Maintenance Guide

Contents

1 **Overview** 3

2 **How to open the project** 3

2.1 GitHub 3

2.2 Unity editor 3

3 **File Structure** 4

4 **Scenes** 5

4.1 SampleScene 5

4.2 MenuScene 8

4.3 BackStory, Controls and CoinShop 9

5 **Scripts** 12

5.1 Animation_Controller 12

5.2 Background_Controller 13

5.3 Backstory_UI_Controller 13

5.4 Camera_Controller 13

5.5 Character_Controller 13

5.6 Coin 14

5.7 coins_Controller 14

5.8 CoinShop 14

5.9 Controls_UI_Controller 15

5.10 Game_State_Controller 15

5.11 Game_UI_Controller 15

5.12 Menu_UI_Controller 16

5.13 Obstacle_Controller 16

5.14 Pipe_Generation 16

5.15 Pipe_Properties 17

5.16 Pipe_System 17

5.17 pipes_Interface 17

5.18 SaveGame 18

5.19 Score 18

6 **Prefabs** 19

7 **Animations** 19

8 Known Bugs 20

8.1 Slight animation glitch on the long straight pipe 20
8.2 The game can is only suited for 1080p resolution 20
8.3 The pipes do not fit together perfectly 20
8.4 Objects can overlap 20
9 **Future development ideas** 21
9.1 Add different pipe shapes 21
9.2 Add pipe features 21
9.3 Add Beat Bonus 21
9.4 Make backstory animated/scrollable 21

1 Overview

We made our game using the unity game engine. In this guide we explain the file structure, the elements of our game and the scripts we have used to give the game functionality. This guide should allow you to understand the complete functionality of the game, allowing you to perform any required maintenance tasks as well as adding new feature to the game if you wish to, some ideas for features are suggested at the end of this guide.

This guide starts with some introductory features which assume little prior knowledge of using GitHub or game development using unity, if you are well versed in these areas you may want to skip to the File Structure section of this guide once you have accessed the repository from the link below.

2 How to open the project

2.1 GitHub

To access and maintain the project you will first need to download/clone the remote repository from GitHub. The link to the GitHub repository is https://github.com/MarcusOWilliams/Software-Engineering-CW2.git

Within the documentation folder of this repository, you can find a more detailed file, that walks you through the steps of how to clone the repository if this is not something you are familiar with. But the general idea of how this project was created is as follows:

An example of the branch structure of our GitHub repository

We had a main branch which always contained the version of the game produced from the previous sprint (we also created a new release of the game at the end of each sprint), throughout each sprint we would do all our work on the development branch, from which we would create a new branch for each specific feature we implemented.

A visualisation of the actual branch structure from out project can be seen at https://github.com/MarcusOWilliams/Software-Engineering-CW2/network

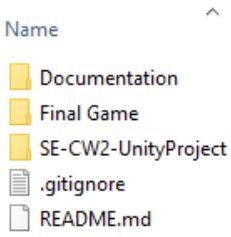
By knowing this structure of the project, you can go back to different stages of the development of the game to see how the game evolved, this may help if it is ever unclear how or why something was created.

2.2 Unity editor

This project was created using the Unity game engine, you can download the Unity hub and editor here: https://unity.com/download , this project was originally developed using Unity Editor 2020.3.23. Once in unity you need to open the folder that has “UnityProject” in its name from the GitHub repository, you are now ready to work on the project.

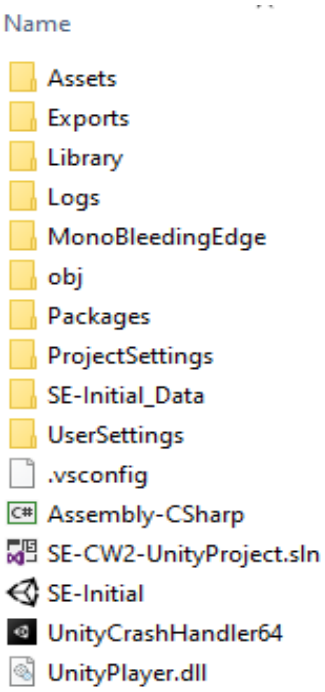
3 File Structure

The root file of the GitHub repository contains 3 subfolders, the one we are interested in for game development and maintenance is SE-CW2-UnityProject.



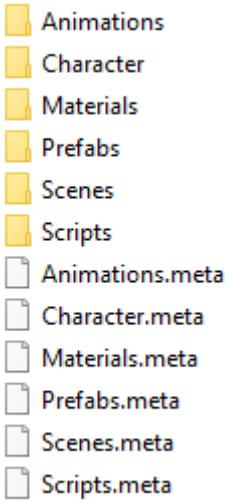
The root folder

When you first open the project in unity you will see an Assets folder, this is where all the files for the game are kept.



The SE-CW2-UnityProject folder

Some main common components of unity projects all described in more detail below, we followed a common structure found in unity projects to keep our game and files organised. The main components of the project are the folders, Scene's, Scripts, Prefabs and Animations, these each have their own folder.

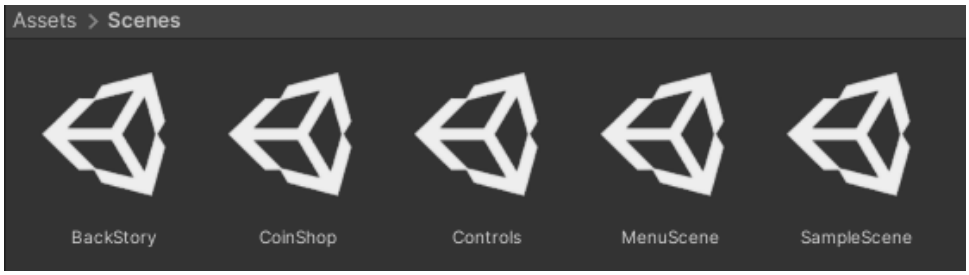


The assets folder

4 Scenes

In Unity, a game is made up of many Scenes, for example the main menu scene and the main game scene. These scenes contain the objects of the game, each scene can be thought of as a unique stage for the game with a different environment. For our game, the main two scenes are called SampleScene and MenuScene. SampleScene is the scene in which the game is played, this is where most of the files are and where most of the development takes place. MenuScene is the scene for the main menu, this is where the user will first load when the game is launched and from here, they will be able to navigate to other scenes. There are also other scenes called BackStory, CoinShop and Controls these are used to show the backstory of the game, a shop to spend your coins and how to play it, respectively.

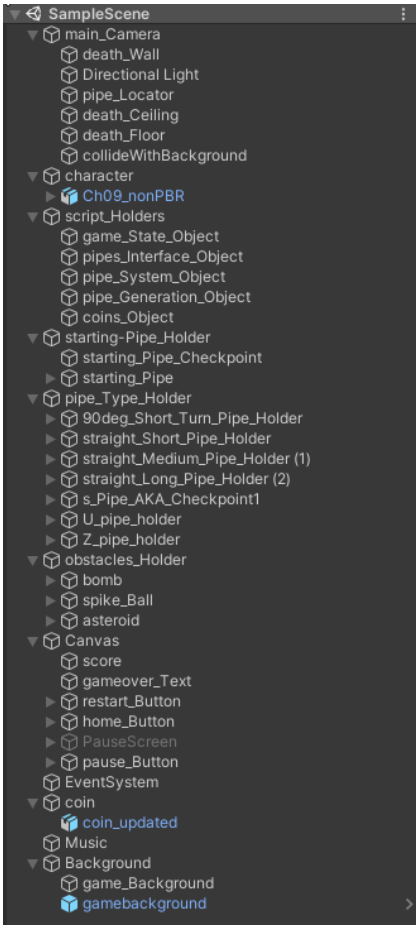
Each scene is made up of many game objects, these can be anything from the background, UI text, objects for the player to interact with, the character, placeholders for scripts and more, the breakdown of each scene into its game objects and what they do is described below.



The scenes in the unity project

4.1 SampleScene

This section is a breakdown off all the parent game objects in SampleScene, what they do and how to alter them:



All game objects in SampleScene

A screenshot of SampleScene in the unity editor

main_Camera – This object is involved with the players view of the game, the parent object is a camera which moves with increasing speed as the game progresses it also controls the locations for the "death walls" which kill the player if they are touched, these are located at the top, bottom and left side of the players view currently, to avoid the player leaving the screen.

This parent object also contains the pipe_Locator, this is used to help decide the position of newly generated objects, originally just pipes, but has been implemented for coin an obstacle generation. The last couple of objects within this class are the lighting and a collider for the background.

The Scripts associated with this object: Camera_controller.

character - This is associated with the players character, the parent object is RigidBody and collider which is involved with the physics of the character, this is what is moved to new checkpoints as the player moves through the pipes.

The child object in this class is the character model, called CharacterModel_SchoolBoy this is the part of the character that the player sees moving through the pipes and it is animated depending on its movement behaviour.

The Scripts associated with this object: Character_Controller, Animation_Controller.

script_Holders – In unity you often need to have a script associated with a game object which is not necessarily a visible component of the game but allows you to interact with a range of other game objects. This parent object is where we kept all our scripts that act as placeholders for scripts needed in SampleScene, the objects are named based on the script they are associated with.

The Scripts associated with this object: Game_State_Controller, Pipe_Interface, Pipe_System, Pipe_Generation, Coin_Controller.

starting-Pipe_Holder – This object is linked to the pipe the player starts in when they enter the scene, the parent object contains the prefab for the pipe itself as well as the checkpoint which indicates the end of the pipe.

pipe_Type_Holder – This is the parent object which contains all the different pipe pieces the player can use in the game. Each pipe piece is its own child and contains: The prefab of the pipe, the checkpoints to move through the pipe, colliders to allow the pipe to be selected and an associated pipe_properties script which deals with setting pipe entry and exit.

obsatcles_Holder - This object holds all the obstacles that appear throughout the game, the obstacles each contain a prefab to give the obstacle its look and a collider which interacts with the player and makes it game over if the player touches the obstacle.

Canvas - The canvas is associated with the UI of the game. It contains the score text, the game over text, pause button, pause screen, home button and restart button (for when the game is over).

The Scripts associated with this object: game_UI_Controller, Score.

EventSystem - The Event System is a default object created in unity which allows you to send events to objects in the application based on an input, be it keyboard, mouse, touch, or custom input. The Event System consists of a few components that work together to send events. An Event System component does not have much functionality exposed, this is because the Event System itself is designed as a manager and facilitator of communication between Event System modules. The most common roles for an event system are: Managing which game object is selected and managing which input module is in use or required. You should not need to edit the EventSystem.

coin - This object is associated with the coins that appear throughout the game; it contains the prefab of the coin.

The Scripts associated with this object: Coin.

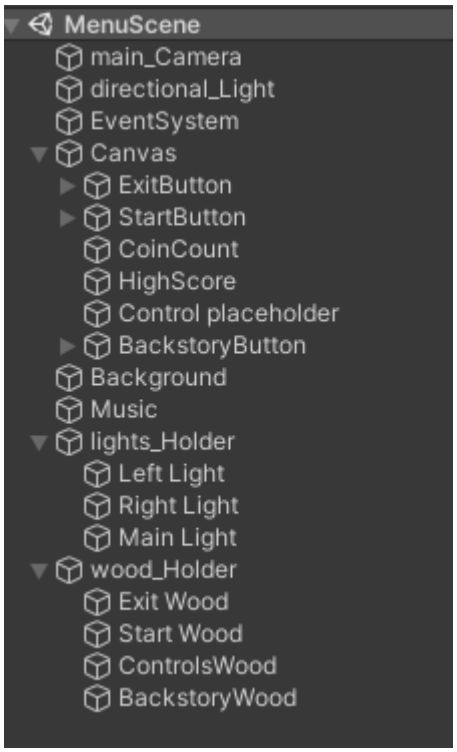
Music – This object is responsible for the music that is played when the scene is running.

Background - Background holds the objects related to the continuous background of the game, it holds the picture of the background and an object which allows the background to repeat.

The Scripts associated with this object: background_Controller.

4.2 MenuScene

This section is a breakdown off all the parent game objects in MenuScene, what they do, and how to alter them:



The game objects in MenuScene

A screenshot of MenuScene in the unity editor

main_Camera - This object is involved with the players view of the menu; the parent object is a camera which shows the stationary menu display.

The Scripts associated with this object: camera_controller

directional_Light - This object manages the subtle lighting projected onto the display. The colour of this light can be easily adjusted within the 'Light' component of the Inspector in Unity. It is currently set at a translucent red tone to complement the dungeon theme.

EventSystem – This game object is the same as described in SampleScene

Canvas - The canvas is associated with the UI of the menu. It contains the start, exit, controls, and backstory buttons, as well as the coin count and high score text.

The Scripts associated with this object: menu_UI_Controller.

Background - As the menu background is not continuous as the SampleScene background is, this object simply contains the chosen menu image.

The Scripts associated with this object: background_Controller.

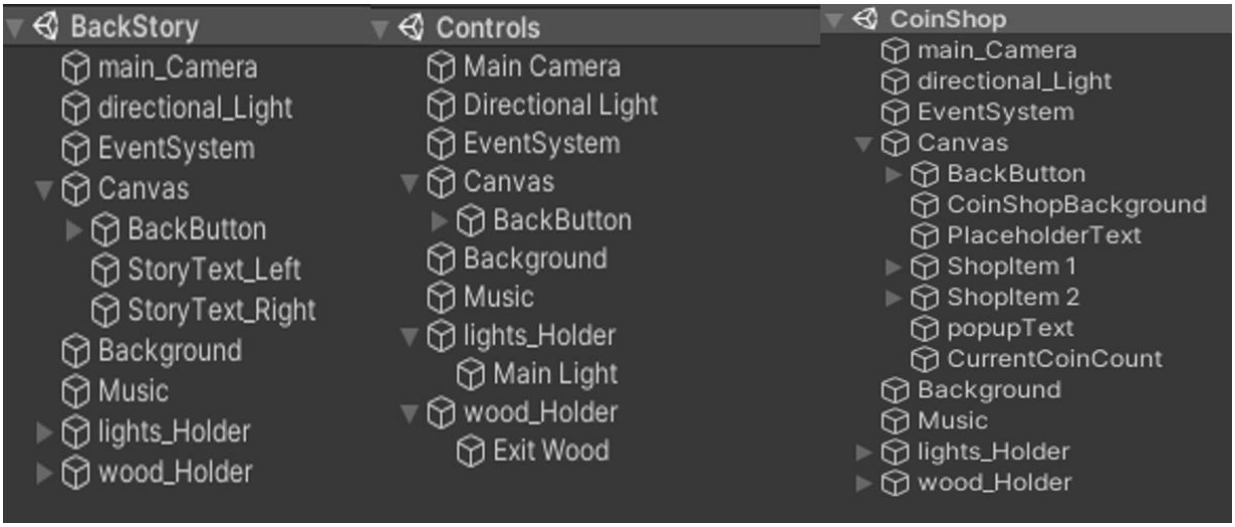
Music - This object is responsible for the music that is played whilst the menu scene is active.

lights_Holder - This object contains three light objects used to represent the fire shown on the left- and right-hand sides of the door on the menu image, as well as the illumination on the central door in the image behind the start/exit buttons etc. Features of these objects such as the colour, shadows, or translucency can be adjusted in the 'Light' component section of the Inspector in Unity.

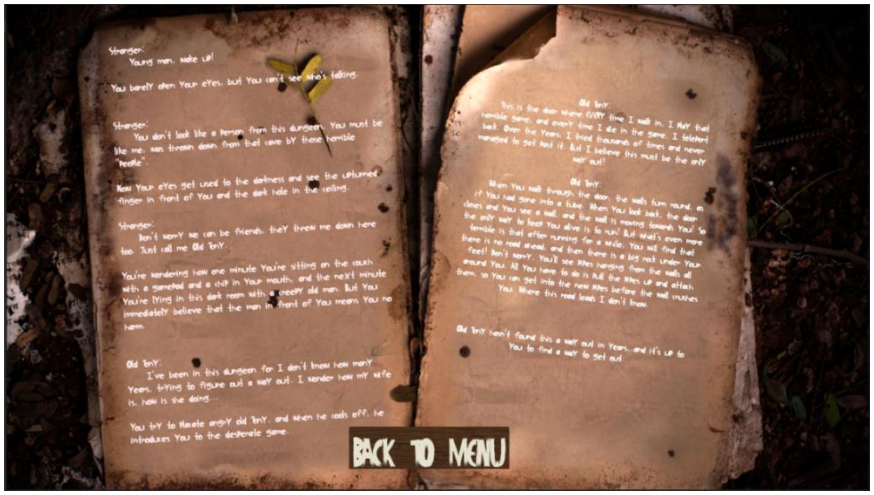
wood_Holder - This object contains four objects, with each being used for one of the four buttons displayed on the menu screen. Each object is rendered with the material M_Wood 3 to match the wooden door theme in the menu background image.

4.3 BackStory, Controls and CoinShop

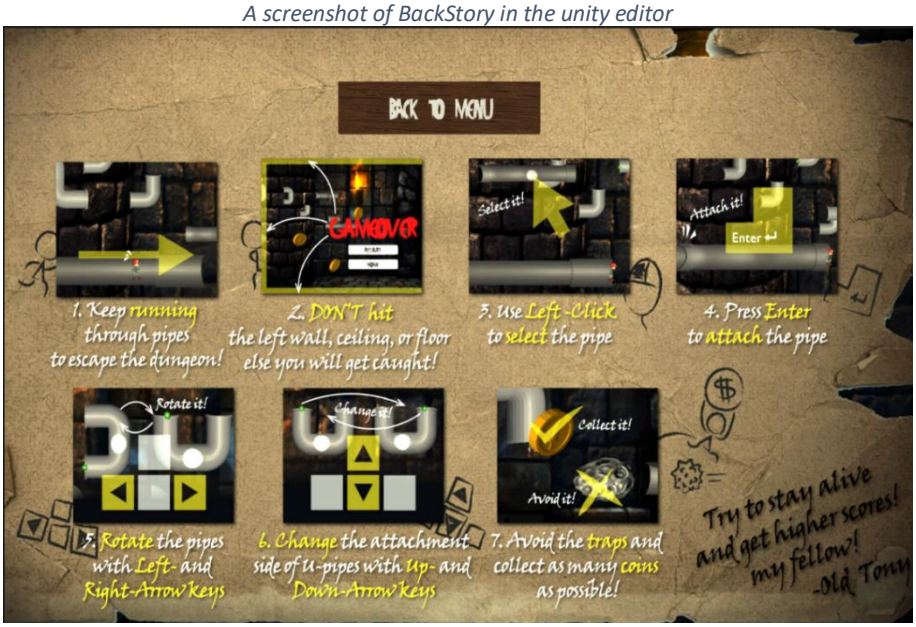
This section is a breakdown off all the parent game objects in BackStory, Controls and CoinShop, what they do, and how to alter them, these three scenes are very similar, they all just act as a way of showing information to the user depending on the button pressed from the menu scene, CoinShop has some extra functionality which is also described below:



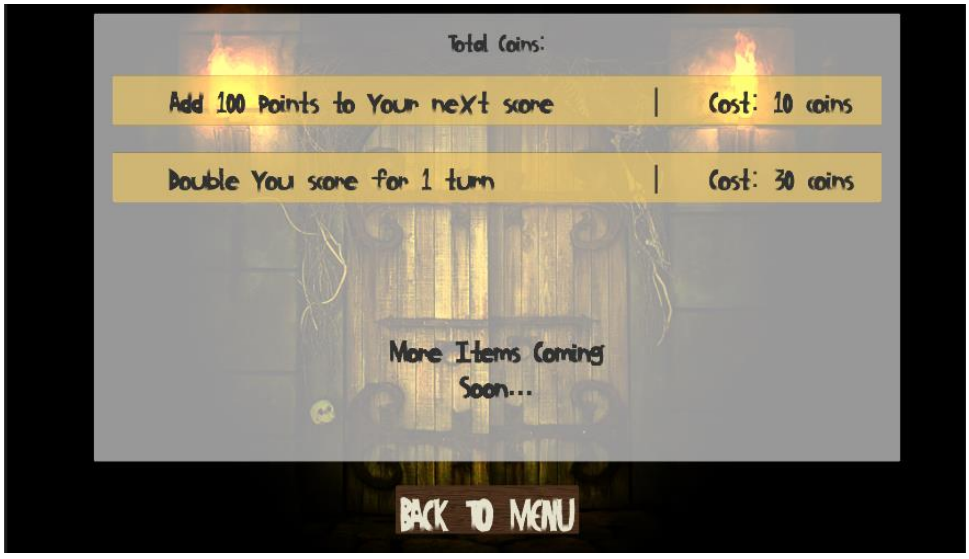
The game objects in BackStory, Controls and CoinShop



A screenshot of BackStory in the unity editor



A screenshot of Controls in the unity editor



A screenshot of CoinShop in the unity editor

main_Camera- The main camera object is used to set the users view of the scene; in both cases this is just the default static world camera that is generated when you make a new scene in unity.

directional_Light - This is the light source that allows the objects to be scene, similarly to the camera, there is only one light source in each scene and it is the default light source that unity creates when you make a new scene

EventSystem - See the description for EventSystem in SampleScene.

Canvas - The canvas varies slightly between the three scenes, but it contains components that make up the UI, in all cases this includes the button which leads back to the main menu. Both the BackStory and CoinShop scenes contain some additional text, whereas in Controls the text is part of the background. The CoinShop scene also has shop items, these are buttons which “purchase” abilities if the Player has enough coins.

The Scripts associated with this object for each scene: backstory_UI_Controller, controls_UI_Controller and CoinShop.

Background - The background object holds the image that makes up the background, in both BackStory and Controls the background was designed for the specific scene, whereas in CoinShop the background from the main menu was used and an extra transparent layer was added to the canvas.

Music - This object is responsible for the music that is played when the scene is running, this is the same music as the main menu.

lights_Holder - lights_holder holds any additional lights for the scene, for example CoinShop has an extra light for each torch that appears in the background

wood_Holder - This object is responsible for holding the wood panels that display the background for the exit buttons in all the scenes

5 Scripts

As mentioned throughout this guide so far, the unity project also contains many scripts, these scripts tell the game objects how to behave, they are what gives the game its functionality. For out project all the scripts are written in C#, we used the Microsoft Visual Studio text editor which has a plugin built specifically for unity game development. Each script contains its own class and methods and acts on specific game objects.

There are some common methods found throughout many of the classes, these are default methods provided with the UnityEngine C# library. The main 3 common methods are Start, Update and FixedUpdate, these are seen throughout our classes. Start is a method which is run once when a scene is entered, before the first frame. Update is a method that is called every single frame while the scene is running, FixedUpdate is like Update, but it is run every physics update (50 times per second), this is useful for when you do not want things to vary based on the players frames per second.

Below is a breakdown of all the class used in our project, along with the methods they contain and the behaviours they are responsible for.

All the scripts used in the project

5.1 Animation_Controller

This class is associated with the character model, within the “character” game object in SampleScene. The responsibility of this class is to display the correct character animation depending on the direction the character is moving in.

Methods:

Start – In this class the start method is used to set some components, first the variables for the character animator and rigidbody component from the character object are set, these are required for changing the animations in the animation controller (described in the animation section of this report) and getting the position of the character respectively. Finally, the initial position is set as the worldCentreOfMass of the rigidbody component.

FixedUpdate – This method is linked to the character controller class which tells the animation controller if the character is running, idle, or the game is over. This main function of this method is to update bool variable that work with the animation controller. There are 5 main bools that are toggled in the method isDead, isRunning, isMovingUp, isMovingDown and goingBackwards. The first check is to see if the game is over, if it is not, and the character is running the position of the rigidbody is used to determine if it is moving up, down or backwards and the corresponding bools are toggled. Finally, if the character is idle all the bools are set to false. FixedUpdate was used instead of Update, because at very high framerates the difference in positions was not large enough to trigger the conditional statements.

5.2 Background_Controller

This script is associated with the background in SampleScene, the main responsibility of this class is to repeat the background as the game continues, making sure that there is never a section of the game with no background.

Methods:

OnTriggerEnter – This method works by using a collider at the end of the background image, when the player reaches the collider a set amount of time is waited and the position of the background image is updated

5.3 Backstory_UI_Controller

This is a very basic script, it is associated with the backstory scene canvas, all it does is load the main menu scene when the user presses the “back to menu” button on the backstory scene. This functionality is within the OnExitButtonPressed() method.

5.4 Camera_Controller

This script is involved with moving the camera when the game is played. It is associated with the main camera object in SampleScene

Methods:

Update - *The* update function checks that the game is not paused, if it is not then the camera continues to move

FixedUpdate - The fixed update function is responsible for controlling the speed of the camera, the camera is incrementally increased as the game is being played.

5.5 Character_Controller

The character_controller class is one of the more important classes it is associated with the character object in SampleScene, its responsibility is to move the character through the pipes, as well as collaborating with the animation class. This class uses the checkpoints generated by each individual pipe and moves the characters position through the checkpoints, stopping if it reaches the end of the pipe.

Methods:

Start – The start method in this class is used to set the initial target position of the character, this is always the position at the end of the starting pipe.

Update – The bulk of the responsibilities of this class are covered in the update method. The first thing that happens is the characters current position is taken, then if the game is not over and the character has not reached its target then it is moved towards the target at a given speed. If the character has reached its target, but there are still checkpoints in the checkpoint queue then the characters' target is updated and it moves towards the new target, this is repeated until it reaches

the final target, where it stops until a new pipe/more checkpoints are added. The method collaborates with the animation controller to declare if the character is moving or idle.

OnTriggerEnter – this one of the default methods in the UnityEngine library, this method is called when the character's rigidbody component collides with another object that is toggled to be a trigger collision. For our project the possible trigger collisions are the death walls, obstacles and coins. If the object the character collides with is a death wall or an obstacle then the method interacts with the game state controller and animation controller to make the game be over, if the object is a coin, then the method interacts with the coin controller to update the coin count/score.

5.6 Coin

This is a simple script attached to the coin prefab (coin_updated), not the parent coin object (which is why it is separate from coins_controller). This class is responsible for rotating the coins continuously to give them a nice look. This functionality is within the Update() method.

5.7 coins_Controller

coins_Controller is attached to the coin game object in SampleScene. Its responsibilities are to generate, starting coins, continuously generate coins and interact with the score class when coins are collected.

Methods:

Start – In the start method calls the starting_Coins and generate_Coin functions.

coinCollected – This method is called from the character_controller when the character collides with a coin, the methods responsibility is to first increase the coin counts for the main menu and score classes, then it deletes the collected coin object from the game.

generate_Coin - This method is responsible for generating new coins as the game progresses, it generates the new coins using the position of the pipe_locator object which is just Infront of the players camera view.

starting_Coins - This method generates 3 initial starting coins, these are constant every time the game runs, they are generated so the player can quickly collect some coins if the play correctly this can be useful to save up for coin shop items.

5.8 CoinShop

This class is associated with the canvas of the CoinShop scene. Its main responsibilities are to allow the purchasing of score bonuses and multipliers, as well as updating the coin count and collaborating with the score class.

Methods:

Start – The start method updates the players coin count and high score at the top of the screen.

BuyScoreBonus – This method is called when the player clicks on the score bonus shop item, it checks if the player does not already own the score bonus (to avoid stack bonuses) and checks if they have enough coins, it then shows a popup text showing the outcome. If the item is purchased

the coins are removed from the player balance, the coin count text is updated and the score script is updated to include the bonus.

BuyScoreMultiplier – This method acts the same as BuyScoreBonus, but for the score multiplier shop item.

popupText – This method shows variable text to the player for 1.5 seconds based on the outcome of the button being pressed.

OnExitButtonClicked – When the player clicks the exit button they are returned to the main menu.

5.9 Controls_UI_Controller

This is a very basic script that is associated with the Controls scene. The script loads the main menu scene when the user presses the “back to menu” button on the control's scene. This functionality is within the OnExitButtonClicked() method.

5.10 Game_State_Controller

This script solely works to set the public game_State string to “play”. As a public string, this can be changed within other scripts, for example in the Game_UI_Controller script, game_State can be set to “pause” whilst the user has paused the game. This functionality is within the Start() method.

5.11 Game_UI_Controller

This script has numerous functions, all of which depend on the state of the game (game_State).

Methods:

Update – This method is called once per frame and obtains the state of the game play. If the game is over, the game over user interface is displayed, giving the user two options: restart or home. Otherwise, the user can continue to play the game.

OnHomeButtonClicked – This method displays the main menu if the home button is pressed on the game over screen.

OnPauseButtonClicked – When the user presses the pause button during the game, the pause screen is displayed, the game progress is halted, and the pause button is hidden from view. The user is shown three options on the pause screen: resume, restart, and exit.

OnResumeButtonClicked – This method allows the game to resume once the resume button is clicked on the pause screen. The pause screen is no longer shown, and the pause button reappears. The continuous pipe generation is restarted; however, the script makes use of the waiter() method.

OnRestartButtonClicked – Once the user presses the restart button on the pause screen, this script restarts the game. This includes resetting the score and the character appearing in the first pipe with new pipe, coin, and obstacle generations ahead. The pause screen is no longer shown, and the pause button reappears.

OnExitGameButtonClicked – This method loads the main menu when the exit button is pressed on the pause screen.

Waiter – This method reduces the likelihood of the obstacles, pipes, and coins overlapping when generated by offsetting the generation of each by 1, 2, and 3 seconds respectively.

5.12 Menu_UI_Controller

This script is like that of the Game_UI_Controller, however, revolves around the main menu.

Methods:

Start – The start method updates the players coin count and high score at the top of the screen.

OnStartButtonClicked – This method loads the SampleScene when the start button is pressed on the main menu. This allows the user to begin playing the game.

OnControlButtonClicked – This method loads the controls scene when the controls button is pressed on the main menu. This scene explains the functionality of each key used to navigate the game.

OnExitButtonClicked – If the exit button is pressed from the main menu, this method is entered and the game is exited.

OnBackStoryClicked – This method loads the backstory scene if the backstory button is pressed on the main menu. The backstory scene provides context for the game which may make it more engaging for the user.

OnCoinShopClicked – If the coin shop button is pressed on the main menu, this method is entered and the CoinShop scene is loaded.

5.13 Obstacle_Controller

This script is concerned with the generation of obstacles for the user to avoid during game play. This aims to increase the difficulty of the game.

Methods:

Start – This method starts obstacle generation by calling the generate_Obstacle method. This is done with a 1 second offset from the generation of coins to reduce any overlap of these icons on the screen.

generate_Obstacle – This method selects an obstacle at random from the obstacles array every 6 seconds. Each obstacle is placed at a random y-coordinate on the screen whilst the game is in play.

5.14 Pipe_Generation

This script creates, generates, and manages the flow of pipes throughout the game.

Methods:

Start – This method creates an array containing all the possible pipes used within the game. To generate the initial set of pipes, the start_Pipe_Generation method is called.

start_Pipe_Generation – This method generates six random pipe pieces that are placed in set positions on screen when the user begins the game. The generate_New_Pipe method is then called after two seconds to continue the pipe generation process for the rest of the game.

generate_New_Pipe – As stated above, this method generates new pipes for the remainder of the game whilst game play is active (i.e., the game has not been paused/lost). To ensure the pipes are spread out on the screen, the script alternates pipe placement between the top and bottom half of the screen.

5.15 Pipe_Properties

This script controls the highlighting, attachment ends, and rotation of pipes.

Methods:

Start – This method has one purpose, to turn off the highlight on the pipe pieces before the game starts.

Update – This method updates the strings that contain the information for where the start and end of the pipe piece are pointing, based on the pipe's rotation. This is updated every frame regardless of whether the pipe piece has been rotated or not.

pipe_Rotate – This method is called to rotate the selected pipe piece. The new rotation angle is stored based on the arrow keys pressed by the user, and the pipe holder is rotated to reflect this. The user is also allowed to change the attachment point for U-shaped pipes from left to right. This method accounts for this and highlights the new attachment point on the pipe piece and reverses/unreversed the pipe piece to reflect the change.

turnHighlightOn – This method has one purpose, to turn on the highlight on the selected pipe piece.

turnHighlightOff – This method also has one purpose, to turn off the highlight on the selected pipe piece.

5.16 Pipe_System

This script determines whether the pipe piece selected by the user can be added to the pipe system. If the pipe can be added, the script allows for its attachment and the subsequent movement of the character.

Methods:

Start – This method sets the initial position for the first pipe piece to be added.

add_Pipe – This method first checks whether the selected pipe piece can be added, then moves the piece to its starting position, then adds the checkpoints from the selected pipe to the checkpoints list to facilitate character movement through the pipe system. If the pipe has been reversed (i.e., the user has changed the attachment side of a U-pipe) then the pipe checkpoints are added in reverse order. The checkpoints are then added to the character's movement queue so that it will continue moving towards each position in sequence. The new direction of the end of the end of the pipe system is recorded and set to check if future pipe pieces may be added onto it. The tag of the newly attached pipe piece is changed to "attached_pipe" for reference by the pipes_Interface script.

5.17 pipes_Interface

This script handles user inputs for pipe rotations, attachments, and selections.

Methods:

Update – This method is called once per frame and clears the queue of generated pipes if the game is over. If the game is still in play and a user has just selected a pipe piece, then that pipe piece is highlighted on the screen and removed from the queue of generated pipe pieces. The selected pipe piece is then passed as a parameter to the onEnter() or onRotate() methods. If a user is de-selecting a pipe piece, then the highlight of the piece is removed accordingly.

onEnter – This method tests to see if the enter key has been pressed. If it has and a pipe piece is selected, then the pipe piece is added to the system.

onRotateInput – This method tests to see if the user has pressed either the left- or right-arrow keys to see if the pipe should be rotated. If so, then the pipe_Properties script is utilised.

5.18 SaveGame

This script manages the coin count and high score values that are displayed in the main menu and coin shop scenes.

Methods:

SaveGameData – This method saves a given coin count and high score form the menu_UI_Controller script.

LoadCoinData – This method loads the coin count for the main menu. If this is the first time the user is playing the game, a coin count of 0 will be displayed.

LoadHighScoreData – This method loads the high score for the main menu. If this is the first time the user is playing the game, a high score of 0 will be displayed.

5.19 Score

This script handles any updates to the score including bonuses/multipliers the user has purchased in the coin shop.

Methods:

Start – This method deals primarily with Boolean variables. The user can buy score bonuses of 100 points or a score multiplier in the coin shop from the main menu. If the user has bought either of these, the score bonus is stored and the scoreBonus variable is set to false, or the addMulti is set to true respectively in this method.

Update – This method is called once per frame. The user's score is updated to the furthest distance travelled by the character along the x-axis (note that the score will not increase/decrease if the character travels backwards). The score bonus stored in the *Start* method is added here if applicable, along with any coins the user has collected so far. The score is doubled in this method if the player has bought the score multiplier from the coin shop. The score text is then updated to reflect these changes.

6 Prefabs

In unity you often want certain game objects to be reusable, for this you can use prefabs. Prefabs are game objects that can be used for multiple instances this allows you to make changes to just the prefab which will in turn update all instances of it within the game.

For our game we used prefabs for 3 main components, the obstacles, the pipes and the coins.

The prefabs folder

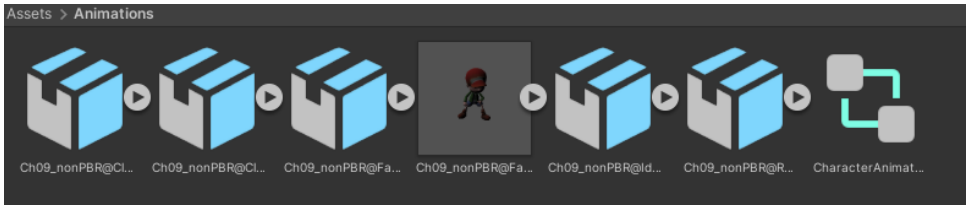
You can import prefabs from a number of different online sources, for example the obstacle prefabs were imported from the free unity asset store. You can also create your own prefabs, a useful program for this is the 3D modelling software called Blender.

We used blender to create all the pipes and the coins for our game. If you want to edit the current pipes, you can import them into blender and alter them as you please before reimporting them into the unity project. There are many useful online tutorials for getting started with blender, so no detail on how to use blender will be covered here, but I recommend checking out the source below if you are looking to create new models or update the existing prefabs.

Blender tutorials: https://www.blender.org/support/tutorials/

7 Animations

Another important part of our unity project was the character animations, these allow the character object to display different movements depending on its current actions/movement. The animations are located in the animation folder of the project, this is also where the character model is located and an animation controller.

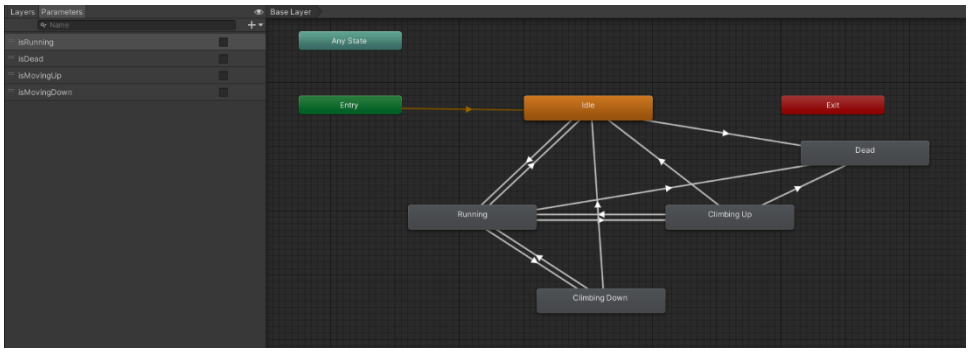


The animations folder

To use animations, you need to have your character model and animations have the same type of rig, in our case that is a humanoid rig. Although you can make custom animations for you character, there are online resources that are both free and copywrite free. For our project we took advantage of the character model and animations from mixamo.com.

The animation controller (seen on the right of the above picture) is what controls the animation state of the character. It works by setting the animation based on specific Boolean scripts set in the

animation_controller script, depending on the Booleans the character transitions between different animations, the animation controller is shown below.



The animations controller

8 Known Bugs

8.1 Slight animation glitch on the long straight pipe

When the player connects a long straight pipe to the end of the main pipe there is a slight bug which causes the character to move backwards for a few frames before moving forwards, this is due to the checkpoint positions of the pipes when they join, and the starting checkpoint not being exactly at the point where the pipe joins, but instead being behind it, this bug should be easily fixable.

8.2 The game can is only suited for 1080p resolution

In all our scenes in the game we have used Text objects in the UI, these objects change size and position based on the resolution and aspect ratio the player is in. So, there is currently a bug where the text all appears in the wrong places if the game is played in anything other than 16:9 1080p resolution, we have tried to get around this bug by force setting the game resolution to 180p when the game is launched, but it would be nice to find a hard fix for the bug that allows the game to be played in multiple resolutions.

8.3 The pipes do not fit together perfectly

There is still a small visual bug when connecting the pipes together because some of the ends of the pipes do not match up perfectly, so the pipes appear a bit off, this bug should be fixable by altering the sizes of the pipe prefabs in the pipe holder game objects so they are all the same dimensions and position for pipe entry and exit.

8.4 Objects can overlap

Currently there is nothing to stop the pipes, coins and obstacles generating too close to each other and causing them to overlap, we have reduced this issue by offsetting the spawn time of coins an obstacles and alternating the pipe location between high and low, but there are still some occurrences where objects are overlapping which doesn't have a good visual effect, a solution for this problem could be giving the obstacles colliders and if the colliders are touching then one of the game objects is removed.

Another issue with pipes being able to overlap is the fact that the player can go back on themselves and overlap the main pipe, while this does not necessarily break the game it was not an intended function, therefore t could be a nice task to make the character die if it touched its previously produced pipe.

9 Future development ideas

9.1 Add different pipe shapes

One easy addition to the game is the addition of more pipe shape, you could look at adding pipes with multiple exit/entry points, for example a T shape pipe. The user would then need some way of deciding which path they take through the pipe.

An addition to this would be generating pipes at different rotations.

9.2 Add pipe features

A feature we would have loved to add but did not have time to was the addition of pipes with different characteristics. This would allow more variation to the game. Some examples of pipe characteristics could include frozen pipes which slow down the player, pipes which speed the player up and locked pipes which cannot be rotated

9.3 Add Beat Bonus

One of our original ideas was to add a musical aspect to the game, the idea of the beat bonus would be that if a player added a pipe in time with the tempo of music, then they would get a bonus added to their score.

9.4 Make backstory animated/scrollable

There is plenty of room for improvement for the backstory of the game, this could simply be making the backstory scene a scrollable story or even an animated movie scene. You could also add some way of the player uncovering more about the story as they progress through the game.