerText.text = minutes.ToString() + ":" + seconds.ToString();
    // make fog end distance equal to time left
    RenderSettings.fogEndDistance = Global.timeRemaining;
}
```

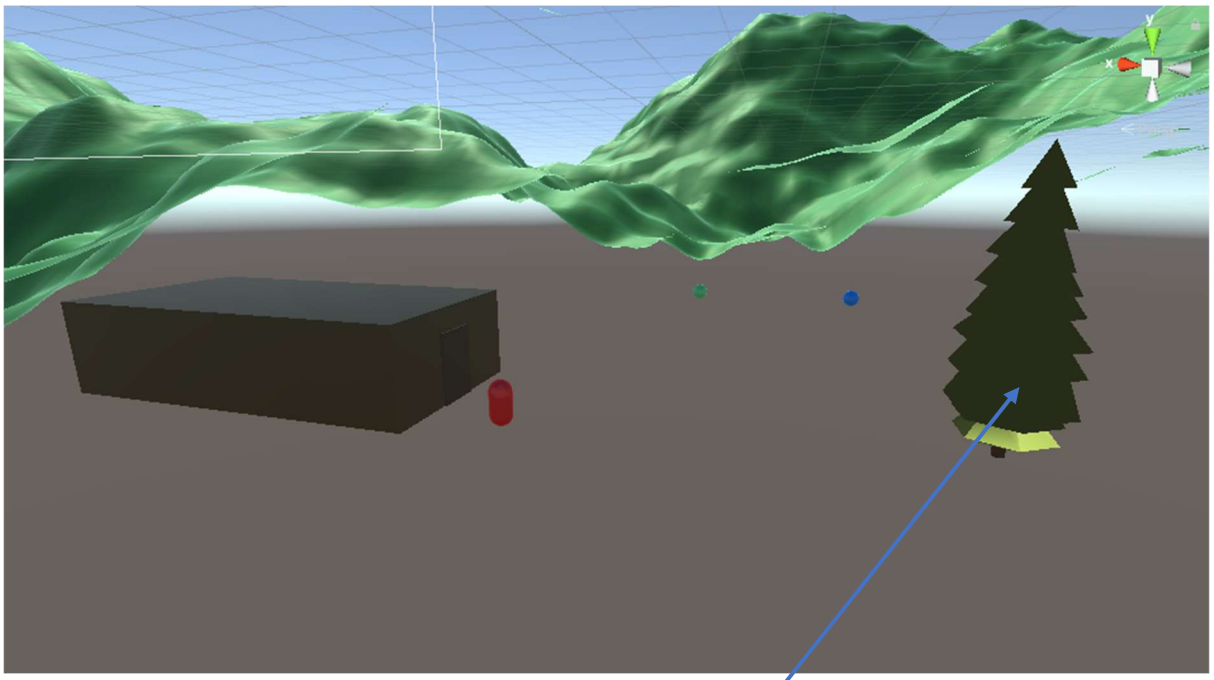
```
// make fog end distance equal to time left
RenderSettings.fogEndDistance = Global.timeRemaining * 2;
```

Date: 20/12/22

At this point, I noticed that my implementation of trees was not very good and was taking away from the random aspect of the game. I found two issues; the trees were not being randomly distributed, but instead their heights were just changing, and trees were poking through the “Water” plane. This was my fault for using Unity’s inbuilt implementation of trees on a terrain. Now I will deploy a full solution for this. The first step was to delete the “Mass placed trees” so that I can start to add the trees back in properly. After this, I created a new empty object called “Trees” to act as a parent object. I then added all the “Tree Type 1” prefabs from the package to the “Trees” object. I renamed them to “Tree1”, “Tree2”, “Tree3”, “Tree4” and “Tree5”. I then moved them all under the map where the player could not see them. I then began to edit the “TerrainGeneration” script, specifically the “CreateObjects” function. I added 5 more cases in the switch statement, to place these 5 different trees. To optimise this function, I also checked whether the height was less than 0, and if so, the code should exit, since no plants need to be spawned below sea level. I had to make a few more references to these tree objects and their parent object. Next, I had to reference the correct objects in the editor. There were not enough trees being spawned. I also thought my solution was not very efficient. To allow for expansion, I generate a list of objects from the parent objects in the “Start” function. Then I can use a second random number generator to pick which tree to use from the list. This will be helpful when I introduce different rarity plants. I was now happy with my solution. The process can be seen below.



Here you can see the two issues I was running into. The trees were not actually random and were the same each time. Also, the trees were spawning in water.



This is where I hide all my objects underneath the map. The player cannot access this space.

```
// reference gameobjects
public GameObject house;
public GameObject character;
public GameObject plant;
public GameObject water;
public GameObject tree1;
public GameObject tree2;
public GameObject tree3;
public GameObject tree4;
public GameObject tree5;

// reference transforms
public Transform plants;
public Transform waters;
public Transform trees;
```

This is my first solution to the problem. I reference all the tree objects individually and assign them their own case in the switch statement. This did not produce enough trees, so I had to look for other solutions. There was no space for expansion here, and it was using up a lot of lines.

```
void CreateObjects(int x, int z, float y)
{
    // convert the x, z and y values to coordinates
    float yCoord = (y * ySize) - 300;
    float xCoord = (z - xMid);
    float zCoord = (x - zMid);

    // check if below water level
    if (yCoord < 0)
    {
        // exit this function
        return;
    }

    // store these in a vector3 variable
    Vector3 position = new Vector3(xCoord, yCoord, zCoord);

    // generate a random number between 0 and 99999
    int num = Random.Range(0, 100000);
    // check number, perform necessary action
    switch (num)
    {
        case 0:
            // create a plant object
            Instantiate(plant, position, Quaternion.identity, plants);
            break;
        case 1:
            // create a water object
            Instantiate(water, position, Quaternion.identity, waters);
            break;
        case 2:
            // create a tree1 object
            Instantiate(tree1, position, Quaternion.identity, trees);
            break;
        case 3:
            // create a tree2 object
            Instantiate(tree2, position, Quaternion.identity, trees);
            break;
        case 4:
            // create a tree3 object
            Instantiate(tree3, position, Quaternion.identity, trees);
            break;
        case 5:
            // create a tree4 object
            Instantiate(tree4, position, Quaternion.identity, trees);
            break;
        case 6:
            // create a tree5 object
            Instantiate(tree5, position, Quaternion.identity, trees);
            break;
        default:
            break;
    }
}
```

```
// declare lists
List<GameObject> plantList;
List<GameObject> waterList;
List<GameObject> treeList;
```

```
// reference gameobjects
public GameObject house;
public GameObject character;
```

Here, I removed most of the game object references. Instead, I created three new lists called "plantList", “waterList” and “treeList”. I then populate these in the “Start” function, with all their respective children items.

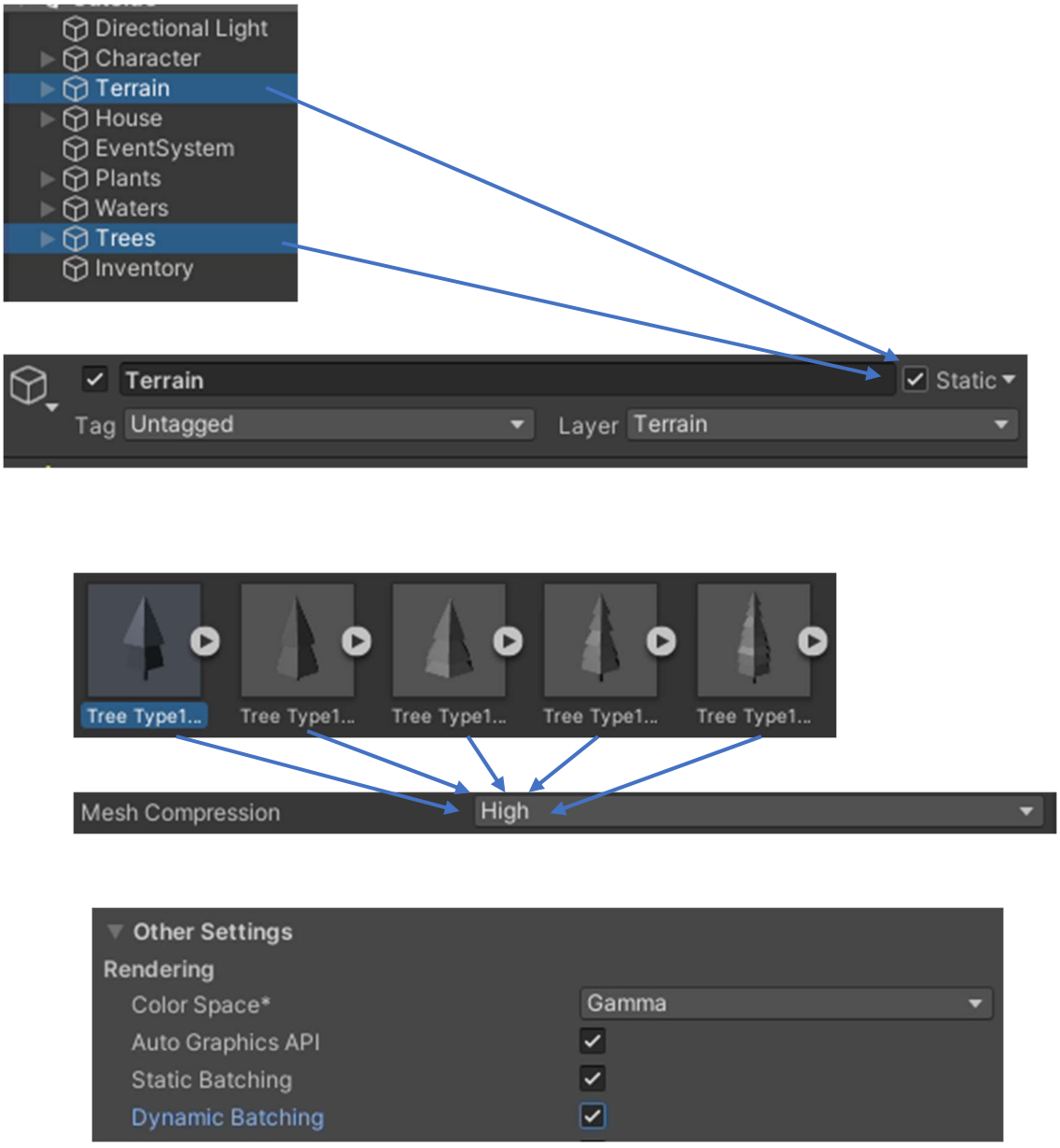
```
// Start is called before the first frame update
void Start()
{
    // generate list of plants
    plantList = new List<GameObject>();
    foreach (Transform child in plants)
    {
        plantList.Add(child.gameObject);
    }

    // generate list of waters
    waterList = new List<GameObject>();
    foreach (Transform child in waters)
    {
        waterList.Add(child.gameObject);
    }

    // generate list of trees
    treeList = new List<GameObject>();
    foreach (Transform child in trees)
    {
        treeList.Add(child.gameObject);
    }

    // calculate random offsets
    xOffset = Random.Range(0f, 9999f);
    zOffset = Random.Range(0f, 9999f);

    // reference terrain component to change data
    Terrain terrain = GetComponent<Terrain>();
    // create new terrain based off of current terrain
    terrain.terrainData = GenerateTerrain(terrain.terrainData);
}
```

With this done, I can use a second random number to determine which tree will be spawned. This will save me time when I come to add more and more objects into the landscape.

```
// generate a random number between 0 and 99999
int num1 = Random.Range(0, 100000);
// check number, perform necessary action
switch (num1)
{
    case 0:
        // create a plant object
        Instantiate(plantList[0], position, Quaternion.identity, plants);
        break;
    case 1:
        // create a water object
        Instantiate(waterList[0], position, Quaternion.identity, waters);
        break;
    case int n when (n >= 10 && n <= 200):
        // generate a random number between 0 and 4
        int num2 = Random.Range(0, 5);
        // create a tree object
        Instantiate(treeList[num2], position, Quaternion.identity, trees);
        break;
    default:
        break;
}
```

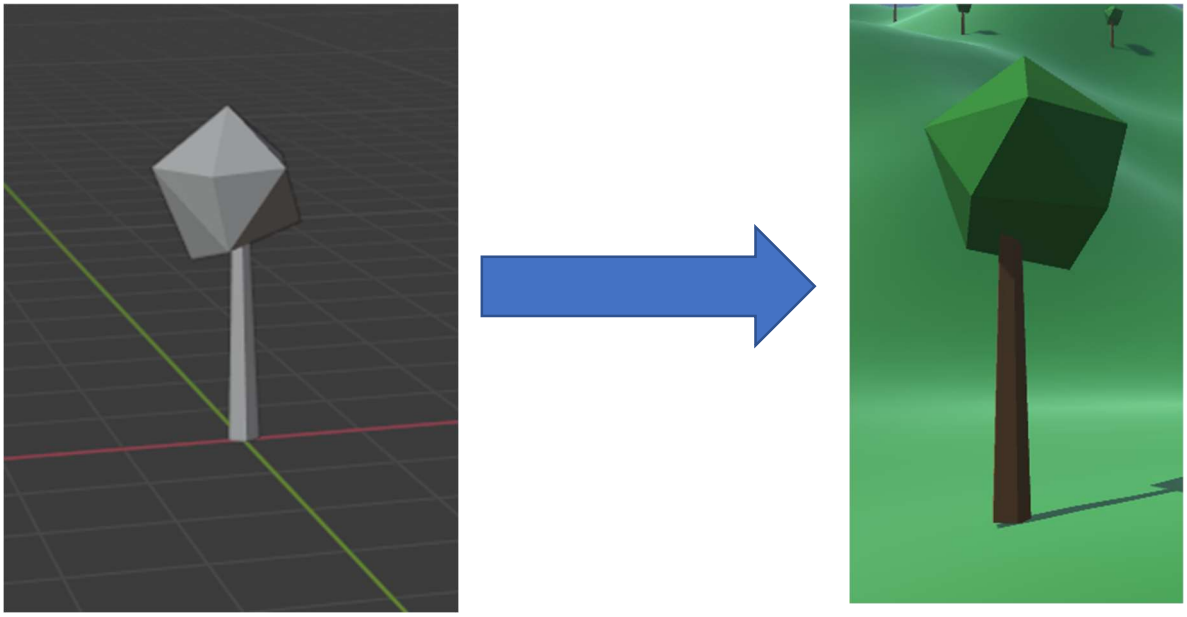
Date: 20/12/22

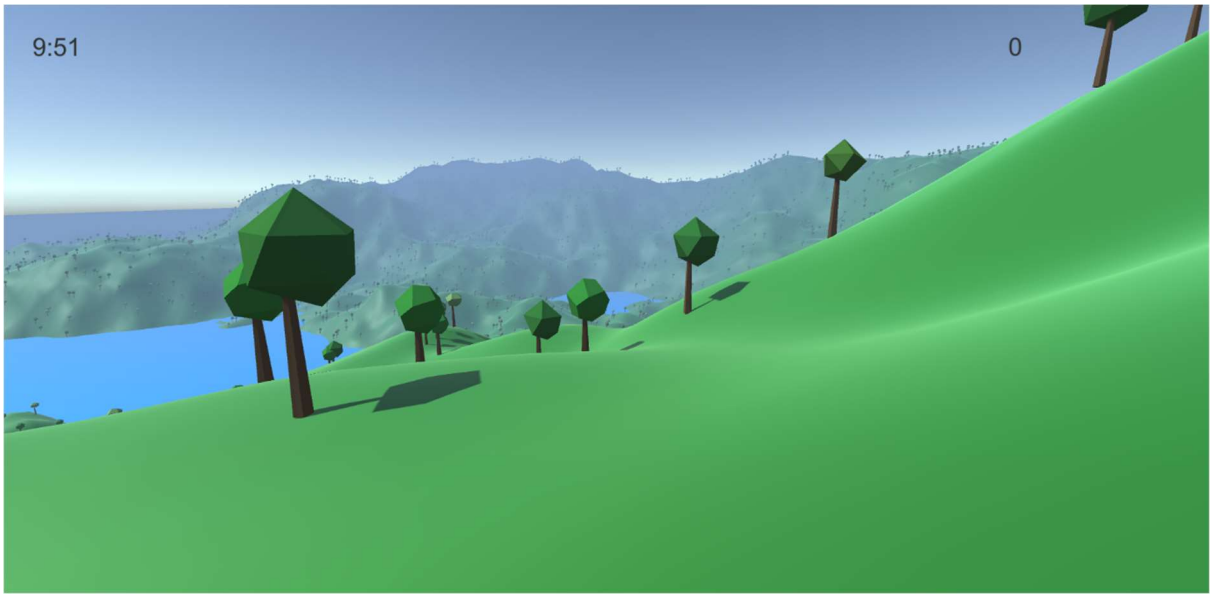
The game was now running quite badly. To help with this, I realised that I could mark the objects which are not moving during runtime as “Static”. I then had to enable static and dynamic batching in my project settings. This should help with optimisation. This was the “Terrain” and “Trees” parent items, along with all their child items. This helped with the performance. I also added “Mesh Compression” to the tree objects and set it to “high”. This also helped slightly with the performance. I felt this was enough optimization for now and can add to this later on in the project. This process can be seen below.



Date: 20/12/22

Here I decided it would be best, in terms of performance and customizability, to stop using the imported package's tree models and just make my own in Blender. So, I made some models that I preferred, and imported them into Unity using the "Import asset" function. I stored them in a new folder called "Blender". With this done, I deleted the package that I previously had installed. I ran into a slight issue, since in Blender, up is the Z-axis, but in Unity, it is the Y-axis. All I had to do was rotate it in blender negative 90 degrees in the X-axis. I made 5 different variants and imported them all in, which took the previous 5 trees' place. I then repeated the optimization techniques as shown earlier. This process can be seen below.





[4] Inventory system

Date: 21/12/22

Here, I needed to implement an inventory system. By moving all picked up items to an “Inventory” parent, I have made this quite easy for myself. The first thing I did was edit the “UI” canvas object within the “Character” object. I created a new child object called “Inventory”. I gave this “Inventory” object a new “Panel” child object. I used my menu “Background” image to fill this object in which a green colour. I then scaled it up, so I filled the centre of the screen. I then created two new text objects, which I can update with the player inventory. Now I just had to make a script to update the second text object with the player’s items. So, I created a new script called “Inventory”. Firstly, I need to go through each child object of the “Inventory” parent and create a string of items. Then, I pushed it to the text object every frame. After referencing the “Inventory” and “Text” objects, I could test this solution. There were lots of repeat items, and I realised that I had to reset the “items” variable every frame. Now this was working. The final part of this was to ensure that the “Inventory” UI component is only visible when the “I” key is being pressed. I had to use the “SetActive” function to toggle the visibility of the panel. I used Unity’s input manager to create a new button called “Inventory” and set it to be “i”. With this done, the inventory screen now popped up when the “i” button is pressed and disappears when it is not. Now test 3 from the design section can be passed. This was a success, and the process can be seen below.

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

public class Inventory : MonoBehaviour
{
    // declare variables
    string items;
    string item;
    string name;
    string[] words;

    // reference gameObjects
    public GameObject panel;

    // reference transforms
    public Transform inventory;

    // reference text
    public Text inventoryText;

    // Update is called once per frame
    void Update()
    {
        // update object visibility
        if (Input.GetButton("Inventory"))
        {
            panel.SetActive(true);
        }
        else
        {
            panel.SetActive(false);
        }

        // reset the "items" variable
        items = null;

        // generate string of items
        foreach (Transform child in inventory)
        {
            name = child.gameObject.name;
            words = name.Split('(');
            item = words[0];
            items += (item + '\n');
        }

        // push to text object
        inventoryText.text = items;
    }
}
```

Here, I declare and instantiate all the variables that I will be using later in the “Update” function.

At this point, I have to reference the “Panel” game object, the “Inventory” transform and the “Text” attribute within the player’s UI.

Here, I check whether “i” is being held down, and if it is the panel is shown. If not, it is hidden.

I have to reset the string every frame.

I go through each child of the “Inventory” parent, convert their name to “Plant” or “Water” and then append it to the “items” string.

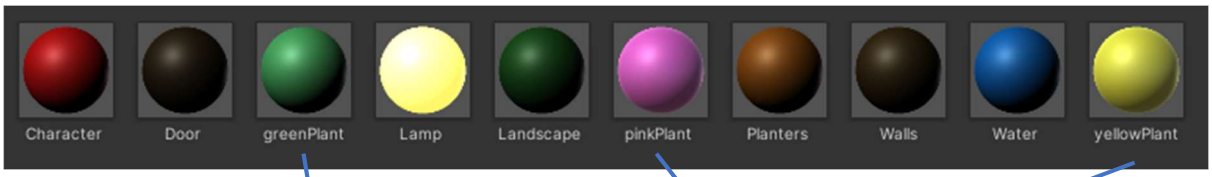
I need to split the name at “(“ since the names are “Plant(Clone)” and “Water(Clone)”.

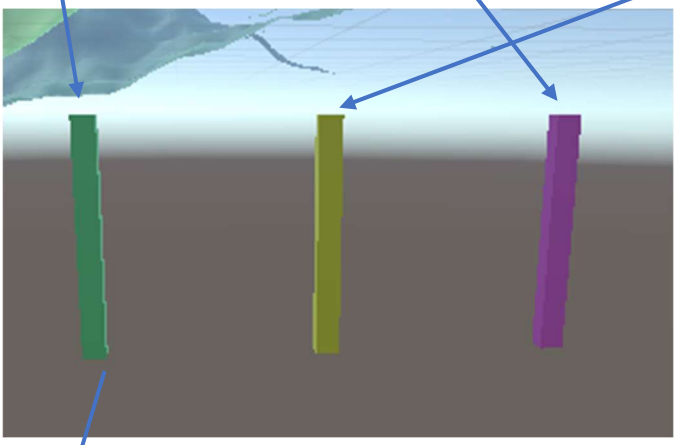
Here, I can finally push it to the text attribute. The final result can be seen here.

[5] Explore mechanic

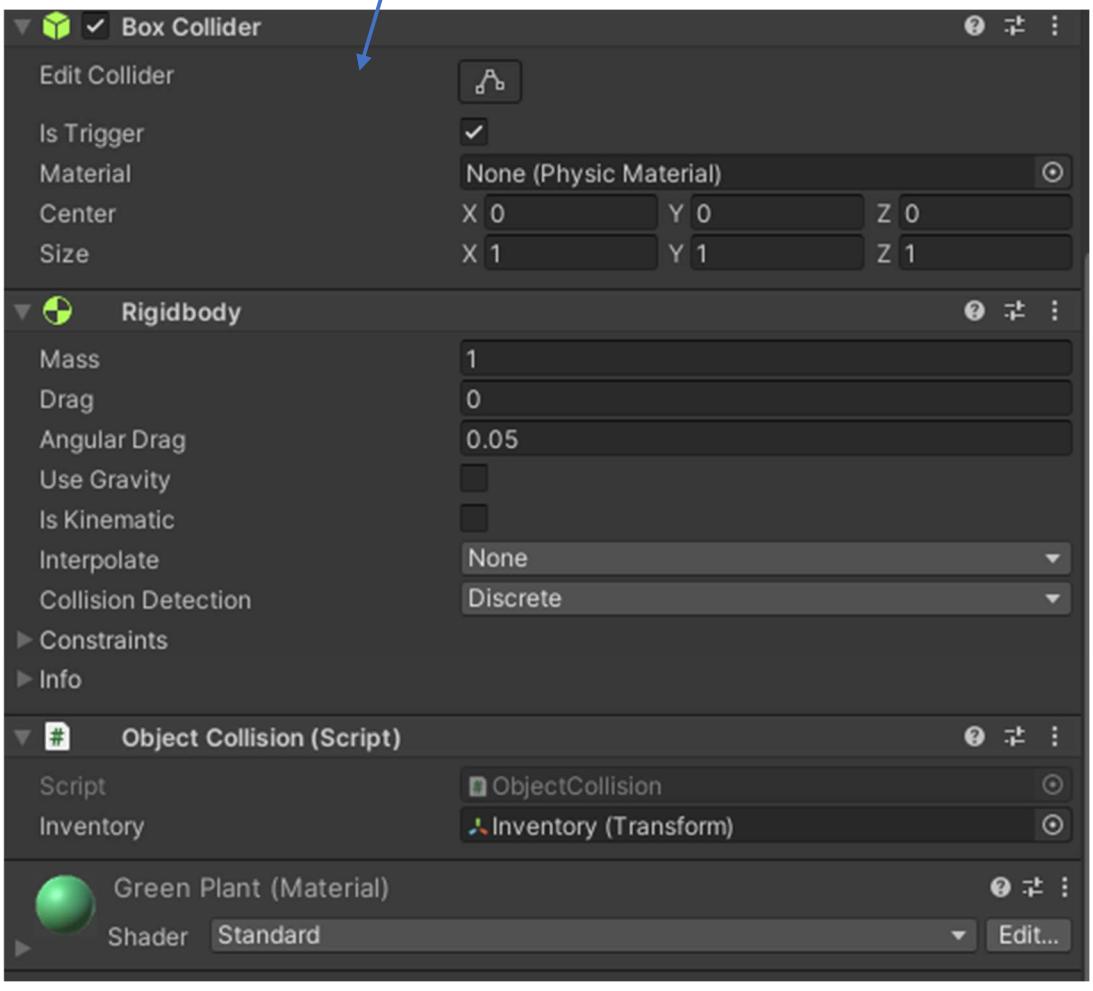
Date: 29/01/23

Here, I need to implement the three different plant rarity types, and their respective effect on the score. The first step here was to create the new models in Unity and apply a coloured material to them. Each plant had to also have a “rigid body” and “objectCollision” script attached to them, so the plant pickup function worked. Their colliders also had to have “is Trigger” checked. Now I had to edit the “TerrainGeneration” script, specifically the “CreateObjects” function to add these extra plants. I ensured that there are four times as many green plants than pink plants, and double as many green plants as yellow plants, to ensure the rarity system from the design section is consistent. This all worked great with the inventory system, except I ran into a problem. Now there are more plants available, the inventory gets filled up quite quickly. Here I decided the best option would be to redo the inventory with a counter instead of a list. Now, all I needed to do was update the score function. Each plant should update the score relative to their rarity. Now this was implemented, I am finished with this section. The process can be seen below as annotated screenshots.





Here, I create and apply colourful shaders to the three different plant rarities.



Here, I ensure that each plant has a rigid body component to ensure the collection method works correctly. I also ensure that each plant has the object collision script attached, along with the reference to the inventory parent.

Here, I implement the new plant rarities into the function that spawns items across the map. I ensure that the rarity of each plant is applied. This also meant that I had to increase the water spawn rate accordingly. Each spawn rate, plants and water, was multiplied by 7.

```
// check number, perform necessary action
switch (num1)
{
    case int n when (n >= 0 && n <= 3):
        // create a green plant object
        Instantiate(plantList[0], position, Quaternion.identity, plants);
        break;
    case int n when (n >= 4 && n <= 5):
        // create a yellow plant object
        Instantiate(plantList[1], position, Quaternion.identity, plants);
        break;
    case 6:
        // create a pink plant object
        Instantiate(plantList[2], position, Quaternion.identity, plants);
        break;
    case int n when (n >= 7 && n <= 13):
        // create a water object
        Instantiate(waterList[0], position, Quaternion.identity, waters);
        break;
    case int n when (n >= 14 && n <= 200):
        // generate a random number between 0 and 4
        int num2 = Random.Range(0, 5);
        // create a tree object
        Instantiate(treeList[num2], position, Quaternion.identity, trees);
        break;
    default:
        break;
}
```

Here I ran into an issue where I collected too many objects, and the list stopped updating and was getting very small. This was the point where I decided to change to a counter-based display due to the increased spawn rates of items.

This page shows the code used to change the inventory from a list to a counter display.

```
// declare variables
string text;
int green;
int yellow;
int pink;
int water;
```

Here I instantiate all the variables I will use later in the script.

```
// Update is called once per frame
void Update()
{
    // update object visibility
    if (Input.GetButton("Inventory"))
    {
        panel.SetActive(true);
    }
    else
    {
        panel.SetActive(false);
    }

    // reset the variables
    green = 0;
    yellow = 0;
    pink = 0;
    water = 0;

    // generate string of items
    foreach (Transform child in inventory)
    {
        switch (child.gameObject.name)
        {
            case "Green plant(Clone)":
                green += 1;
                break;
            case "Yellow plant(Clone)":
                yellow += 1;
                break;
            case "Pink plant(Clone)":
                pink += 1;
                break;
            case "Water(Clone)":
                water += 1;
                break;
        }
```

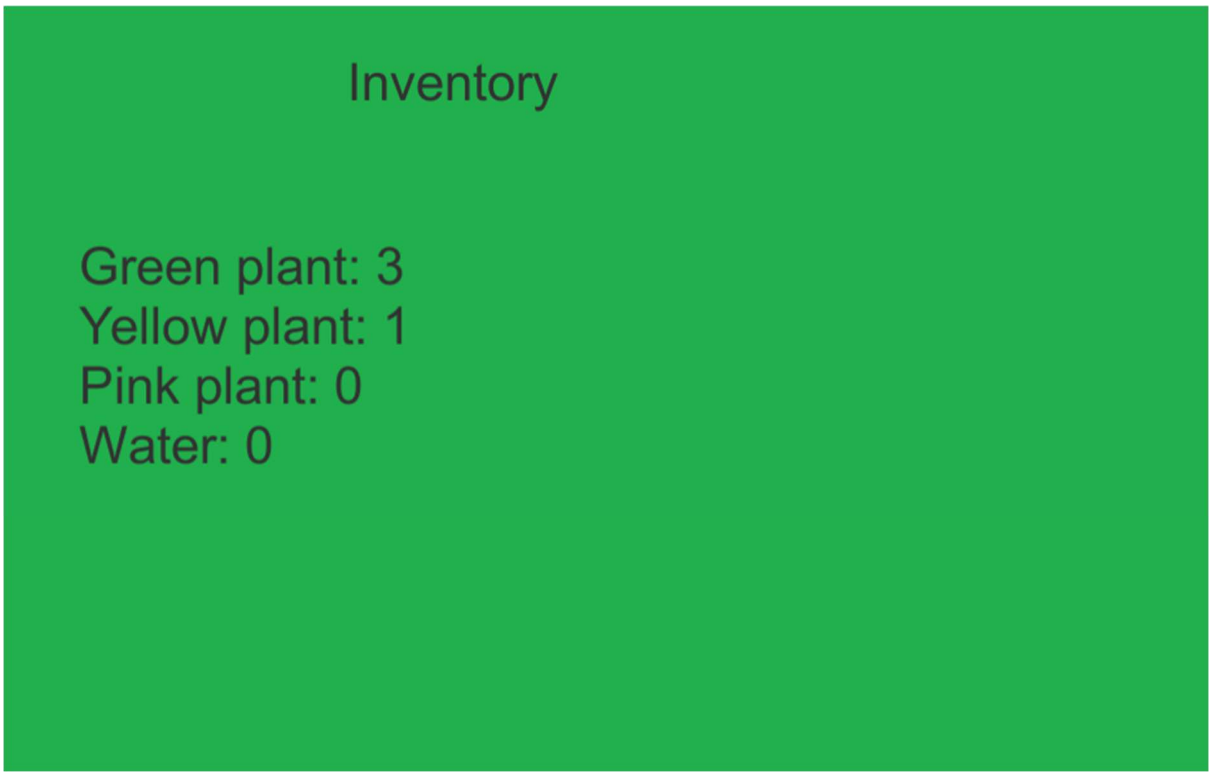
Here, I reset all the variables each frame so that it does not increase to infinity.

Here, I use a switch statement nested within a for loop to get an integer value for the amount of each object collected.

```
// create text variable
text = "Green plant: " + green +
            "\nYellow plant: " + yellow +
            "\nPink plant: " + pink +
            "\nWater: " + water;

// push to text object
inventoryText.text = text;
```

Here, I create a string with all the counter variables in a readable format.

Here I push the new “text” variable to the inventory text object referenced earlier in the script.



This is the result of the counter-based inventory display. Now collecting lots of objects is not an issue.

```
// check for collisions
void OnTriggerEnter(Collider col)
{
    if (col.gameObject.tag == "Player")
    {
        // move the object out of view
        transform.position = new Vector3(-20, -5, 0);
        // change the parent to inventory
        transform.parent = inventory;
        // add respective score
        switch (gameObject.name)
        {
            case "Green plant(Clone)":
                Global.score += 1;
                break;
            case "Yellow plant(Clone)":
                Global.score += 2;
                break;
            case "Pink plant(Clone)":
                Global.score += 4;
                break;
            case "Water(Clone)":
                Global.score += 1;
                break;
        }
    }
}
```

This code will check what plant has been picked up and add to the score variable according to their rarity.

Date: 30/01/23

Now, I need to set up a global “inventory” variable which is updated every frame with a list of its objects. This will then be used in the “Inside” scene to distribute plants and water. This should be quite simple, and I need to edit the “Inventory” script. I created a list that updates every frame, with all the items the player has collected so far. Each frame this is pushed to a variable in the “Global” script called “inventory”. Now I had to quickly create this “inventory” variable. This should work, and I can use this in the next section. This process can be seen below.

```
// declare variables
string text;
int green;
int yellow;
int pink;
int water;

string item;
string name;
string[] words;
List<string> inv;
```

Here, I declare the extra variables that I will need for this section of development, within the “Inventory” script.

```
// reset the variables
inv = new List<string>();
green = 0;
yellow = 0;
pink = 0;
water = 0;
```

I add this section within the for loop, which is iterating through all the items the user has currently collected. It resets the variables each iteration so that it does not count to infinity, but also updates.

```
// add to inventory list
name = child.gameObject.name;
words = name.Split('(');
item = words[0];
inv.Add(item);
```

I add this section to add any item names to the “inv” list.

```
// push items to inventory global
Global.inventory = inv
```

```
public class Global : MonoBehaviour
{
    // variable to store mouse sensitivity
    public static float sensitivity = 1000f;
    // variable to store timer length
    public static float timeRemaining;
    // variable to store player score
    public static int score;
    // variable to store player inventory
    public static List<string> inventory = new List<string>();
}
```

I then set the “inventory” variable within the “Global” script to the value of the “inv” local variable. I had to create this new “inventory” variable within the “Global script”.

[5] Player house

Date: 31/01/23

Here, I am going to set up the “Planters” 2D array as discussed in the design section, so that plant and water objects can be distributed when the player enters their house. Firstly, I edited the object hierarchy, creating a “Planters” parent object where I could attach scripts to. The first thing that I did next was create a new script called “UpdatePlanters” and attached it to the “Planters” parent object. I also needed to create the “Planters” array within the “Global” script, so that player progress is retained. My diagram from the design section helped me to understand the rows and columns of this array. Next, I created the two functions that I designed in the design section to convert between number and string variants of the colours. This will save me time in the future. Now I could work on the ”updatePlants” function. I used my pseudocode from the design section to develop this code. I did find a bug in my pseudocode, which resulted in the “updatePlant” function filling the “Planters” array with plants. I just needed to put a break in once each plant was added. Now I could move on to the “moveObjects” procedure. Again, I followed the pseudocode I developed in the design section. I had to add the “Plants” objects into the “Inside” scene, and reference them to the script. This now seemed to work well, and I could move onto the watering mechanic. Now test 1 from the design section will be passed. This process can be seen below as annotated screenshots.

Here I create the “Planters” parent object, which I can attach the “UpdatePlanters” script to so that it runs every time the player enters the scene.

```
public class Global : MonoBehaviour
{
    // variable to store mouse sensitivity
    public static float sensitivity = 1000f;
    // variable to store timer length
    public static float timeRemaining;
    // variable to store player score
    public static int score;
    // variable to store player inventory
    public static List<string> inventory = new List<string>();
    // variable to store planter information
    public static int[,] planters = new int[16, 2];
}
```

Here I add the 2D array “planters” to the “Global” script.

```
// function to decode number
string intToStr(int num)
{
    // string to store name
    string name = "";
    // switch between input
    switch (num)
    {
        case 1:
            name = "Green plant";
            break;
        case 2:
            name = "Yellow plant";
            break;
        case 4:
            name = "Pink plant";
            break;
    }
    // return decoded number
    return name;
}

// function to encode string
int strToInt(string name)
{
    // int to store num
    int num = 0;
    // switch between input
    switch (name)
    {
        case "Green plant":
            num = 1;
            break;
        case "Yellow plant":
            num = 2;
            break;
        case "Pink plant":
            num = 4;
            break;
    }
    // return encoded string
    return num;
}
```

Here I create two functions that convert between encoded colour values and decoded colour values, and vice versa. This should save me a lot of time during later development within this milestone.

Encoded colour values: “1”, “2”, “4”.

Decoded colour values: “Green plant”, “Yellow plant”, “Pink plant”.

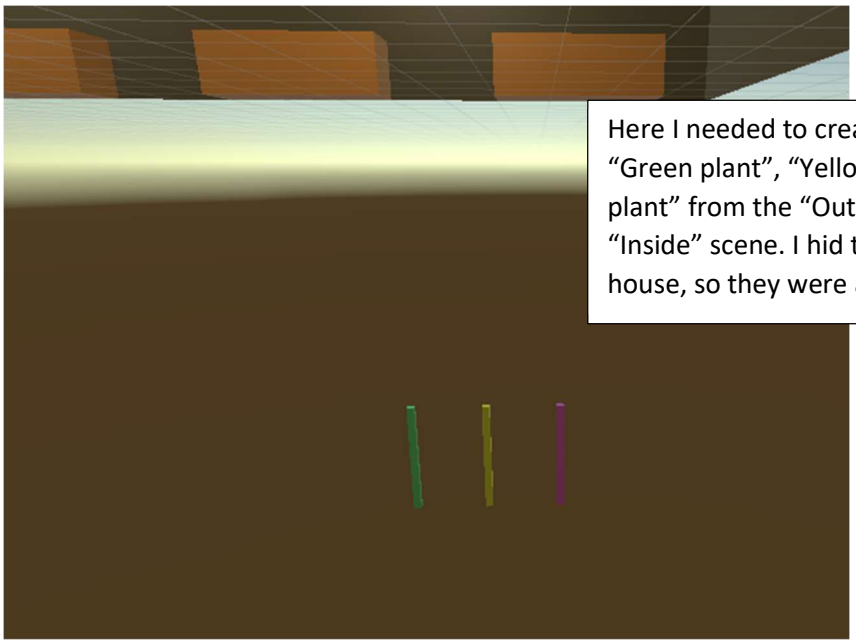
```
void updatePlants()
{
    // calculate inventory length
    int length = Global.inventory.Count;
    // iterate through all the items in inventory
    for (int i = 0; i < length; i++)
    {
        // check if item is a plant
        if (Global.inventory[i].Contains("plant"))
        {
            // iterate through planter slots
            for (int j = 0; j < 16; j++)
            {
                // if there is an empty slot
                if (Global.planters[j, 0] == 0)
                {
                    // set colour
                    Global.planters[j, 0] = strToInt(Global.inventory[i]);
                    // set height
                    Global.planters[j, 1] = 1;
                    // break out of the loop
                    break;
                }
            }
        }
    }
}
```

Here I create the “updatePlants” procedure from my design section. There were very few changes that needed to be made, and it worked well out of the box.

```
void moveObjects()
{
    // set up a variable to store a game object, with default value to avoid error
    GameObject plant = greenPlant;
    // iterate through planter array
    for (int i = 0; i < 16; i++)
    {
        // store plant information in different variables
        int colour = Global.planters[i, 0];
        int height = Global.planters[i, 1];

        // if there is a plant, and the slot is not empty
        if (colour != 0)
        {
            // switch between plant colour
            switch (colour)
            {
                case 1:
                    // set green plant
                    plant = greenPlant;
                    break;

                case 2:
                    // set yellow plant
                    plant = yellowPlant;
                    break;

                case 4:
                    // set pink plant
                    plant = pinkPlant;
                    break;
            }

            // create the position variable
            float xCoord = (float)(3.25 - 5 * (i / 4) - 0.5 * (i % 4));
            float yCoord = (float)(0.5 * height);
            float zCoord = (float)(-4.5);
            Vector3 position = new Vector3(xCoord, yCoord, zCoord);
            // create the new object
            Instantiate(plant, position, Quaternion.identity);
        }
    }
}
```

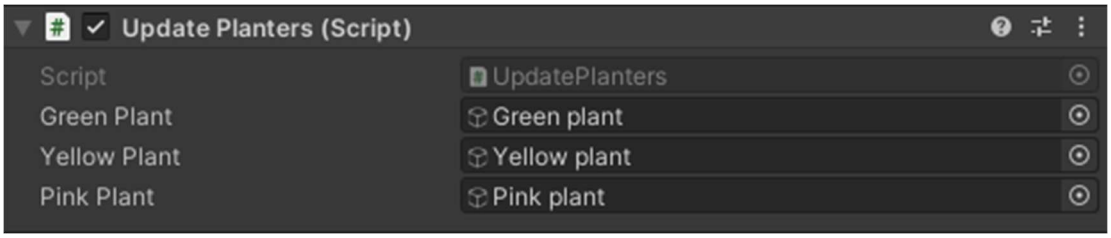
Here I created the “moveObjects” procedure, again from my design section. No changes had to be made, and it flawlessly moved all the objects referenced in the “planters” 2D array to their positions.

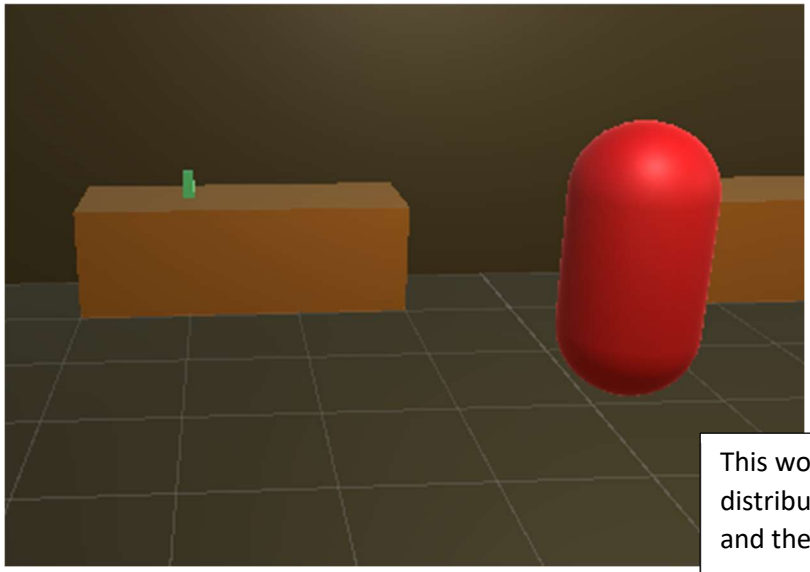
The “xCoord”, “yCoord” and “zCoord” calculations took a bit of trial and error to perfect the positioning of any plant objects.



Here I needed to create copies of the “Green plant”, “Yellow plant”, and “Pink plant” from the “Outside” scene into the “Inside” scene. I hid them under the player house, so they were always out of sight.

I then had to reference them to the “UpdatePlanters” script.





This worked well, and plants were distributed within the “planters” 2D array, and then moved into the planters.

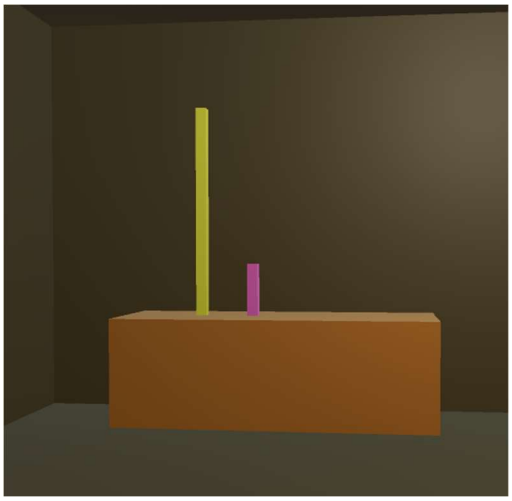
[5] Watering system

Date: 01/02/23

Here, I need to implement the watering system. I covered this in my design section, but in summary plants that are watered need to have their height increased by 1, and any plants not watered need to be removed, since they die. The first step here is to create the new procedure “updateWater”. Now, I could follow my pseudocode from the design section to write the subroutine. My pseudocode functioned flawlessly, which marks the end of this subsection. Now test 2 and 3 from the design section will be passed. The process can be seen below as annotated screenshots.

```
void updateWater()
{
    // calculate amount of water in inventory
    int amount = 0;
    foreach (string item in Global.inventory)
    {
        if (item == "Water")
        {
            amount += 1;
        }
    }
    // iterate through all the plants currently stored
    for (int i = 0; i < 16; i++)
    {
        // check if there is water left and there is a plant
        if (amount > 0 && Global.planters[i, 0] != 0)
        {
            // increase current plant height by 1
            Global.planters[i, 1] += 1;
            // decrement amount value
            amount -= 1;
        }
        // if there is no water remaining and a plant
        else if (amount <= 0 && Global.planters[i, 0] != 0)
        {
            // remove current plant
            Global.planters[i, 0] = 0;
            Global.planters[i, 1] = 0;
        }
    }
}
```

Here, I was able to follow my pseudocode from the design section to create this procedure. It worked well, and there were no problems.



Here, a watered plant of height 3 can be seen when compared to a plant of height 1.

[5] Point system

Date: 02/02/23

Now, I need to implement the point system. I will be referring to points as coins, since I think that will make more sense to the user. The user receives coins if one of their plants grow to a height of 4, or if they try to store plants but their planter is full. To check for fully grown plants, I created a new procedure called “updateCoins”, and followed the pseudocode I developed in the design section. I then had to create a “coins” variable within the “Global” script. Now, I needed to edit the “updatePlants” procedure to reward the player with coins when they bring back excess plants. Again, I was able to follow my pseudocode from the design section here. Now, I needed a place where the player can view the number of coins they have. I used Paint to create a coin icon and removed the background. Then, I had to create a “Canvas” object, and renamed it to “Coins”. Then I could add an image child object, where I imported my coin icon. Then I created a text attribute to store the amount of coins the player has collected. I then created a reference to this text attribute within the script, and pushed the amount of coins to this attribute when the “updateCoins” procedure is called. I then called all the methods from this milestone in the “Start” function, in the same order than I decided in my flowchart from the design section. Now test 4, 5 and 6 from the design section will be passed. This can be seen as annotated screenshots below.

```
void updateCoins()
{
    // iterate through the planters array, checking if height is max
    for (int i = 0; i < 16; i++)
    {
        // if plant is at max height
        if (Global.planters[i, 1] == 4)
        {
            // reward the player with coins
            Global.coins += Global.planters[i, 0] * 100;
            // remove the current plant
            Global.planters[i, 0] = 0;
            Global.planters[i, 1] = 0;
        }
    }
}
```

Here I could follow my pseudocode to create this procedure. This worked well, and any plants of height 4 are automatically converted to their respective coin reward when the “Inside” scene is loaded.

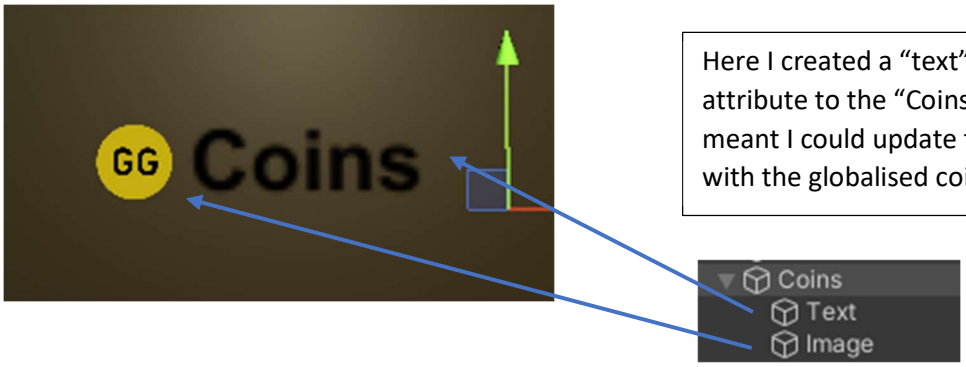
Here I needed to add the “coins” integer variable to the “Global” script.

```
// script to store global variables

public class Global : MonoBehaviour
{
    // variable to store mouse sensitivity
    public static float sensitivity = 1000f;
    // variable to store timer length
    public static float timeRemaining;
    // variable to store player score
    public static int score;
    // variable to store player inventory
    public static List<string> inventory = new List<string>();
    // variable to store planter information
    public static int[,] planters = new int[16, 2];
    // variable to store coins
    public static int coins;
}
```

```
// check if i > 15 and the item is a plant
else if (Global.inventory[i].Contains("plant") && i > 15)
{
    Global.coins += strToInt(Global.inventory[i]);
}
```

Here I add any excess plant objects to the “coins” variable with their respective rarity values.

Here I create the coin logo within paint. I had to then remove the background an import it to Unity as an asset.



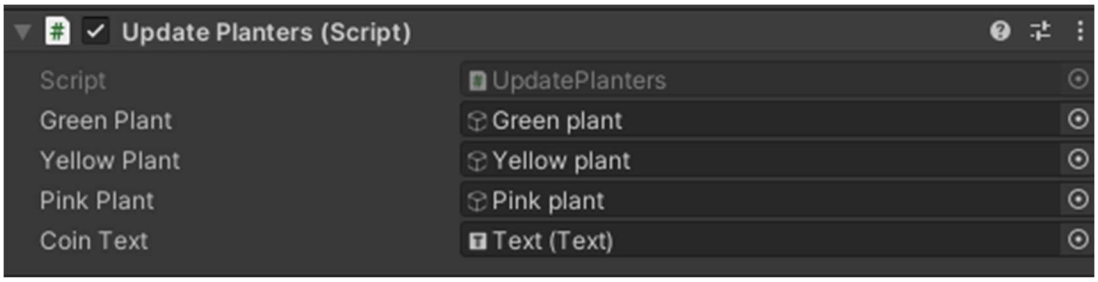
Here I created a “text” and “image” attribute to the “Coins” canvas. This meant I could update the text attribute with the globalised coin value.

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

```
// reference to the score text object
public Text coinText;
```

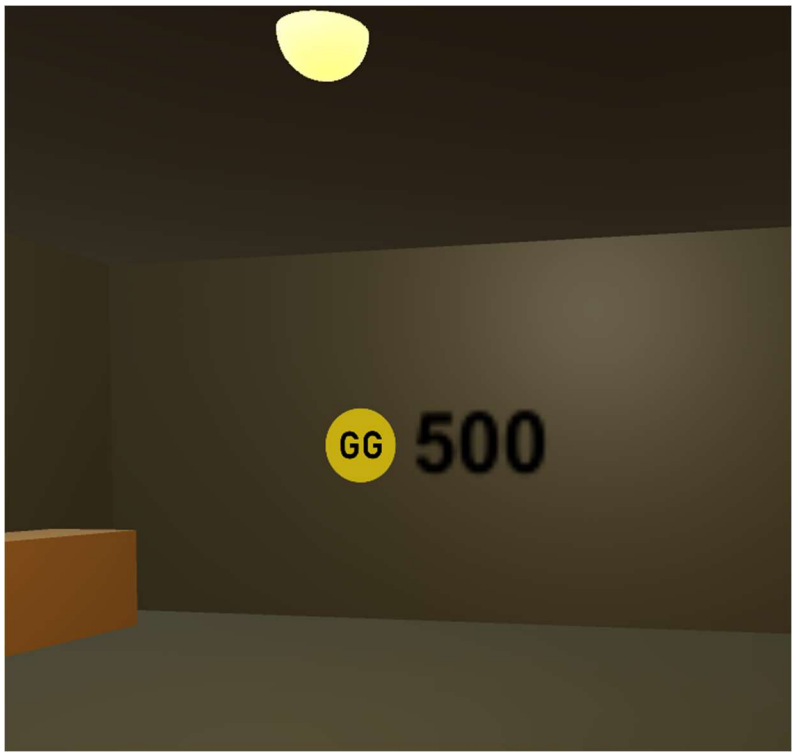
```
// push the coins value to text attribute
coinText.text = Global.coins.ToString();
```

Now, I could edit the “UpdatePlanters” script, adding a reference to the “Coins” text attribute at the start. Then I could edit the “updateCoins” procedure to push the globalised value to the text attribute every time it is called.



```
// Start is called before the first frame update
void Start()
{
    updateWater();
    updatePlants();
    updateCoins();
    moveObjects();
}
```

Now I could edit the “Start” function within the “UpdatePlanters” script to call each function in the correct order individually.

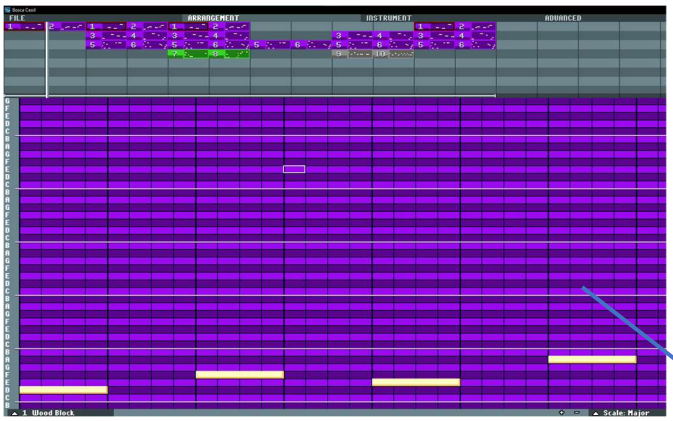


This worked well, and the coins can be collected, and the display is updated.

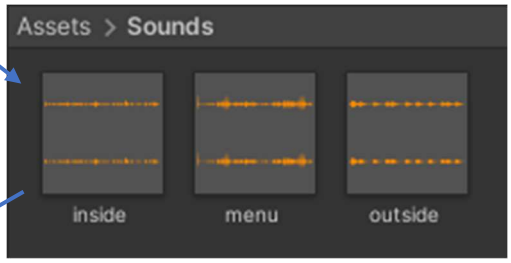
[6] Sound

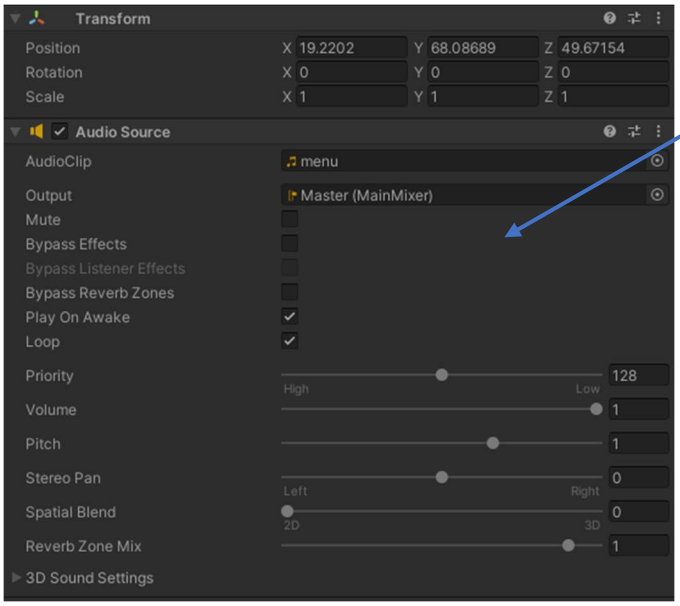
Date: 10/02/23

Here, I would like to put some sound into the game for the user. The best place to start here would be the menu. So, I used the Bosca Ceoil software to create a loop-able track that I could play in the "Start" scene. I created a new "Sounds" folder and imported my "menu.wav" file. I then created a new empty object in the "Start" scene, and called it "Music". I then added the "Audio source" component to this object. I referenced my "menu.wav" file as the input, and the "Master" music mixer that I implemented earlier in development. Then, I selected "Loop". I also did the same process for the "Inside" and "Outside" scene. "inside.wav" would play when the user is inside their house. The music for the "Inside" scene composed of mainly piano notes and chords, so it was calming for the user. Although, I lowered the volume slightly for the "Inside" scene, so it did not overwhelm the user. For the "Outside" scene, I imported "outside.wav" which composed of singular synth noises, since I did not want it to become annoying for the user. I also lowered the volume to 0.2 for this scene to benefit the user. Test 1, 2 and 3 from the design section will be passed now. This worked great, and the process can be seen below as annotated screenshots.



Here, I use Bosca Ceoil to compose the music for my "Start", "Inside" and "Outside" scenes.



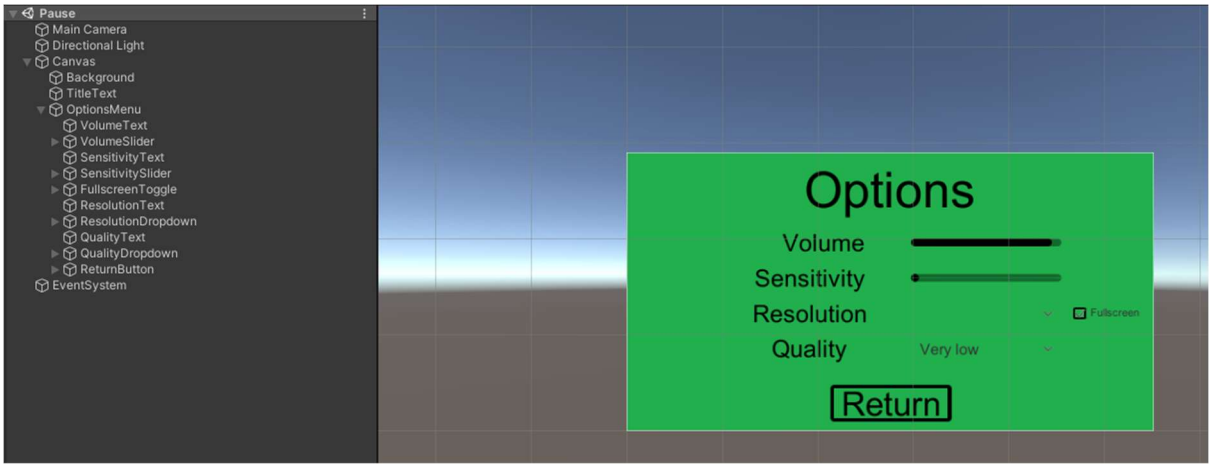


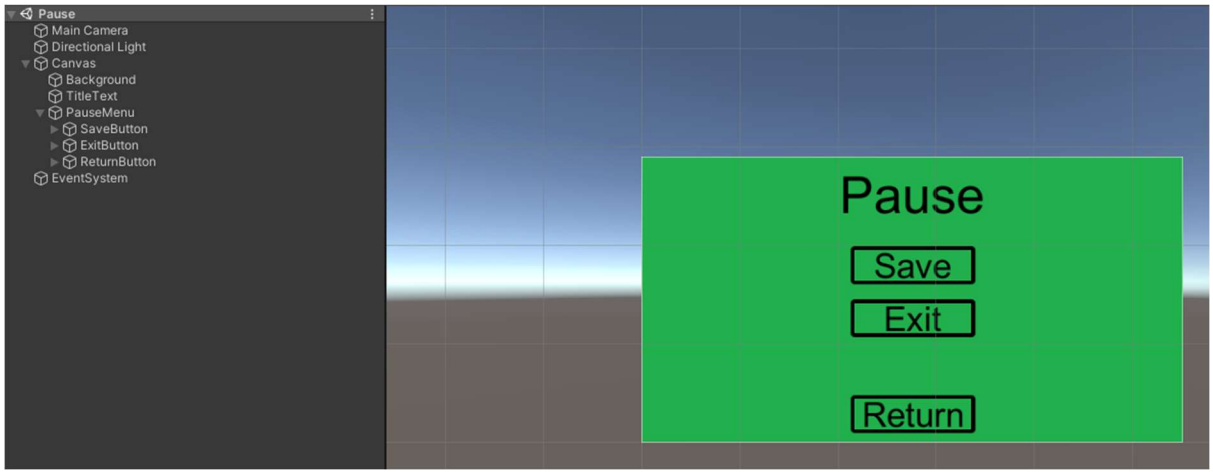
Here, I use the "Audio Source" component attached to an empty object called "Music" in each scene. I selected "Play on Awake" and "Loop" so that the music would play on repeat whilst the player remained in that scene.

[6] Pause menu

Date: 11/02/23

Here, I would like to implement a pause menu. I will follow my sketch up from the design section. The first thing I did here was copy over the options menu from the “Options” scene, to the new “Pause” scene. I renamed the “OptionsMenu” object to “PauseMenu”. From here, I could edit and rename all the objects to match the functionality set out in the design section. Now, I had to create a script to add functionality to the buttons. I used my “OptionsMenu” script as a guide. I called this new script “PauseMenu”. I then added this script to the “PauseMenu” object. With the new “Exit” and “Return” functions implemented in the script, I could reference them through the “OnClick()” functions for each button. Now, the buttons had functionality. For now, I have not added the “Save” procedure. However, there was no way for the player to access this scene. As discussed in the design section, the player will only be able to access this menu from the “Inside” scene. So, I had to edit the “Inside” scene, and add a new empty object called “Pause”. Then, I created a new script, called “Pause”. This script will simply check every frame whether the escape key has been pressed. If so, the “Pause” scene is loaded. I then attached this script to the “Pause” object within the “Inside” scene. Test 4, 5 and 7 should now be passed from the design section.





```
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    // function that runs at beginning
    void Start()
    {
        // ensures that the cursor is unlocked
        Cursor.lockState = CursorLockMode.None;
    }

    // function to exit to main menu
    public void Exit()
    {
        // exit to the start screen
        SceneManager.LoadScene("Start");
    }

    // function to return back to game
    public void Return()
    {
        // return back to the player house
        SceneManager.LoadScene("Inside");
    }
}
```

Here I create the new "PauseMenu" script.

This needs to be ran so that the user can move their mouse around and click buttons.

This will return the user to the main menu. It should be run when the player clicks the "Exit" button.

This will return the user to their game. It should be run when the "Return" button is clicked.

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

public class Pause : MonoBehaviour
{
    // Update is called once per frame
    void Update()
    {
        // update object visibility
        if (Input.GetButton("Cancel"))
        {
            SceneManager.LoadScene("Pause");
        }
    }
}
```

This is the "Pause" script. It is run when the player is in the "Inside" scene. It constantly checks whether the player presses the escape button, and if so the "Pause" scene is loaded. The "Cancel" button is what I have set for escape in the input manager.

Date: 12/02/23

Unfortunately, the implementation of the pause menu also caused a bug that I noticed. When the "Inside" scene is loaded, the "UpdatePlanter" script will run, causing plants to be distributed, water to be applied, etc. This is mostly wanted, however, since the inventory is not being reset when the player enters the pause menu, the user can enter and leave the pause menu and water is applied to plants over and over again. This is certainly unwanted. To fix this, I need the "UpdatePlanter" script to run a certain way if the player just came from the "Outside" scene, and a different way if they came from any other scene. To do this, I instantiated a new variable within my "Global" script called "justExplored". Then, within the "TerrainGeneration" script's "Start" method, which is ran every time the "Outside" scene is loaded, I set the "justExplored" variable to true. Now I can add an if/else statement within the "UpdatePlanters" "Start" method, checking whether the player has just explored. If they had, I set it to false at the end of that branch. The "moveObjects" and "updateCoins" methods are ran anyway since they have no reliance on the player inventory.

```
public class Global : MonoBehaviour
{
    // variable to store mouse sensitivity
    public static float sensitivity = 1000f;
    // variable to store timer length
    public static float timeRemaining;
    // variable to store player score
    public static int score;
    // variable to store player inventory
    public static List<string> inventory = new List<string>();
    // variable to store planter information
    public static int[,] planters = new int[16, 2];
    // variable to store coins
    public static int coins;
    // variable to store whether the player just explored
    public static bool justExplored = false;
}
```

Here, I add the "justExplored" variable to my "Global" script.

```
// Start is called before the first frame update
void Start()
{
    // set justExplored to true
    Global.justExplored = true;
```

Here I set the value of "justExplored" to true within the "Start" function of the "TerrainGeneration" script, which is ran when the "Outside" scene is loaded.

```
// Start is called before the first frame update
void Start()
{
    if (Global.justExplored)
    {
        updateWater();
        updatePlants();
        Global.justExplored = false;
    }
    updateCoins();
    moveObjects();
}
```

Here, I implement the branching into the "Start" method of the "UpdatePlanters" script.

[6] Save game

Date: 12/02/23

Here, I would like to implement the save mechanic. As discussed in the design section, I only need to save the "coins" integer, and "planters" 2D array which are both within the "Global" class. I will be using Unity's "PlayerPrefs" system, in which integers, strings and Booleans can be "set" and "get", then saved. So, to begin, I will allow the player to save their coins since this is the easier to implement. This included creating a "Save" function within the "PauseMenu" script which can be attached to the save button in the pause menu. I set the "Coins" value within the "PlayerPrefs" to whatever value is stored in "Global.coins". Then, I saved any changes using "PlayerPrefs.Save()". This process can be seen below in the form of annotated screenshots.

```
// function to save crucial variables
public void Save()
{
    PlayerPrefs.SetInt("Coins", Global.coins);
    PlayerPrefs.Save();
}
```

Here, I store the player's coin integer value, and save it with "PlayerPrefs".

Date: 13/02/23

Saving the "planters" 2D array is not so simple unfortunately. The "PlayerPrefs" functionality only covers integers, strings and Booleans. So, I will have to convert my 2D array into separate strings to store it. Although this is not a very efficient solution, it is all I have to work with at my understanding level. I used two for loops, iterating through every single data item in the array, then set individual integer values within "PlayerPrefs". Test 6 from the design section should now be passed. This process can be seen below.

```
// procedure to save crucial variables
public void Save()
{
    // set up name variable
    string name = "";
    // store the Global.coins integer
    PlayerPrefs.SetInt("Coins", Global.coins);
    // store the Global.planters 2D array
    for (int i = 0; i < 16; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            name = "Plant[" + i.ToString() + "," + j.ToString() + "]";
            PlayerPrefs.SetInt(name, Global.planters[i, j]);
        }
    }
    // save all the changes to PlayerPrefs
    PlayerPrefs.Save();
}
```

Here I loop through all the planter array's elements, setting a new "PlayerPrefs" integer as the array value. This will result in 32 new saved integers.

[6] Load game

Date: 14/02/23

Here, I need to implement the loading of saved variables. This is available to the user in the "Start" scene, with the load button. So, I created a new "Load" procedure within the "MainMenu" script. Then, I "get" the saved coin value, if any, and open the "Inside" scene. I then had to reference this new "Load" function to the load button within the "Start" scene. This script worked as expected, and the data was saved to the disk. I also realised at this point that I was unnecessarily using indexing for the "MainMenu" script "New" function. I changed this to avoid future confusion. This process can be seen below in the form of annotated screenshots.

```
// function to load any saved variables
public void Load()
{
    if (PlayerPrefs.HasKey("Coins"))
    {
        Global.coins = PlayerPrefs.GetInt("Coins");
        SceneManager.LoadScene("Inside");
    }
}
```

```
// function to run when "New" button is pressed
public void NewGame()
{
    SceneManager.LoadScene("Inside");
}
```

Date: 15/02/23

Here, I need to add the loading feature for the 2D planters array. I just need to do the reverse of the save functionality. Again, this is an inefficient implementation, but the more efficient solutions require much more complex code. Besides this, the implementation worked great and now the user has a complete save and load feature. Test 8 should be passed from the design section. This process can be seen below.

```
// procedure to load any saved variables
public void Load()
{
    // set up name variable
    string name = "";
    // load the Global.coins integer
    Global.coins = PlayerPrefs.GetInt("Coins");
    // load the Global.planters 2D array
    for (int i = 0; i < 16; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            name = "Plant[" + i.ToString() + "," + j.ToString() + "]";
            Global.planters[i, j] = PlayerPrefs.GetInt(name);
        }
    }
    // load the inside scene
    SceneManager.LoadScene("Inside");
}
```

Here, I loop through all the planter array's elements, getting the saved integer and pushing it to the array.

Date: 16/02/23

There was one aspect of this system that I had not considered. If the player starts playing, collects some plants and coins and then exits to the main menu, all the data will still be stored in memory due to the global nature of these variables. Therefore, if they were to click “New”, their previous game data would still be there. Therefore, I need to edit the “New” button’s functionality to wipe the “Global.coins” and “Global.planters” variable. Test 9 from the design section can now be passed. With this working, the save and load feature is fully functional, and the process can be seen below.

```
// procedure to run when "New" button is pressed
public void NewGame()
{
    // wipe any data stored in memory
    Global.planters = new int[16, 2];
    Global.coins = 0;
    SceneManager.LoadScene("Inside");
}
```

Here, I wipe the “planters” and “coins” global variables when the player starts a new game.

[7] Score to coins

Date: 08/03/23

Here, I will remove a previously implemented feature that is overly complex, somewhat futile and buggy. Previously, I have been adding any extra plants collected by the user as coins. The implementation that I used was also quite buggy, since I did not think the logic through properly in my design section. A better implementation would be just adding the score to the player's coin count each time they enter their house. This is for a couple reasons: it is a simpler method, it further rewards the player for water collection, and it adds real function to the score. So, I simply removed the code to add additional plants to the coins counter and added the score instead. Now test 3 from the design section can be passed. This was quite simple, and the process can be seen below as annotated screenshots.

```
// check if i > 15 and the item is a plant
else if (Global.inventory[i].Contains("plant") && i > 15)
{
    Global.coins += strToInt(Global.inventory[i]);
}
```

I removed this code, since I did not account for water being part of the inventory, and was a flawed implementation.

```
// Start is called before the first frame update
void Start()
{
    if (Global.justExplored)
    {
        updateWater();
        updatePlants();
        // set justExplored to false
        Global.justExplored = false;
        // add score to coins
        Global.coins += Global.score;
    }
    updateCoins();
    moveObjects();
}
```

I added this line within the if branch within the "Start" method in "UpdatePlanters" to add the score to the coin variable.

I had to put this line in the if statement since the score should only be added after the player explores, and at no other point.

I could not add it to the "updateCoins" method otherwise I would have to move the "updateCoins" method into the if statement, which would mean the coin text attribute would not update.

[7] Timer fix

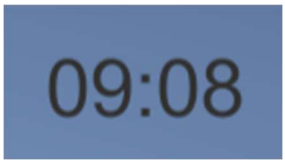
Date: 10/03/23

Here, I will add some if statements to my “Timer” script, to check whether there is a single digit, and to add a leading zero to the front if there is. Whilst I was doing this, I also cleaned my code up a bit by converting the integer values to strings straight away instead of calling the “ToString()” method over and over again. This will increase my code efficiency slightly. Now test 1 from the design section can be passed. This worked well, and the process can be seen below as documented screenshots.

```
// convert values to strings
string minutes_str = minutes.ToString();
string seconds_str = seconds.ToString();
// check if a leading zero is required\
if (minutes_str.Length == 1)
{
    minutes_str = "0" + minutes_str;
}
if (seconds_str.Length == 1)
{
    seconds_str = "0" + seconds_str;
}
// push this to the text object
timerText.text = minutes_str + ":" + seconds_str;
```

Here I convert the “minutes” and “seconds” variables to “minutes_str” and “seconds_str”, from integer to string.

Now I could check the length of the values, and if it is one, I add a leading zero to the string.

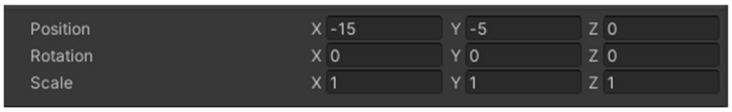


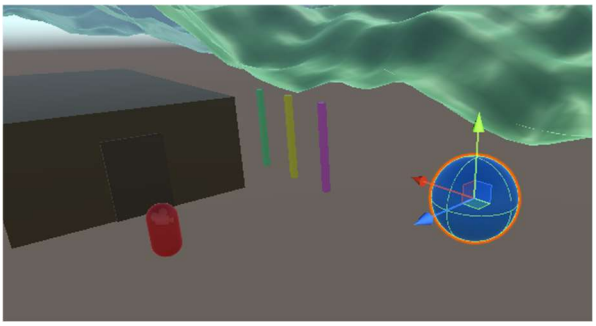
The solution worked well.

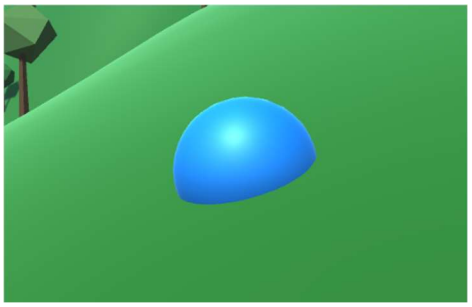
[7] Easier item pickup

Date: 11/03/23

At this point, I had to simply change the radius of water objects from “0.5” to “1”. Test 2 from the design section can be passed. This worked well and can be seen below as documented screenshots.







Here I increase the water object size, and it worked well within the game. It can now be picked up easier.

[7] Game over load

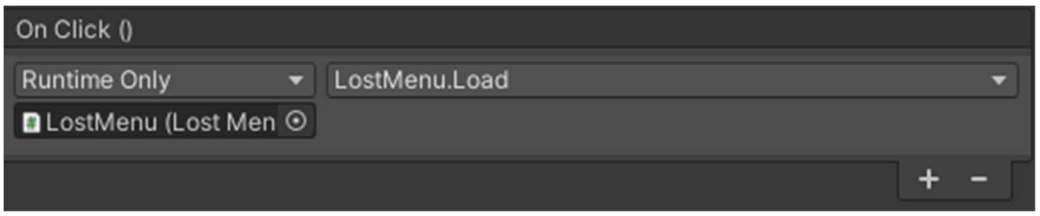
Date: 12/03/23

In this section, I need to edit the “LostMenu” script. I created a new function called “Load”. Then, I edited the “Load” function from the “MainMenu” script and set it as static, so it could be called from other locations. Then, I called this procedure from the “Load” function in “LostMenu” script. Finally, I had to reference this within Unity. Now test 4 from the design section will be passed. This process can be seen below as annotated screenshots.

```
// procedure to load any saved variables
public static void Load()
```

```
public void Load()
{
    // call the load procedure from MainMenu
    MainMenu.Load();
}
```

Here, I set the “Load” function in “MainMenu” to static, call it from the “Load” function in the “LostMenu” script. Then I reference the function from the button editor.



Date: 13/03/23

I ran into an issue when implementing the above solution. When the player loaded their game from the game over screen, their score value was added to the coin value, and the items were distributed. I need to clear the global variables when the game over screen is displayed. The process can be seen below.

```
// function that runs at beginning
void Start()
{
    // ensures that the cursor is unlocked
    Cursor.lockState = CursorLockMode.None;
    // clear inventory and score variables
    Global.inventory = new List<string>();
    Global.score = 0;
}
```

Date: 14/03/23

Unfortunately, I was still missing one line of code for this to work well. I had forgotten about the "justExplored" variable, which will need to be set to false in the game over script, so that the plants aren't updated with an empty inventory when the player loads into their save. This was a quick fix in the "LostMenu" start function, and the process can be seen below.

```
// function that runs at beginning
void Start()
{
    // ensures that the cursor is unlocked
    Cursor.lockState = CursorLockMode.None;
    // clear inventory and score variables
    Global.inventory = new List<string>();
    Global.score = 0;
    // set just explored to false
    Global.justExplored = false;
}
```

Here I set "Global.justExplored" to false to avoid any malfunctions with plant watering when the player loads their save after getting lost.

Evaluation

[1] Testing

Date: 16/09/22

Test number	What is being tested and inputs	Expected outputs	Time stamp	Pass/Fail
1	The player can look up and down by moving their mouse up and down.	The camera rotates up and down. It locks out at a certain distance up and down.	0:04	Pass
2	The player can look left and right by moving their mouse left and right.	The character body rotates left and right, changing direction.	0:18	Pass
3	The player can move forwards and backwards with the “W” and “S” keys.	The character moves forwards and backwards relative to direction.	0:31	Pass
4	The player can move left and right with the “A” and “D” keys.	The character moves left and right relative to direction.	0:42	Pass
5	The player can jump up with the “SPACE” key.	The character moves up into the air, then falls back down due to gravity. The gravity is an appropriate level.	0:54	Pass
6	The player can walk up steps.	The character moves up steps of certain height, by shifting up.	1:02	Pass
7	The player can walk up slopes.	The character walks up slopes of up to 45 degrees.	1:15	Pass

The following link will play the walk-through video of testing this milestone.

https://www.youtube.com/watch?v=dtQ-RL6qMDM

[1] Evaluating

Date: 17/09/22

Overall, I think this milestone can be seen as a success. All the tests from my design section were passed and are clearly demonstrated in the walk-through video. The main take away from this milestone in terms of development is the use of projecting spheres to detect Unity layers. I can reuse this technique for all sorts of applications, such as walking through doors, picking up flowers, and even accessing the upgrades menu.

With more time, I would consider adding a crouch mechanic, and implementing terrain where the user has to crouch under objects. I think this would add some variance to the movement in the game. The stakeholder interview, which can be seen below, gave me more insight into how to improve the game. The main takeaways were to include some sort of mouse sensitivity option, and to reduce the character speed. Whilst developing my next milestone, I hope to implement a mouse sensitivity slider. Perhaps I will reduce the character speed in a later milestone if I have time. My stakeholder also mentioned a control tutorial, which would be great, but I do not have the time to implement such a thing. This could be added in an update or a DLC much later in development.

Stakeholder interview

Does the mouse control feel natural and intuitive?

Yes, although the sensitivity is quite a bit higher than I am personally used to.

When moving the character, does it feel fluid and like you are in control?

Mostly. The movement speed is a little faster than I would like, and in but that's not a massive issue. I think for the purposes of a slower-paced game, a slightly slower movement speed would feel more comfortable.

Are the movement keys confusing, and is a tutorial necessary?

I didn't find them confusing, although for people who aren't as familiar with PC games, a brief tutorial would probably be helpful.

Does the gravity and jumping mechanics feel normal, and is the strength correct?

The jumping and gravity mechanics do feel natural.

When walking up slopes and stairs, do the step heights and hill inclines seem correct?

Yes.

Do you have any further comments or improvements?

It looks great so far.

Success criteria

Number	Criteria	Success	Evaluation
5	Player movement in three dimensions, including the ability to jump.	Fully met	The player is able to move in the X and Y directions. The player is able to jump and has a gravity system to simulate freefalling.
6	Mouse control to move the character camera.	Fully met	The player is able to look around in all directions. The camera locks out at the top and bottom of rotation since the player should not be able to look behind them. Movement left and right simply rotates the character body, so the player can change the character direction.

Improvements

Feature	Time scale	Justification
Crouch	Update or DLC	This feature will be quite time consuming to implement and cannot justify spending a lot of time on a minor feature like this.
Sensitivity slider	Later milestone	I see this as quite an important feature since different people have different sensitivities and need to be able to adjust it. I would like to implement this in a later milestone.
Reduced character speed	None	Whilst this feature is easy to implement, I think it would have a negative effect on the player once I

		implement a large landscape that they have to explore.
Tutorial	Update or DLC	I do not see this feature as very important, and I think most users will be able to understand the controls without a tutorial. This would be time consuming and difficult to implement, so maybe I will add it in an update much further down the development pipeline.

Usability features

Below, I check whether I implemented the proposed usability features from my design section.

Design	Implemented
The use of “W”, “A”, “S”, “D” and Space keys for character movement.	Success

[2] Testing

Date: 30/09/22

Test number	What is being tested and inputs	Expected outputs	Time stamp	Pass/Fail
1	The player can access a start screen when the game is run.	The “Start” scene is loaded.	0:00	Pass
2	The player can load into a new game by clicking the “New” button.	The “Inside” scene is loaded.	0:09	Pass
3	The player can quit the application by clicking the “Quit” button.	The application closes.	0:20	Pass
4	The player can access an options menu by clicking the “Options” button.	The “Options” scene is loaded.	0:31	Pass
5	The player can change the volume by moving the “Volume” slider.	The master game volume changes and persists.	0:41	Null
6	The player can change the mouse sensitivity by moving the “Sensitivity” slider.	The mouse sensitivity changes and persists.	0:51	Fail
7	The player can change the resolution by selecting a resolution from the “Resolution” drop down menu.	The game resolution changes to the selected resolution and persists.	1:02	Pass
8	The player can toggle the game fullscreen or windowed.	The game becomes fullscreen or windowed and persists.	1:24	Pass
9	The player can change the game quality by selecting a quality from the “Quality” dropdown.	The game becomes the quality that the user selected and persists.	1:35	Pass
10	The player can return to the start screen when clicking the “Return” button.	The “Start” scene is loaded.	1:50	Pass

The following link will play the walk-through video of testing this milestone.

https://www.youtube.com/watch?v=Wlo9_8p_6Dg

[2] Evaluating

Date: 02/10/22

When looking at this milestone as a whole, I see this as a success. The menu system is clear, readable and easy to navigate. Most of the tests from the design section are clearly demonstrated in the walk-through video, and pass. However, two of the tests did not pass. The first was the “Volume” slider, which did not pass because there is no evidence of it functioning in the video. I will have to demonstrate its function later in development when the game has sound. Furthermore, the sensitivity slider also had no evidence of functioning in the video. This is because it doesn’t work,

and I was not able to implement it in development. I hope to code a solution to this later in development, since I see it as an important feature.

If I had more time to work on my development, I would like to add other menu screens, such as a player select menu so that multiple people can have saves on one computer. Furthermore, I would like to implement a save and load menu, in which the user can have multiple saves and select which one they would like to load. This would give them more freedom when using the save and load functions. The stakeholder interview was very insightful for this milestone. My stakeholder highlighted the fact that the "Sensitivity" slider does not work. Furthermore, they would like some more colour, or 'life' as they put it, in the menus. However, they do appreciate the simple and easy to navigate nature of my menu structure. The interview can be seen below.

Stakeholder interview

Overall, are the menus easy to understand and use?

Yes.

Are the menus clear and readable?

Yes.

Do all the options function correctly with your machine?

All except the 'sensitivity' slider.

Are the menus aesthetic, and if not, how could I improve them?

I like the minimalist approach, though a little more colour or 'life' wouldn't be unwelcome.

Do you feel like the menu system is simple and easy to navigate?

Yes - some games' menus are easy to get lost in due to too many redundant or unneeded options. I like that this isn't a problem with Gardening Game.

Success criteria

Number	Criteria	Success	Evaluation
1	The opening screen must allow the user to create a new game, load a previous game save or access the settings menu.	Partially met	At this point in development, the user can create a new game and access a settings menu. They cannot load a previous game save since the save and load feature has not been implemented. Therefore, the success criteria was partially met.
3	Settings menu that allows the user to change the resolution and change the game volume.	Fully met	The user can change the game resolution and change the game volume in the options menu. They can also toggle fullscreen and change the game quality. Although I have not been able to get the sensitivity slider to work, I have exceeded the success criteria.

Improvements

Feature	Time scale	Justification
Player select menu	Update or DLC	This would be a great feature to have, especially for users with families, but I do not

		have the time to add it in this development cycle, so I hope to add it in a later DLC.
Save and load menu	Update or DLC	This would be a very time-consuming feature, and I do not think the feature would be worth the time investment.
Sensitivity slider	Later milestone	I see this as an important feature, for reasons discussed previously. I hope to fix this feature in a later milestone, but for now I will move on to adding more essential features.
More colourful menus	Update or DLC	Whilst this is important feedback, it would take my quite a lot of time to find a solution that looks good, but also scales well for all resolutions. Therefore, I would like to add this feature in an update or DLC and spend more time on important features at the moment.

Usability features

Below, I check whether I implemented the proposed usability features from my design section.

Design	Implemented
Big and simple buttons	Success
Black outlining used in the menus	Success

[3] Testing

Date: 20/12/22

Test number	What is being tested and inputs	Expected outputs	Time stamp	Pass/fail
1	The player can see planters inside their house.	Planter objects should be visible.	0:05	Pass
2	The player can leave their house by walking through the door.	The “Outside” scene is loaded.	0:12	Pass
3	A random landscape is loaded when the player leaves their house.	Random landscape is produced.	0:25	Pass
4	The house is positioned correctly on the landscape.	The house sits correctly on the landscape.	0:35	Pass
5	A timer is visible when the player leaves their house.	10:00 can be seen somewhere on the screen.	0:44	Pass
6	The timer is shown in minutes and seconds and is decreasing.	The timer decreases by 1 second every second.	0:53	Pass
7	A gradient is applied to the landscape so that it trails off at the end.	The landscape gradually trails off to sea level.	3:52	Pass
8	A game over screen is displayed when there is no time left on the timer.	The “Game over” scene is loaded.	4:20	Pass

The following link will play the walk-through video of testing this milestone.

https://www.youtube.com/watch?v=GEYKiQ7t3bY

[3] Evaluating

Overall, I see this milestone as a major success. This was quite a large milestone, and the features added are very impactful. Every test that I wrote during the design section was passed and clearly demonstrated during the walk-through video. I learned a lot during this milestone, especially how to move game objects within a script, which I will need to use next milestone. I struggled a lot with the gradient functionality during development, and I am very happy with the end result.

If I were to have more time working on this milestone, I would add a few more features. Firstly, colouring the gradient based on height values would make the landscape much more aesthetic. I could assign a white colour to the highest points on the map, to act as snow. Furthering this, very steep parts of the landscape could be coloured grey or brown instead of green, because in reality grass would not grow in those areas. Another feature I would like to add would be more variant landscapes. Although the landscape is random, it does generally look very similar every time. I would like to add biomes, and I could tweak factors such as flat/mountainous, swampy/dry or even the landscape size itself. The stakeholder interview was very insightful, and outlined some fixes and features I could implement. The first was the fact that the timer does not display the required 0 before the number when it is below 10 seconds. Furthermore, my stakeholder mentioned that a

compass or map feature could be quite helpful when navigating the landscape and reduce the game difficulty a bit. This interview can be seen below.

Stakeholder interview

What do you think about the layout and colouring of the planters within the house?

I think the brown colouring is well-suited to their purpose as planters. It perhaps might make more sense to have them as flatter, wider planters, although I don't see this as a substantial issue.

Are you content with walking through the house door as the method to begin the explore mechanic, and is it intuitive?

Yes. Walking out of the door is very much an appropriate starting mechanic, as it solidifies the spawn point as the player's 'home'.

Is the implementation of the random landscape satisfactory, and does it add to the game's replay value and variance?

Yes, I certainly think it gives the game replay value, as it ensures that you are experiencing a different environment every time. It can at times be difficult to keep track of exactly where you have been, as the landscape all looks quite similar - but this isn't really problematic because it simply raises the skill ceiling of the game in general.

Do you like the use of a sea level, and the implementation of "swimming"? Does it add to the landscape?

Yes - it might make more sense to have lakes at different altitudes, but this isn't something which takes away from the game at all. The 'swimming' mechanic is a good way of making the lakes distinguishable from the rest of the landscape in a functional sense

What do you think about the timer in the top left? Should it remain after I implement a "rolling fog" feature, or would it become futile?

I think it's useful, and I doubt it would be futile if a rolling fog were to be implemented. That said, if things like a compass or map have been purposefully omitted to make the game more 'skill-based' then this might be taken into consideration as well. I also noted that when the second timer goes below ten seconds, it displays the time with no zero before the second digit (e.g., 4:9 instead of 4:09).

Is the landscape big enough? Does the landscape naturally trail off?

Yes, it's definitely big enough - probably closer to too big than too small in fact. I like the way the landscape trails off, really giving the feeling of a secluded island, perfect for a relaxing solo game.

Do you like the layout of the game over screen?

It's fine - nothing special and could maybe be a bit more creative, although it definitely fits with the overall minimalist style of the game. I don't think it's in urgent need of change, but it could probably do with some work at some point.

Success criteria

Number	Criteria	Success	Evaluation
8	10-minute time loop when exploring.	Fully met	When the player leaves their house, a ten minute timer begins, which decreases every second.
12	Generate a random landscape every time the player explores.	Fully met	Every time the player leaves their house, the explore mechanic begins, and the "Outside" scene is loaded. This creates a unique and random landscape with Perlin noise each time
14	Skybox that wraps around the landscape.	Fully met	The default Unity skybox adds a blue colouring all around the landscape.
15	House the player can return to after exploring.	Fully met	The player can leave and return to their house through the door, which transports them between the "Inside" and "Outside" scene.
24	Game over screen that displays when the user gets lost.	Fully met	When the timer depletes to 0, the player is "lost" and is sent to the "Game over" screen.

Improvements

Feature	Time scale	Justification
Height colouring	Update or DLC	Whilst this feature would make the game more realistic and aesthetic, I would rather focus on more important features in the time I have remaining.
Landscape biomes	Update or DLC	Unfortunately, this would be a very time-consuming feature, even if it would add a lot of variation and fun into the game.
Timer fix	Later milestone	This is a very simple fix that I can make, and I will implement it in a later milestone.
Map or compass	Update or DLC	A map could be implemented by creating a 2D map of the Perlin noise function. A compass would be much more complex. Both of these however are too time consuming at this stage of development, and I do not have time to implement them.

Usability features

Below, I check whether I implemented the proposed usability features from my design section.

Design	Implemented
Addition of a timer in the top left of the screen.	Success
Link between fog and time remaining in seconds.	Success
Landscape is automatically generated when the player walks into the door.	Success
Use of a gradient, or any other means of positioning the player safely on the landscape.	Success

[4] Testing

Date: 14/01/23

Test number	What is being tested and inputs	Expected outputs	Time stamp	Pass/fail
1	Tree objects are distributed evenly across the landscape.	Tree objects can be seen across the landscape.	0:10	Pass
2	Plant/water objects are distributed evenly across the landscape.	Plant/water objects can be seen across the landscape.	0:23	Pass
3	An inventory screen can be seen when the “I” button is held down.	An inventory screen can be seen, displaying the player’s current objects.	1:42	Pass
4	The plant/water objects are moved to the player inventory when walked over.	The plant/water object is removed from player sight and moved to their inventory.	1:50	Pass
5	The score is displayed on the player user interface.	The score can be seen in the top right corner of the screen.	2:06	Pass
6	The score increases every time a player collects a plant/water.	The score increments by one.	2:37	Pass

The following link will play the walk-through video of testing this milestone.

https://www.youtube.com/watch?v=x-xfH9tRvys

[4] Evaluating

This milestone can be seen as another success. Every test that I created during the development section was passed and has been clearly demonstrated in the walk-through video. My main takeaway from this milestone is how to duplicate objects, but also how to optimize mass objects in one scene so that the user can have a smooth experience.

With more development time, I would love to add a few more features. The first would be a more interactive inventory screen, that allows users to discard unwanted items, and an inventory limit. This would allow more flexibility for the user and doesn’t force them to carry items that they do not necessarily want. Another feature would be to have a more complex tree distribution algorithm, so that the trees would not spawn on steep parts of the terrain, or perhaps spawn in clumps to act more like a forest. This would create a more realistic environment for the player. My last idea here would be to add powerups across the map, with abilities such as increasing the game timer, or creating a beacon where the player house is. This could extend the explore time and make the game more interesting. My stakeholder was very helpful here too, giving me ideas, such as editing the way items are displayed in the inventory to be a counter instead of a list. They also decided that the plant/water item amount was too low and could certainly be increased. Along with this, they highlighted the fact it is quite hard to pick up items sometimes, and that I should increase the collider size. Finally, they outlined how a key bind system would be helpful for the user and erase any confusion about key bindings. This interview can be seen below.

Stakeholder interview

Is the tree distribution within the landscape suitable? Should there be any more or any less?

I think the random distribution works fine. It could be more realistic if there were greater- and lesser-forested areas, but I don't think this is really a necessary feature in ensuring overall player enjoyment.

Is the plant/water distribution within the landscape suitable? Should there be any more or any less?

I think it would be good to increase the number of plant and water deposits, but not by more than a factor of three or four.

Is the method of picking up plant/water objects satisfactory? Can you think of any better methods for this?

It's satisfactory but could certainly be improved. There were times when I would walk directly over an object (or so I thought) without it being picked up, so perhaps a larger radius would be more suitable.

Does the inventory screen clearly display what items you have picked up? Is the "I" button a suitable key to access this screen, and is it intuitive?

Yes, it does. I'd prefer if it showed a counter (e.g., Plant x3, Water x4), but once again this isn't really problematic - just a matter of preference. The 'I' key is fine for accessing this screen, although maybe a key-binding system could be useful to mitigate the risk of people not finding it to be intuitive (again - this really isn't necessary, just an idea for general future QOL improvements).

Does the score system work correctly, and is it easy to see and understand?

As far as I can tell, it works fine and makes perfect sense. I haven't had any problems with the score system.

Success criteria

Number	Criteria	Success	Evaluation
7	System that allows the user to explore the landscape and collect plants.	Fully met	The player can walk over any plant/water object, and it will be moved out of sight and into their inventory. This gives the impression that they physically collected the plant/water object and is realistic.
9	Display the current score on the user's screen when they are exploring.	Fully met	The player's score can be seen as a number on the top right of their screen. This is easy and convenient for the user.
11	Player increases score every time they collect a plant when exploring.	Fully met	Every time a player collects a plant, the score increases by 1. This is exactly what I set out to do.
20	Inventory system that allows the user to see what plants they have collected on their journey and the rarity, and discard any they don't want.	Partially met	The inventory screen currently just displays the plants the user has collected. They cannot discard any items or see their rarity, and for that reason I have partially met these criteria.

Improvements

Feature	Time scale	Justification
Interactive inventory	Update or DLC	This feature would definitely benefit the user; however it will be quite complex and time consuming. For this reason, I would add this in a later update.
Complex tree distribution	Update or DLC	An algorithm like this would be quite hard to create. When comparing the time required to develop, and what it adds to the end game, I cannot justify spending my time on this feature for now.
Powerups	Update or DLC	This would be an amazing feature to add and could increase the explore length and skill ceiling. Unfortunately, it would take a lot of time to implement and would be better fit for an update or DLC.
Easier item pickup	Later milestone	Even though I am changing the models for plant objects next milestone, this feedback will still be relevant for any water objects. This is an easy fix, which includes just increasing the hitbox size of water objects.
Increased plant/water	Later milestone	This problem came to my attention when I was testing the game myself. This is an easy fix, and I can just increase the probability of a plant/water object spawning.
Inventory counter	Later milestone	This would be a very simple change to make. Especially later in development, when there are multiple rarity plants and more item pickups, this would be helpful to the user. I hope to implement this in a later milestone.
Key-bind system	Update or DLC	This addition would greatly benefit the user, however a feature like this is far beyond my understanding and very complex. For this reason, I will implement it when I have developed my knowledge and in a later update.

Usability features

Below, I check whether I implemented the proposed usability features from my design section.

Design	Implemented
Implementation of an inventory screen.	Success
Use of clear, black text on a green backdrop for the menu.	Success
Resizing the text to best fit in buttons and text boxes.	Success
Score counter in the top right of the screen.	Success
Ability to just walk over a plant or water object to pick it up.	Success
Items automatically moved into player inventory.	Success
Tree objects have no collider attached to them.	Success

[5] Testing

Date: 03/02/23

Test number	What is being tested and inputs	Expected outputs	Time stamp	Pass/fail
1	The plants the player has collected are distributed into the house planters.	Plants are put into the planters, at height of 1, from left to right. There should be 4 plants per planter.	1:12	Pass
2	The water the player has collected is used to increase plant height.	Plants are watered from left to right and will increase by a height of 1.	2:34	Pass
3	The plants which are not watered die.	Plants that are not watered should be removed from the house planters.	3:24	Pass
4	The plants which reach a maximum height are converted to coins.	Green plants generate 100 coins, yellow 200 coins and pink 400 coins. If they are converted to coins, they are removed from the house planters.	4:13	Pass
5	Any excess plants collected are converted to coins.	Green plants generate 1 coin, yellow 2 coins and pink 4 coins if they are excess. They should not be planted inside.	11:31	Fail
6	The player's coin value is displayed and updated on the house wall.	The globalised integer value should be displayed on the player wall.	11:42	Pass

The following link will play the walk-through video of testing this milestone.

https://www.youtube.com/watch?v=enPA3p3uqzY

[5] Evaluating

This milestone can be seen as another success. Most of the tests that I set out in the design section were passed in the walk-through video. However, one test failed due to a bad implementation, and poorly thought-out pseudocode. I will hopefully find a way around this issue in the next few milestones. Other than this, I learned a lot about global variables and arrays in this milestone and hope to take this knowledge into the next final milestones.

With more development time, the first feature I would work on is fixing the excess plant to coin function. However, I think there is a better way to implement a feature like this. Instead of converting excess plants to coins, I will just add the score value on to the coin value every time the player returns from exploring. This is a much simpler solution and adds purpose to the score mechanic. Along with this, I would like to build upon the coin functionality, such as having upgrades or an end goal to purchase, as outlined in my design section. My stakeholder was a great help in this section, as they reassured my tests and were happy with all the implemented features. They highlighted the fact that the water object pickup was still quite finnicky sometimes, and I will fix this in a future milestone. The interview can be seen below.

Stakeholder interview

Are you satisfied with the implementation of plants being distributed from player inventory to the house planters?

Yes. It is simple and effective.

Do your plants get watered properly? Do they die when they are not watered?

Yes. I haven't had issues with plants incorrectly dying/not dying.

Are the 3 different plant rarities, and their relative coin rewards, relevant and suitable?

Yes. I enjoy the variety that this mechanic brings.

Do you like the implementation of the coin system?

I do. I like that it provides a measure of development alongside the basic plant collection and maintenance objective.

Now that there is increased plant and water object spawn rates, are you satisfied with the number of objects across the terrain?

Yes. I no longer find that I wander for minutes without finding an object. I would still say that the pickup mechanic could do with some tweaking; it still feels quite clunky.

Success criteria

Number	Criteria	Success	Evaluation
13	Generate random plants with different rarities every time the player explores.	Partially met	There are 3 different plant rarities implemented in this milestone. However, they are not randomly generated, so this criterion is partially met.
16	System that ages the plants every time the user leaves their house.	Fully met	Every time the player leaves their house, their plants are either watered or die, depending on the amount of water the player brings back.
17	System that simulates plant water intake.	Fully met	A plant's height is increased by 1 if it is watered. This shows growth and uses up water every time.
21	System that allows players to convert grown plants into points.	Fully met	This is done automatically for the user, when a plant reaches a height of 4, it is removed and converted to its respective coin value.

Improvements

Feature	Time scale	Justification
Score to points	Later milestone	This will be a fix for the excess plant feature not working. I think this is a better solution anyway and gives purpose to the score function. It should be relatively easy to implement, so it will be implemented in a later milestone.
Upgrades and end goal	Update or DLC	Whilst this feature would add a lot to the game loop and replay-ability of the game, it take a lot of time and

		learning for me to implement a big feature like this. Therefore, I will implement this in an update or DLC.
Easier item pickup	Later milestone	This has been suggested a few times now, and I will increase the size of the water objects so that they are easier to pick up in a later milestone.

Usability features

Below, I check whether I implemented the proposed usability features from my design section.

Design	Implemented
Displaying the number of coins collected on the house wall.	Success
All the water they have collected is used up watering their plants.	Success
All the plants they have collected is automatically distributed into the planters.	Success
Any grown plants are automatically converted into coins.	Success

[6] Testing

Date: 17/02/23

Test number	What is being tested and inputs	Expected outputs	Time stamp	Pass/fail
1	The menu music is played in a loop when the player is in the “Start” scene.	Menu music is played in loop.	0:02	Pass
2	The inside music is played in a loop when the player is in the “Inside” scene.	Inside music is played in loop.	1:14	Pass
3	The outside music is played in a loop when the player is in the “Outside” scene.	Outside music is played in loop.	1:55	Pass
4	The pause menu can be accessed by pressing the “escape” key.	Pause menu is loaded.	3:00	Pass
5	The player can return back to their game by clicking “return” in the pause menu.	Player is returned to game.	3:11	Pass
6	The player can save their game by clicking “save” in the pause menu.	Essential variables are saved.	3:29	Pass
7	The player can return to the opening screen by clicking “exit” in the pause menu.	Start screen is loaded.	3:39	Pass
8	The player can load their game by clicking “load” in the main menu.	Game save is loaded.	3:49	Pass
9	The player can start a new game by clicking “new” in the main menu.	New game is loaded.	4:00	Pass

The following link will play the walk-through video of testing this milestone.

https://www.youtube.com/watch?v=5mWoEOtwM_I

[6] Evaluating

Overall, this milestone can be seen as a major success. Every test that I designed previously was passed during the walk-through video, and all the implemented features work great. I learned a lot about playing music in Unity and was aided a lot by the fact that I had already implemented a music mixer in an earlier milestone. Furthermore, I learned a lot about save and load solutions, and hope to use this knowledge in the future.

If I had access to more development time, I would like to implement a few more features. Firstly, I would allow the player to access a pause menu even when they are exploring, ensuring that the save function also stores player location and landscape details. This gives the player more freedom when playing the game. Furthermore, I would like to use binary files to store the player data instead of the “PlayerPrefs” class which stores the data in respective registries on the host computer. Whilst this is harder to achieve, it would be a more secure storage of data, and it would ensure that users cannot edit their save data. My stakeholder was a big help here, highlighting the fact that the “Load” button does not work from the game over screen. This is a button that I had forgotten about and did not test. This will be fixed in a later milestone. The interview can be seen below.

Stakeholder interview

Does the music implemented in the main menu, house and outside fit the game theme and feel?

Yes. Each soundtrack feels very well suited to the activity in question.

Are you able to access the pause menu from within the player house by pressing the "Escape" button? Is this key intuitive and easy to remember?

Yes.

Is the menu system fluid and intuitive? Is it easy to navigate?

Yes.

Are you able to save your game easily and consistently?

Yes.

Are you able to load your saved game?

Mostly, however I did notice that the 'Load' button does not work when it appears on the 'Game Over' screen.

Are you able to start a new game, even with a previous game save?

Yes.

Success criteria

Number	Criteria	Success	Evaluation
2	The save/load menu must allow users to save their current game or load their past save.	Fully met	Players can save their game from the pause menu and load their game from the start menu. This criterion has been fully met.
4	Pause menu that the user can access when playing the game.	Partially met	Players can access a pause menu by pressing "escape" when inside their house. Therefore, this has been partially met since they cannot access this screen when exploring outside.

Improvements

Feature	Time scale	Justification
Accessible pause menu	Update or DLC	Whilst this feature aids the game usability, the time it would take to edit the save function and the extra storage it would require due to a larger save file makes this feature quite a low priority. Therefore, it would be implemented in a future update or DLC.
Binary save file	Update or DLC	This is quite an important feature, but it would require a lot of research and further C# and Unity understanding, which I am currently incapable of. Therefore, I would have to introduce this change in a later update or DLC.

Game over load button	Later milestone	This is a previously added button, that I have yet to add functionality to. This needs to be fixed which will be quite easy and will be added in a later milestone.

Usability features

Below, I check whether I implemented the proposed usability features from my design section.

Design	Implemented
Implementation of a pause menu.	Success
Simple menu implementation of the save and load functions.	Success
Save and load functions.	Success

[7] Testing

Date: 16/03/23

Test number	What is being tested and inputs	Expected outputs	Time stamp	Pass/fail
1	The timer populates single digits in the minutes or seconds with a 0 prefix.	A zero can be seen before any single digits.	0:15	Pass
2	Water objects can be easily and reliably collected when walked over.	The item can be picked up in a single action, and multiple attempts are not required.	0:36	Pass
3	The player's score value is added to the coin value every time they return.	The coin value is updated with the addition of the player's score.	1:03	Pass
4	The player's save file can be loaded from the game over screen.	Any saved data values are loaded, and the scene is changed to "Inside" when the load button is pressed.	11:29	Pass

The following link will play the walk-through video of testing this milestone.

https://www.youtube.com/watch?v=d9RLXpxcMug

[7] Evaluating

Date: 16/03/23

This milestone can be seen as a major success, and all the fixes that I planned to implement work and function correctly. All the tests that I constructed during the design section were passed and displayed during the walk-through video. I was able to use the knowledge that I had picked up in previous milestones and apply it to issues highlighted in earlier milestones.

With more development time, I would try to fix other issues or implement more suggested features. However, if I had to think of some new features here, I would change the models for plant and water objects, perhaps using actual flower models for plant objects, and a water bucket object for water objects. This would require a lot of work, and I would need to get better at Blender. I would also like to move all the tree objects down a few units, since sometimes when they are on a slope the player can see beneath the trunk. Finally, when distributing trees, I would give them random rotational values to make the environment look more unique and random. My stakeholder was very happy with this milestone. They found that the fixes implemented aided usability, and they were unable to think of any further improvements to the game.

Stakeholder interview

Are you happy with the fixes and implemented features within this section?

Yes.

Does the new water object help with picking up items?

Yes, I haven't had any further problems with this since this update.

Are there any further issues or features that you would like to suggest?

No – I think the game works well as it is.

Improvements

Feature	Time scale	Justification
Plant/water models	Update or DLC	Whilst this feature would make the game more realistic, it would require a lot of work learning 3D modelling, and there is very little time left until release, so this would have to be contained in an update or DLC.
Better tree distribution	Update or DLC	This feature would not actually be very difficult to implement, but due to the very little time left until release, this would have to be included in a later update or DLC.

Usability features

Below, I check whether I implemented the proposed usability features from my design section.

Design	Implemented
Adding functionality to the “Load” button.	Success
Timer fix which adds a zero onto any single digits.	Success
Easier item pickup fix lets the user consistently and reliably pick up water objects.	Success
Score to points fix adding more reason to the score mechanism.	Success

Post development testing

Date: 20/03/23

Post development test	Testing to be performed	Time stamp	Pass/fail
The menu system is intuitive and easy to navigate.	The menu is a closed system and allows users to navigate back and forth between scenes.	0:03	Pass
The music is suitable and not overwhelming.	The music does not become annoying and is not too loud in scenes.	0:34	Pass
When exploring, landscape is randomly generated, with trees, plants and water distributed across it.	Landscape is different each time, with new plants and water objects.	1:11	Pass
Plants and water objects can be collected when exploring.	Objects are picked up when the player walks over them.	1:33	Pass
Plants can be grown with collected water, and coins are rewarded.	Plants increase in height when watered and are converted to coins if they reach maximum height.	2:18	Pass
Score is converted to coins each time the player returns.	The score value is added to the total coin value every time the player returns to their house.	4:23	Pass
The planter information and coin value can be saved locally on the host computer.	Data can be saved by clicking the save button.	4:35	Pass
Game saves can be loaded from local files.	Data can be loaded by clicking the load button.	4:49	Pass

The following link will play the walk-through video of testing this milestone.

https://www.youtube.com/watch?v=SWJISkKMRJQ&t=77s

Beta testing

At this point in development, my stakeholder played the game and confirmed my testing with his own testing. The game worked well, and my stakeholder was happy.

Improvement summary

Here I discuss the improvements and fixes that I was able to implement during development.

Feature	Suggested	Implemented	Explanation
Sensitivity slider	Milestone 1 Milestone 2	Milestone 2 Milestone 3	In Milestone 2, I implemented the sensitivity slider, but I could not get it to work. In Milestone 3, I thought of a new method using a "Global" script to store the variable, and it worked well.
Timer fix	Milestone 3	Milestone 7	In Milestone 7 I check whether the minutes or second value consists of one digit and add a leading zero if that is the case.
Easier item pickup	Milestone 4 Milestone 6	Milestone 5 Milestone 7	In Milestone 5, I changed the plant model to a square prism shape, which is much easier to pick up for the user. The water objects are still somewhat difficult to pick up. In Milestone 7 I double the radius of the water objects, so they are easier to be picked up.
Increased plant/water	Milestone 4	Milestone 5	In Milestone 5, I added more plant rarities, which in turn caused me to multiply the plant/water spawn rate by around 7.
Inventory counter	Milestone 4	Milestone 5	In Milestone 5, I had to change the inventory system to a counter-based display, since with the increased spawn rates, there were too many plants to fit on the screen.
Score to points	Milestone 5	Milestone 7	In Milestone 7, I implemented this change, removing the old code which converted excess plants into coins. Instead, I add the score value to the coin value each time the player returns to their house.
Game over load	Milestone 6	Milestone 7	In Milestone 7 I add functionality to the "Load" button in the game over screen, reusing the "Load" function from the "MainMenu" script.

Maintenance

Unmet success criteria

Number	Criteria	Success	Evaluation
10	A high score menu that the player can view to see previous top scores.	Not met	I did not implement this during development. The reasoning for this was due to a lack of importance, and a focus on other features. I would like to implement this in a later update or DLC. To implement this, I would have to save the player's score to the save file and change the value whenever they score higher than their highest score.
18	Map that the user can access while exploring	Not met	This feature was never implemented, due to the complexity of this suggestion and how it completely overweighs the usability it would add to the game. Perhaps I will add this in a future update or DLC. To implement this feature, I would use a top-down camera on the landscape, with a pixelating filter to give an abstracted view to the user.
19	Animals that randomly roam the landscape.	Not met	I planned to add this feature if I had surplus time at the end of development, which I did not. Therefore, I would like to add this feature to add more game atmosphere in a future update or DLC. I would need to use Unity's "NavMeshAgent" and pathfinding algorithms to implement this feature.
22	Upgrade system that allows players spend points to upgrade their tools for faster use times.	Not met	This feature would really aid the replay-ability of my game; however, this would take a huge amount of time to implement, which I do not have access to. Therefore, I would like to add this feature in a future update or DLC. This would require a new menu system, and further data saved to the host media.
23	An end goal that the player can work towards, perhaps bought with points.	Not met	This is quite an important feature, and gives the player something to work towards, however I was not able to think of a suitable goal during this development cycle, so would like to implement this in a future update or DLC. This would require further interfaces and menus, with a purchasable end goal.

Further usability features

Here I discuss any usability features that I would develop or develop further if I had more time.

Feature	Evaluation
Gradient surrounding player house	Whilst I did find an alternative to this during development, in which I just adjust the height of the house to the height of the landscape, creating a gradient around the user would give a more natural and easier to use environment for the player. This makes it easier for the user to access their house.
Game tutorial	This is a very important feature that I would include if I had more time. As it stands, there is no information for the player on how the game works, and they are left to work it out on their own. To increase playability and usability I would include a new menu screen which outlines all the core game mechanics.
Control menu	This is important for new gamers, since they might not understand what keys are normally used for certain functions within games. I would include a new menu screen outlining all the keymaps. This could be taken further and developed into a key binding menu.
Text to speech setting	This is important for users who have vision problems. Perhaps I could implement a feature which allows for compatibility with screen readers. The software could read out what the character is currently looking at, opening up the player base to the visually impaired, and increasing usability.
Colour blind setting	Due to the nature of my game, a lot of the menus and visuals are green. Therefore, any users who are colourblind may have trouble playing. So, I would like to include a feature in the settings in which you can select different colour-blind modes, which in turn would change the colourings of the game. This increases usability for users with colour blindness.

Overcoming limitations

Here I discuss the limitations that I outlined during the analysis section, and how I would overcome them if I had more development time.

Feature	Evaluation
Complex object models	To implement this feature, I would spend some time learning Blender, or use premade assets from the Unity asset store. Then, I would import the models and set them as the plant/water models.
Infinite landscape	This feature would require generating “chunks” of landscape, incrementing and decrementing the Perlin noise offset depending on the player location. It would be quite hard to optimise.
Player model/animations	This feature would require using Blender to create a character model. From here I would have to learn Unity’s animation controller and animate the character for certain movements.
Swimming movement	This feature would require a secondary character movement script, with its own variables and constants to simulate the character swimming in water.
3D grass texture	This feature would consist of creating grass models in Blender or using a premade asset from the Unity asset store. Then, I could place the grass around the landscape, and allow it to move around when the player walks over it. Perhaps it could make a trail.
Graphical raytracing	This feature would require using a more resource heavy render pipeline within Unity, that deals with ray tracing. This would not be very difficult to implement, but it would negatively impact game performance.
Weather systems	This would require random events occurring throughout the explore mechanic. I could change the time by editing the height of the sun every time the player explores. It would take some more work to implement rain, extra fog, etc.

Here, I discuss suggested features or improvements that weren’t implemented during development. These can be seen as further limitations to my solution that I discovered throughout development.

Feature	**Evaluation**
Crouch	This could be implemented by checking for the “Control” button being pressed, which would just half the character size, which in turn lowers main camera height, giving the impression of crouch.
Tutorial	This could be done multiple ways. I could create a scene that walks the player through the common mechanics in the game. On the other hand, I could just create a new menu which lists all the mechanics and rules that occur in the game.
Player select menu	This would require a new menu being added to the opening screen. Furthermore, I would have to change the save/load function, checking for the selected player.
Save and load menu	This would require an overhaul of the save/load system. I would have to create binary save files and list the files to the user so that they can select which one they would like to load.
More colourful menus	This would be an easy implementation; I would have to create a new background image that scales well with all resolutions. Perhaps I could change the button and text colourings too.

Height colouring	This would require storing all the height values produced from the Perlin noise function. The largest would be set to a white colour to represent snow. Furthermore, surfaces with higher gradient could be coloured brown instead of green, since grass would not grow.
Landscape biomes	This could be done by using an array of pre-made biomes, which are randomly selected at the start of the explore mechanic. However, to make it more variant, I could use different aspects such as landscape colour, tree colour, plant types, weather and more to create hundreds of unique landscapes.
Map or compass	Implementing a map has already been discussed. However, the compass creates quite a challenge, and I would have to set up a north pole relative to the landscape. Then, I would have to ensure the needle always points to the north pole.
Interactive inventory	This would require an overhaul of the inventory system, creating an interactive menu, and allowing the player to remove objects. Furthermore, I could create an inventory limit. Perhaps this could be one of the upgrades.
Complex tree distribution	This feature would require a much more in-depth algorithm surrounding the tree distribution, perhaps ensuring that they are spawned in groups, like forests, and that they only are placed on relatively flat surfaces.
Powerups	This is one of the more interesting suggestions, and it would be quite simple to implement. I would have to create new objects and add them to the list of distributed items. Then I could come up with abilities, such as clearing fog, increasing player speed, magnetism when picking up items, and more.
Key-bind system	This is a big feature in terms of usability. To implement this, I would have to learn much more about the Unity input system and create a new menu within the settings menu.
Upgrades and end goal	This suggestion has already been discussed in the success criteria section.
Accessible pause menu	This would require many more values being saved to the host computer, such as player location, current landscape seed, and more. Or I could just disable saving in the pause menu when the player is exploring.
Binary save file	This is one of the most important suggestions. Currently the save system is reliable, but not secure at all. It can be easily changed by the user. Using a binary file would be much better for multiple already discussed reasons, but would require lots of learning on my end, since I am unsure how to implement such a feature.
Plant/water models	This suggestion has already been discussed in the limitations from the analysis section.
Better tree distribution	This is quite an easy implementation, just creating random rotational values and assigning them to each tree. Furthermore, I would move all the trees down a unit or so, to ensure the player cannot see the underside of the tree trunk.

Bibliography

Baraniuk, Chris. 2019. *The mysterious origins of an uncrackable video game.* 23 September. Accessed May 13, 2022. https://www.bbc.com/future/article/20190919-the-maze-puzzle-hidden-within-an-early-video-game.

Barone, Eric. 2016. *Stardew Valley.* Seattle, 26 February.

Demirçin, Yiğithan. 2022. *How to create a timer in Unity?* 7 January. Accessed November 23, 2022. https://mobidictum.biz/unity-timer/.

n.d. *Entombed (Atari 2600).* https://en.wikipedia.org/wiki/Entombed_(Atari_2600).

Flafla2. 2014. *Understanding Perlin Noise.* 09 August. Accessed December 5, 2022. https://adrianb.io/2014/08/09/perlinnoise.html.

Macgregor, Jody. 2019. *Farming Simulator 19 sold over two million copies.* 01 July. Accessed May 27, 2022. https://www.pcgamer.com/farming-simulator-19-sold-over-two-million-copies/.

Prahlow, Andrew. 2019. *Outer Wilds.* Los Angeles, 29 May.

Rivera, Joshua. 2019. *Outer Wilds' Time Loop Is Beautiful.* 4 June. Accessed May 13, 2022. https://kotaku.com/outer-wilds-time-loop-is-beautiful-1835238063.

Studios, Polypack. 2022. *POLYPACK Nature Pack - Low Poly 3D Art.* 20 October. Accessed August 13, 2022. https://www.unrealengine.com/marketplace/en-US/product/polypack-nature-pack-low-poly-3d-art.

Thirslund, Asbjørn. 2019. *FIRST PERSON MOVEMENT in Unity - FPS Controller.* 27 October. Accessed August 10, 2022. https://www.youtube.com/watch?v=_QajrabyTJc.

—. 2017. *SETTINGS MENU in Unity.* 6 December. Accessed September 20, 2022. https://www.youtube.com/watch?v=YOaYQrN1oYQ.

—. 2017. *START MENU in Unity.* 29 November. Accessed September 18, 2022. https://www.youtube.com/watch?v=zc8ac_qUXQY.

Totaka, Kazumi. 2020. *Animal Crossing: New Horizons.* Kyoto, 20 March.

Unity. 2022. *Collider.OnTriggerEnter(Collider).* 03 December. Accessed December 19, 2022. https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html.

—. 2022. *Scene.name.* 04 November. Accessed November 13, 2022. https://docs.unity3d.com/ScriptReference/SceneManagement.Scene-name.html.

—. 2022. *SceneManager.GetActiveScene.* 4 November. Accessed November 13, 2022. https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.GetActiveScene.html.

Vector, Broken. n.d. *Low Poly Tree Pack.* Accessed December 17, 2022. https://assetstore.unity.com/packages/3d/vegetation/trees/low-poly-tree-pack-57866.

Zając, Maciej. 2020. *How to pass variables between scenes in Unity? [1] .* 8 December. Accessed November 28, 2022. https://www.youtube.com/watch?v=BZjmqMd-4vo.

Appendices

DoorDetect.cs

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

// script to change the scene when
// the character walks into a door

public class DoorDetect : MonoBehaviour
{
    // reference to the DoorCheck object
    public Transform doorCheck;
    // radius of sphere to project
    public float doorDistance = 0.4f;
    // new mask which links to layer
    public LayerMask doorMask;

    bool touchingDoor;

    // Update is called once per frame
    void Update()
    {
        // creates sphere at:
        // player position
        // radius doorDistance
        // mask doorMask
        touchingDoor = Physics.CheckSphere(doorCheck.position, doorDistance, doorMask);

        // changes scene between "Inside" and "Outside" if touching door
        if (touchingDoor)
        {
            string name = SceneManager.GetActiveScene().name;
            switch (name)
            {
                case "Inside":
                    SceneManager.LoadScene("Outside");
                    break;
                case "Outside":
                    SceneManager.LoadScene("Inside");
                    break;
                default:
                    Debug.Log("Error occurred");
                    break;
            }
        }
    }
}

```

Global.cs

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

// script to store global variables

public class Global : MonoBehaviour
{
    // variable to store mouse sensitivity
    public static float sensitivity = 1000f;
    // variable to store timer length
    public static float timeRemaining;
    // variable to store player score
    public static int score;
    // variable to store player inventory
    public static List<string> inventory = new List<string>();
    // variable to store planter information
    public static int[,] planters = new int[16, 2];
    // variable to store coins
    public static int coins;
    // variable to store whether the player just explored
    public static bool justExplored = false;
}
```

Inventory.cs

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

public class Inventory : MonoBehaviour
{
    // declare variables
    string text;
    int green;
    int yellow;
    int pink;
    int water;

    string item;
    string name;
    string[] words;
    List<string> inv;

    // reference gameObjects
    public GameObject panel;

    // reference transforms
    public Transform inventory;

    // reference text
    public Text inventoryText;

    // Update is called once per frame
    void Update()
    {
        // update object visibility
        if (Input.GetButton("Inventory"))
        {
            panel.SetActive(true);
        }
        else
        {
            panel.SetActive(false);
        }

        // reset the variables
        inv = new List<string>();
        green = 0;
        yellow = 0;
        pink = 0;
        water = 0;
```

```
48
49          // generate string of items
50          foreach (Transform child in inventory)
51          {
52              switch (child.gameObject.name)
53              {
54                  case "Green plant(Clone)":
55                      green += 1;
56                      break;
57                  case "Yellow plant(Clone)":
58                      yellow += 1;
59                      break;
60                  case "Pink plant(Clone)":
61                      pink += 1;
62                      break;
63                  case "Water(Clone)":
64                      water += 1;
65                      break;
66              }
67
68              // add to inventory list
69              name = child.gameObject.name;
70              words = name.Split('(');
71              item = words[0];
72              inv.Add(item);
73          }
74
75          // create text variable
76          text = "Green plant: " + green +
77                      "\nYellow plant: " + yellow +
78                      "\nPink plant: " + pink +
79                      "\nWater: " + water;
80
81          // push to text object
82          inventoryText.text = text;
83          // push items to inventory global
84          Global.inventory = inv;
85      }
86  }
87
```

LostMenu.cs

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

public class LostMenu : MonoBehaviour
{
    // function that runs at beginning
    void Start()
    {
        // ensures that the cursor is unlocked
        Cursor.lockState = CursorLockMode.None;
        // clear inventory and score variables
        Global.inventory = new List<string>();
        Global.score = 0;
        // set just explored to false
        Global.justExplored = false;
    }

    public void Load()
    {
        // call the load procedure from MainMenu
        MainMenu.Load();
    }

    public void Return()
    {
        // loads the start scene
        SceneManager.LoadScene("Start");
    }
}
```

MainMenu.cs

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

public class MainMenu : MonoBehaviour
{
    // procedure that runs at beginning
    void Start()
    {
        // ensures that the cursor is unlocked
        Cursor.lockState = CursorLockMode.None;
    }

    // procedure to run when "New" button is pressed
    public void NewGame()
    {
        // wipe any data stored in memory
        Global.planters = new int[16, 2];
        Global.coins = 0;
        SceneManager.LoadScene("Inside");
    }

    // procedure to load any saved variables
    public static void Load()
    {
        // set up name variable
        string name = "";
        // load the Global.coins integer
        Global.coins = PlayerPrefs.GetInt("Coins");
        // load the Global.planters 2D array
        for (int i = 0; i < 16; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                name = "Plant[" + i.ToString() + "," + j.ToString() + "]";
                Global.planters[i, j] = PlayerPrefs.GetInt(name);
            }
        }
        // load the inside scene
        SceneManager.LoadScene("Inside");
    }

    public void Options()
    {
        SceneManager.LoadScene("Options");
    }

    // procedure to run when "Quit" button is pressed
    public void QuitGame()
    {
        // quit the application
        Application.Quit();
    }
}
```

MouseLook.cs

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

public class MouseLook : MonoBehaviour
{
    // create transform for the player body
    public Transform playerBody;
    // create a variable for the x-rotation
    float xRotation = 0f;

    // start is called before the first frame update
    void Start()
    {
        // lock the cursor to the centre of the screen
        Cursor.lockState = CursorLockMode.Locked;
    }

    // update is called once per frame
    void Update()
    {
        // creates variables to store the mouse axis
        float mouseX = Input.GetAxis("Mouse X") * Global.sensitivity * Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * Global.sensitivity * Time.deltaTime;

        // every frame increase xRotation by mouseY
        xRotation -= mouseY;
        // clamp the rotation
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);


        // rotate camera around the x-axis
        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        // rotate the player body around the y-axis
        playerBody.Rotate(Vector3.up * mouseX);
    }
}
```

ObjectCollision.cs

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

public class ObjectCollision : MonoBehaviour
{
    // reference the transforms
    public Transform inventory;

    // check for collisions
    void OnTriggerEnter(Collider col)
    {
        if (col.gameObject.tag == "Player")
        {
            // move the object out of view
            transform.position = new Vector3(-20, -5, 0);
            // change the parent to inventory
            transform.parent = inventory;
            // add respective score
            switch (gameObject.name)
            {
                case "Green plant(Clone)":
                    Global.score += 1;
                    break;
                case "Yellow plant(Clone)":
                    Global.score += 2;
                    break;
                case "Pink plant(Clone)":
                    Global.score += 4;
                    break;
                case "Water(Clone)":
                    Global.score += 1;
                    break;
            }
        }
    }
}
```

OptionsMenu.cs

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

public class OptionsMenu : MonoBehaviour
{
    // reference to audio mixer
    public AudioMixer audioMixer;
    // reference to resolution dropdown
    public Dropdown resolutionDropdown;
    // array to store the resolutions
    Resolution[] resolutions;

    // function that runs at beginning
    void Start()
    {
        // ensures that the cursor is unlocked
        Cursor.lockState = CursorLockMode.None;

        // fill array with resolutions
        resolutions = Screen.resolutions;
        // remove any options already in dropdown
        resolutionDropdown.ClearOptions();
        // list to store string versions of resolutions
        List<string> options = new List<string>();

        // integer to store the user's current resolution (indexed)
        int currentResolutionIndex = 0;

        // loops through all the resolutions
        for (int i = 0; i < resolutions.Length; i++)
        {
            // must be in format: "width" + " x " + "height"
            string option = resolutions[i].width + " x " + resolutions[i].height;
            // add to options list
            options.Add(option);

            // check whether current resolution is equal to resolutions[i]
            if (resolutions[i].width == Screen.currentResolution.width &&
                resolutions[i].height == Screen.currentResolution.height)
            {
                // if so, save it as the current resolution index
                currentResolutionIndex = i;
            }
        }

        // add the options to the dropdown menu
        resolutionDropdown.AddOptions(options);
        // set the default options to the current resolution
        resolutionDropdown.value = currentResolutionIndex;
        // refresh the dropdown menu
        resolutionDropdown.RefreshShownValue();
    }
```

```
58      // function to change the game volume
59      public void SetVolume(float volume)
60      {
61          // sets the audio mixer volume to slider value
62          audioMixer.SetFloat("Volume", volume);
63      }
64
65      // function to change the game quality
66      public void SetQuality(int qualityIndex)
67      {
68          // sets the game quality to desired level
69          QualitySettings.SetQualityLevel(qualityIndex);
70      }
71
72      // function to toggle fullscreen mode
73      public void SetFullscreen(bool isFullscreen)
74      {
75          // set fullscreen to the input bool
76          Screen.fullScreen = isFullscreen;
77      }
78
79      // function to set the resolution
80      public void SetResolution(int resolutionIndex)
81      {
82          // new variable to store the selected resolution
83          Resolution resolution = resolutions[resolutionIndex];
84          // accepts in format: width (int), height (int), fullscreen (bool)
85          Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
86
87      }
88
89      public void SetSensitivity(float sensitivity)
90      {
91          Global.sensitivity = sensitivity;
92      }
93
94      public void Return()
95      {
96          SceneManager.LoadScene("Start");
97      }
98  }
99
```

Pause.cs

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

public class Pause : MonoBehaviour
{
    // Update is called once per frame
    void Update()
    {
        // check if escape is pressed
        if (Input.GetButton("Cancel"))
        {
            SceneManager.LoadScene("Pause");
        }
    }
}
```

PauseMenu.cs

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

public class PauseMenu : MonoBehaviour
{
    // procedure that runs at beginning
    void Start()
    {
        // ensures that the cursor is unlocked
        Cursor.lockState = CursorLockMode.None;
    }

    // procedure to save crucial variables
    public void Save()
    {
        // set up name variable
        string name = "";
        // store the Global.coins integer
        PlayerPrefs.SetInt("Coins", Global.coins);
        // store the Global.planters 2D array
        for (int i = 0; i < 16; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                name = "Plant[" + i.ToString() + "," + j.ToString() + "]";
                PlayerPrefs.SetInt(name, Global.planters[i, j]);
            }
        }
        // save all the changes to PlayerPrefs
        PlayerPrefs.Save();
    }

    // procedure to exit to main menu
    public void Exit()
    {
        // exit to the start screen
        SceneManager.LoadScene("Start");
    }

    // procedure to return back to game
    public void Return()
    {
        // return back to the player house
        SceneManager.LoadScene("Inside");
    }
}
```

PlayerMovement.cs

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

public class PlayerMovement : MonoBehaviour
{
    // reference the character controller
    public CharacterController controller;

    // speed of character movement
    public float speed = 12f;
    // value for gravity
    public float gravity = -9.81f;
    // value for jump height
    public float jumpHeight = 1f;

    // reference the GroundCheck object
    public Transform groundCheck;
    // radius of sphere to project
    public float groundDistance = 0.4f;
    // create a mask to tell sphere what objects to check for
    public LayerMask groundMask;

    // vector to store velocity
    Vector3 velocity;

    // boolean to store whether player is grounded or not
    bool isGrounded;

    // Update is called once per frame
    void Update()
    {
        // creates sphere of player position, radius groundDistance and mask groundMask
        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

        // sets the character velocity to 0 if the player is touching the floor and has velocity
        if (isGrounded && velocity.y < 0)
        {
            velocity.y = 0f;
        }

        // assign keyboard input to variables
        float x = Input.GetAxis("Horizontal"); // A and D
        float z = Input.GetAxis("Vertical"); // W and S

        // direction that the character will move
        Vector3 move = transform.right * x + transform.forward * z;

        // call the character controller function, takes a Vector3
        controller.Move(move * speed * Time.deltaTime);

        // check if the player wants to jump
        if (Input.GetButtonDown("Jump") && isGrounded)
        {
            velocity.y = Mathf.Sqrt(-2 * gravity * jumpHeight);
        }

        // increase the velocity by gravity variable
        velocity.y += gravity * Time.deltaTime;

        // reference character controller and make player fall
        controller.Move(velocity * Time.deltaTime);
    }
}
```

Score.cs

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

public class Score : MonoBehaviour
{
    // reference to the score text object
    public Text scoreText;

    // Start is called before the first frame update
    void Start()
    {
        Global.score = 0;
    }

    // Update is called once per frame
    void Update()
    {
        scoreText.text = Global.score.ToString();
    }
}

```

TerrainGeneration.cs

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

public class TerrainGeneration : MonoBehaviour
{
    // declare terrain size variables
    static int xSize = 4096; // x axis
    static int zSize = 4096; // z axis
    static int ySize = 800; // y axis

    // calculate midpoint
    int xMid = xSize / 2;
    int zMid = zSize / 2;

    // declare perlin noise variables
    float scale = 4f;
    float xOffset; // random value
    float zOffset; // random value
    int octaves = 6;
    float lacunarity = 2f; // frequency multiplier
    float persistence = 0.5f; // amplitude multiplier

    // declare lists
    List<GameObject> plantList;
    List<GameObject> waterList;
    List<GameObject> treeList;

    // reference gameobjects
    public GameObject house;
    public GameObject character;

    // reference transforms
    public Transform plants;
    public Transform waters;
    public Transform trees;

    // reference character controller
    public CharacterController controller;
```

```
41      // Start is called before the first frame update
42      void Start()
43      {
44          // set justExplored to true
45          Global.justExplored = true;
46
47          // generate list of plants
48          plantList = new List<GameObject>();
49          foreach (Transform child in plants)
50          {
51              plantList.Add(child.gameObject);
52          }
53
54          // generate list of waters
55          waterList = new List<GameObject>();
56          foreach (Transform child in waters)
57          {
58              waterList.Add(child.gameObject);
59          }
60
61          // generate list of trees
62          treeList = new List<GameObject>();
63          foreach (Transform child in trees)
64          {
65              treeList.Add(child.gameObject);
66          }
67
68          // calculate random offsets
69          xOffset = Random.Range(0f, 9999f);
70          zOffset = Random.Range(0f, 9999f);
71
72          // reference terrain component to change data
73          Terrain terrain = GetComponent<Terrain>();
74          // create new terrain based off of current terrain
75          terrain.terrainData = GenerateTerrain(terrain.terrainData);
76      }
```

```
78      float CalculateHeight(int x, int z)
79      {
80          // calculate distance from centre
81          int xDist = Mathf.Abs(xMid - x);
82          int zDist = Mathf.Abs(zMid - z);
83          float dist = Mathf.Sqrt(Mathf.Pow(xDist, 2) + Mathf.Pow(zDist, 2));
84
85          // calculate the gradient1 multiplier (outer gradient)
86          int cutoff = xMid - 200;
87          float gradient = 1f;
88          if (dist > cutoff)
89          {
90              float extra = dist - cutoff;
91              gradient = gradient - (extra * 0.0005f);
92          }
93
94          // calculate the y value
95          float y = 0f;
96          float xCoord = (float)x / xSize * scale + xOffset;
97          float zCoord = (float)z / zSize * scale + zOffset;
98          float frequency = 1f;
99          float amplitude = 0.5f;
100         for (int i = 0; i < octaves; i++)
101         {
102             y = y + (amplitude * Mathf.PerlinNoise(xCoord * frequency, zCoord * frequency));
103             y = y * gradient;
104             frequency = frequency * lacunarity;
105             amplitude = amplitude * persistence;
106         }
107
108         // move objects if necessary
109         MoveObjects(dist, y);
110
111         // create objects if necessary
112         CreateObjects(x, z, y);
113
114         // return the final height
115         return y;
116     }
```

```
118     void MoveObjects(float distance, float y)
119     {
120         // check if current point is at centre
121         if (distance == 0f)
122         {
123             // convert terrain y value to height coordinate
124             float height = (y * ySize) - 300;
125             // check if it is below water level
126             if (height < 0f)
127             {
128                 height = 0f;
129             }
130             // move the house to new height
131             house.transform.position = new Vector3(0, height, 0);
132             // move the character to new height
133             controller.enabled = false;
134             character.transform.position = new Vector3(-2, height + 4f, 0);
135             controller.enabled = true;
136         }
137     }
```

```
139     void CreateObjects(int x, int z, float y)
140     {
141         // convert the x, z and y values to coordinates
142         float yCoord = (y * ySize) - 300;
143         float xCoord = (z - xMid);
144         float zCoord = (x - zMid);
145 
146         // check if below water level
147         if (yCoord < 0)
148         {
149             // exit this function
150             return;
151         }
152 
153         // store these in a vector3 variable
154         Vector3 position = new Vector3(xCoord, yCoord, zCoord);
155 
156         // generate a random number between 0 and 99999
157         int num1 = Random.Range(0, 100000);
158         // check number, perform necessary action
159         switch (num1)
160         {
161             case int n when (n >= 0 && n <= 3):
162                 // create a green plant object
163                 Instantiate(plantList[0], position, Quaternion.identity, plants);
164                 break;
165             case int n when (n >= 4 && n <= 5):
166                 // create a yellow plant object
167                 Instantiate(plantList[1], position, Quaternion.identity, plants);
168                 break;
169             case 6:
170                 // create a pink plant object
171                 Instantiate(plantList[2], position, Quaternion.identity, plants);
172                 break;
173             case int n when (n >= 7 && n <= 13):
174                 // create a water object
175                 Instantiate(waterList[0], position, Quaternion.identity, waters);
176                 break;
177             case int n when (n >= 14 && n <= 200):
178                 // generate a random number between 0 and 4
179                 int num2 = Random.Range(0, 5);
180                 // create a tree object
181                 Instantiate(treeList[num2], position, Quaternion.identity, trees);
182                 break;
183             default:
184                 break;
185         }
186     }
```

```
188     float[,] GenerateHeights()
189     {
190         // create a grid of floats for heights of terrain
191         float[,] heights = new float[xSize, zSize];
192         // use perlin noise
193         for (int x = 0; x < xSize; x++)
194         {
195             for (int z = 0; z < zSize; z++)
196             {
197                 heights[x, z] = CalculateHeight(x, z);
198             }
199         }
200         // return the heights array
201         return heights;
202     }
203 
204     TerrainData GenerateTerrain(TerrainData terrainData)
205     {
206         terrainData.heightmapResolution = xSize + 1;
207         // set up size of terrain
208         terrainData.size = new Vector3(xSize, ySize, zSize);
209         // modify heights of floats in terrain
210         terrainData.SetHeights(0, 0, GenerateHeights()); // from start point 0, 0
211         return terrainData;
212     }
213 }
214 
```

Timer.cs

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

public class Timer : MonoBehaviour
{
    // boolean to check whether timer is running
    private bool timerRunning = false;
    // reference to the timer text object
    public Text timerText;

    // Start is called before the first frame update
    void Start()
    {
        // assign the timer length
        Global.timeRemaining = 600;
        // starts timer automatically
        timerRunning = true;
    }

    // Update is called once per frame
    void Update()
    {
        if (timerRunning)
        {
            // if there is time left on the timer
            if (Global.timeRemaining > 0)
            {
                // subtract the time length of the last frame
                Global.timeRemaining -= Time.deltaTime;
                // calculate minutes and seconds
                int minutes = Mathf.FloorToInt(Global.timeRemaining / 60);
                int seconds = Mathf.FloorToInt(Global.timeRemaining % 60);
                // convert values to strings
                string minutes_str = minutes.ToString();
                string seconds_str = seconds.ToString();
                // check if a leading zero is required\
                if (minutes_str.Length == 1)
                {
                    minutes_str = "0" + minutes_str;
                }
                if (seconds_str.Length == 1)
                {
                    seconds_str = "0" + seconds_str;
                }
                // push this to the text object
                timerText.text = minutes_str + ":" + seconds_str;
                // make fog end distance equal to time left
                RenderSettings.fogEndDistance = Global.timeRemaining * 2;
            }
            else
            {
                // end the timer
                Global.timeRemaining = 0;
                timerRunning = false;
                // timer has run out, change the scene
                SceneManager.LoadScene("Lost");
            }
        }
    }
}
```

UpdatePlanters.cs

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

public class UpdatePlanters : MonoBehaviour
{
    // reference gameobjects
    public GameObject greenPlant;
    public GameObject yellowPlant;
    public GameObject pinkPlant;

    // reference to the score text object
    public Text coinText;

    // Start is called before the first frame update
    void Start()
    {
        if (Global.justExplored)
        {
            updateWater();
            updatePlants();
            // set justExplored to false
            Global.justExplored = false;
            // add score to coins
            Global.coins += Global.score;
        }
        updateCoins();
        moveObjects();
    }

    // function to decode number
    string intToStr(int num)
    {
        // string to store name
        string name = "";
        // switch between input
        switch (num)
        {
            case 1:
                name = "Green plant";
                break;
            case 2:
                name = "Yellow plant";
                break;
            case 4:
                name = "Pink plant";
                break;
        }
        // return decoded number
        return name;
    }
```

```
54      // function to encode string
55      int strToInt(string name)
56      {
57          // int to store num
58          int num = 0;
59          // switch between input
60          switch (name)
61          {
62              case "Green plant":
63                  num = 1;
64                  break;
65              case "Yellow plant":
66                  num = 2;
67                  break;
68              case "Pink plant":
69                  num = 4;
70                  break;
71          }
72          // return encoded string
73          return num;
74      }
75
76      void updateWater()
77      {
78          // calculate amount of water in inventory
79          int amount = 0;
80          foreach (string item in Global.inventory)
81          {
82              if (item == "Water")
83              {
84                  amount += 1;
85              }
86          }
87          // iterate through all the plants currently stored
88          for (int i = 0; i < 16; i++)
89          {
90              // check if there is water left and there is a plant
91              if (amount > 0 && Global.planters[i, 0] != 0)
92              {
93                  // increase current plant height by 1
94                  Global.planters[i, 1] += 1;
95                  // decrement amount value
96                  amount -= 1;
97              }
98              // if there is no water remaining and a plant
99              else if (amount <= 0 && Global.planters[i, 0] != 0)
100             {
101                 // remove current plant
102                 Global.planters[i, 0] = 0;
103                 Global.planters[i, 1] = 0;
104             }
105         }
106     }
```

```
108     void updatePlants()
109     {
110         // calculate inventory length
111         int length = Global.inventory.Count;
112         // iterate through all the items in inventory
113         for (int i = 0; i < length; i++)
114         {
115             // check if item is a plant
116             if (Global.inventory[i].Contains("plant"))
117             {
118                 // iterate through planter slots
119                 for (int j = 0; j < 16; j++)
120                 {
121                     // if there is an empty slot
122                     if (Global.planters[j, 0] == 0)
123                     {
124                         // set colour
125                         Global.planters[j, 0] = strToInt(Global.inventory[i]);
126                         // set height
127                         Global.planters[j, 1] = 1;
128                         // break out of the loop
129                         break;
130                     }
131                 }
132             }
133         }
134     }
135
136     void updateCoins()
137     {
138         // iterate through the planters array, checking if height is max
139         for (int i = 0; i < 16; i++)
140         {
141             // if plant is at max height
142             if (Global.planters[i, 1] == 4)
143             {
144                 // reward the player with coins
145                 Global.coins += Global.planters[i, 0] * 100;
146                 // remove the current plant
147                 Global.planters[i, 0] = 0;
148                 Global.planters[i, 1] = 0;
149             }
150         }
151         // push the coins value to text attribute
152         coinText.text = Global.coins.ToString();
153     }
154
```

```
155     void moveObjects()
156     {
157         // set up a variable to store a game object, with default value to avoid error
158         GameObject plant = greenPlant;
159         // iterate through planter array
160         for (int i = 0; i < 16; i++)
161         {
162             // store plant information in different variables
163             int colour = Global.planters[i, 0];
164             int height = Global.planters[i, 1];
165 
166             // if there is a plant, and the slot is not empty
167             if (colour != 0)
168             {
169                 // switch between plant colour
170                 switch (colour)
171                 {
172                     case 1:
173                         // set green plant
174                         plant = greenPlant;
175                         break;
176 
177                     case 2:
178                         // set yellow plant
179                         plant = yellowPlant;
180                         break;
181 
182                     case 4:
183                         // set pink plant
184                         plant = pinkPlant;
185                         break;
186                 }
187 
188                 // create the position variable
189                 float xCoord = (float)(3.25 - 5 * (i / 4) - 0.5 * (i % 4));
190                 float yCoord = (float)(0.5 * height);
191                 float zCoord = (float)(-4.5);
192                 Vector3 position = new Vector3(xCoord, yCoord, zCoord);
193                 // create the new object
194                 Instantiate(plant, position, Quaternion.identity);
195             }
196         }
197     }
198 }
199 
```