A Virtual Reality App to Interact with a 2D Arcade Game

Harry Suggett

Supervisor: Dr Yukun Lai

CM3203 One Semester Individual Project – 40 Credits

Cardiff University School of Computer Science & Informatics

28th May 2021

Abstract

In this project, the aim was to create a virtual reality (VR) app that could demonstrate a novel method of interacting with traditional 2D video games in a 3D VR environment. To achieve this, I created both a standalone 2D game to represent a typical side-scrolling platform game, and a 3D environment in which the player can interact with the 2D game using a VR headset and controllers.

This project has 3 main objectives:

- Create a 3D VR environment in which the arcade machine sits.
- Create a base 2D game to be displayed on the virtual arcade screen.
- Allow a user to interact with both the 2D and 3D environment by using the VR controllers to pick up the 3D blocks and interact with the 2D screen.

In this report, I will detail the development and evaluation of the features comprising this app.

Acknowledgements

I would like to give acknowledgement to my project supervisor, Dr Yukun Lai, for his continued support and guidance on this project.

I would also like to acknowledge all the participants who completed my user questionnaire, as their feedback was invaluable in evaluating the app.

Finally, I would like to acknowledge Unity for the use of their game engine. Without the use of the Unity game engine and its components, developing this application would have been extremely complex and out of scope due to the available timeframe.

Contents

Table of Figures

Introduction

In this project, the goal was to create a virtual reality (VR) app that could demonstrate a novel method of interacting with traditional 2D video games in a 3D VR environment. To achieve this, I created both a standalone 2D game to represent a typical side-scrolling platform game, and a 3D environment in which the player can interact with the 2D game using a VR headset and controllers.

The 2D game was designed to be reasonably simple, with only basic enemies and a final goal at the end of the levels. This was because the main purpose of the 2D game was to serve as the basis for the interaction within the 3D environment.

The 3D environment consists of an arcade machine in a room. This arcade machine has a screen to display the 2D game. The user can interact with the 2D game by picking up blocks and pressing them into the screen. These blocks then interact with the 2D game, allowing the player to jump on them and complete levels.

This concept of interacting with a 2D game in a 3D space using VR is very novel and during research, I could not find any prior examples of such a game. Implementing this interaction will require translating the location of the block in the 3D space, into the 2D space of the game. In Unity 2D games are actually in a 3D space, along a single axis, so this will add complexity to the solution.

The app is aimed at a target audience that already plays regular 2D or 3D games since the design and control scheme should be familiar to those who have played a 2D platformer game before. The target age demographic is anyone over the age of 13 as this is the age that Oculus recommends as a minimum for their VR headsets.

The players of the game will need a basic understanding of how to install and run a VR application, but this is the same for any VR app and no more effort is required than for any other VR application readily available to download from the internet. The controls are simple and should not pose any significant learning curve to someone who has played a VR game in the past. However, if they have never used a VR headset before, then this may require some period of learning and in which case an introduction tutorial such as Oculus's "First Steps" should give most people enough of an introduction to the controls for them to use the app successfully.

I developed the app to be used with an Oculus headset, specifically the Oculus Quest 2 running from a PC. However, most VR headsets would work with this app provided it had enough buttons on the controllers. The app does not require any walking around, as the virtual arcade machine is presented in front of the user. Despite this, most VR headsets do have a minimum play area when used in a 'Roomscale' mode (typically 2x2 metres). Oculus does, however, allow for a stationary play mode which would be suitable for this app.

This project has 3 main objectives:

- Create a 3D VR environment in which the arcade machine sits.
- Create a base 2D game to be displayed on the virtual arcade screen.
- Allow a user to interact with both the 2D and 3D environment by using the VR controllers to pick up the 3D blocks and interact with the 2D screen.

The project Is only intended to be a proof-of-concept to demonstrate the method of interaction with the 2D game within VR, but the game should function well and meet all objectives, with potential for future improvement and expansion of the concept.

Background

Virtual Reality

The concept of virtual reality (VR) was arguably not invented by any one person but rather created by a series of advancements in engineering and technology. The term “virtual reality” itself was popularised by Jaron Lanier of VPL Research in 1987[1], however, there were many developments into stereographic 3D images and videos before this time, which could be considered early examples of the technology:

In **1838** Sir Charles Wheatstone described stereopsis, which is the perception of depth resulting from receiving 2 images, one from each eye[2]. From this discovery, Wheatstone created an early type of stereoscope which could display static 3D images using a pair of mirrors at 45 degrees[3].

In **1935** the American science fiction author Stanley Weinbaum described a pair of goggles in his book Pygmalion's Spectacles, which enabled "a movie that gives one sight and sound ... taste, smell, and touch. ... You are in the story, you speak to the shadows (characters) and they reply ... the story is all about you, and you are in it."[4]. This description very closely matches the current model of a VR headset, albeit typically without the senses of taste, smell, and touch.

In **1956** Morton Heilig invented his “Sensorama”, which was a booth designed to stimulate the senses using 3D video, stereo sound, vibrations, scent producers and fans[5]. Heilig patented the device in 1962 and went on to develop the Telesphere Mask[6], which was the first head-mounted display (HMD), an important precursor to modern HMDs.

In **1979** McDonnell-Douglas Corporation added head tracking into its HMD, the VITAL helmet, designed for military use. This head tracking followed the pilot’s eye movements so the display could match the computer-generated images[7]. This is akin to how modern VR headsets track your head movements.

Jaron Lanier and Thomas Zimmerman founded VPL Research in **1985**. This company was the first to sell VR goggles and gloves[1]. Jaron Lanier was the person credited with popularising the term “Virtual Reality” while working at VPL Research. VPL Research’s VR and graphics patents were later purchased by Sun Microsystems in 1999 after the company declared bankruptcy in 1990[8].

Investment into VR technology somewhat stagnated until **2010**, when Palmer Luckey created the first prototype of the Oculus Rift headset. The prototype had a much larger field of view than previous headsets and used a PC to deliver the images[1]. In 2012 Palmer Luckey launched a Kickstarter crowdfunding campaign for his Oculus Rift headset, which raised over 2.4 million dollars[9].

In **2014** Facebook bought Oculus for 2 Billion Dollars[1], this illustrated the massive increase in funding for VR at the time, and may now be seen as a relatively low price for the company.

Since then, VR has progressed both in outright technological advancements and in reducing the cost of entry into the tech. Today you can buy a headset such as the Oculus Quest 2 for $299, a huge decrease from the initial cost of even the original Oculus Rift headset at $599 at launch. Modern headsets come with tracked controllers, and many do not require tracking cameras at all, unlike the original Oculus Rift, and rather use an inside-out tracking system. The modern headsets have also increased in both resolution and screen refresh rates, giving a more immersive experience.

The VR market size is projected to rise to over $100 billion by 2027. This increased investment into the technology has been further fuelled by the Covid-19 pandemic. With less social contact permitted, or even wanted anymore, people are looking to new technology to socialise online and pass their time indoors.

VR has many applications, although now if you ask someone what they think VR is for, they are likely to say “gaming”. This is not the only use case, however, as there are almost limitless applications for VR including fitness, teaching, healthcare and communication.

VR has two levels of tracking: 3dof (3 degrees of freedom), where the rotation of the headset is tracked, and 6dof (6 degrees of freedom), where the rotation and position of the headset are tracked. My app is tracked to 6dof, as this is important for the interaction method I aimed this app at.

VR also has the potential to change the way people think about sensitive topics. One study quoted: “put a buzzing joystick in participants’ hands, mimicking a chainsaw as their virtual hands sawed down a tree. Afterwards, when an experimenter pretended to accidentally knock over a glass of water, those who had sawed down a virtual tree reached for 20 percent fewer napkins than those who only read a passage describing a tree being cut down”[10]. This shows that VR has the potential to make a positive, or even negative, impact on the way people think about their actions.

I chose to use VR to create this app, as I believe it gives a unique opportunity to introduce people to different ways of interacting with something they may be quite familiar with, such as a 2D platformer game. Almost everyone has played a game like Super Mario Bros, but people may not have thought about interacting with such a game in a VR environment.

Game Engine - Unity

To develop this app, I used the game engine Unity. Unity is a game engine that allows cross-platform development, frequently used to create mobile and desktop games. It was first released in 2005 and supports multiple platforms such as PC, Consoles, VR and mobile.

Unity allows you to develop games quickly, by handling most of the graphics in the graphical user interface (GUI), leaving more time for coding, at the expense of some amount of freedom in terms of low-level access to the graphics APIs. Unity works by coding scripts and attaching them to game objects in the unity scene. These scripts are now written in C# in Unity, so this is the programming language I use in this project. Unity used to also support scripts in UnityScript (JavaScript) and Boo, though support for these was deprecated.

I chose to use the Unity game engine, as it allowed me to focus on the functionality of the app, rather than spending a large amount of time coding low-level graphics functionality in C. As the project has a timescale of 12 weeks, not using a game engine was not feasible due to the time which would have been needed to code the solution.

Unity also has built-in support for developing VR and other mixed reality applications through their XR (Extended Reality) plug-in framework, which gives native support for platforms such as Oculus, PlayStation VR, and Windows Mixed Reality[11]. I used this plug-in to allow me to develop for the Oculus platform, although it would be trivial to make builds for other platforms supported by this plugin, as I only used controls that are available on most headsets. Additionally, the third-party OpenVR Unity Plug-in could be utilised to support different headsets through the SteamVR platform.

I decided to use the Unity version 2019.4.20f1 as it was a long-term version that didn't mandate the use of Unity's new input system, which was less well documented online due to its recent release.

C#

Unity uses the C# language for scripting. C# is an object-oriented programming language developed by Microsoft as part of the .NET framework alongside Visual Basic and F#. C# uses a syntax similar to other C style languages such as C++ and Java.

Existing Solutions

I have found a few examples of existing solutions that use a similar concept of a 2D game being projected into a 3D world as in my app, although none use VR to actually interact with a 2D game using tracked controllers.

One of these is Tetris Effect, which has a VR version[12]. This takes an ordinarily 2D game of Tetris and places it into a 3D VR environment.

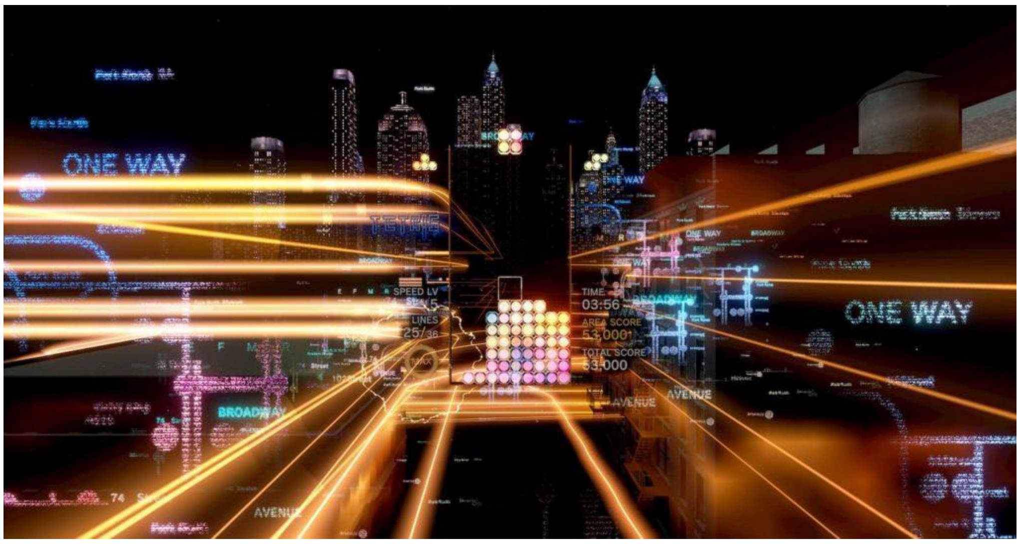


Figure 1 - Tetris Effect

In this example, the VR aspect doesn't add anything to the gameplay, rather it just adds to the immersion of the 2D game by placing it into a VR environment. This could be seen as just a port of an existing game into VR, rather than using a novel way to interact with the game, such as turning the Tetris blocks with your hands in VR.

Another example of a 2D game being projected into 3D is Super Mario Odyssey[13]. In this 3D game, there are sections where you can play 2D minigames projected onto walls in the 3D world. This adds variety to the gameplay by using nostalgia from old 2D Mario games.

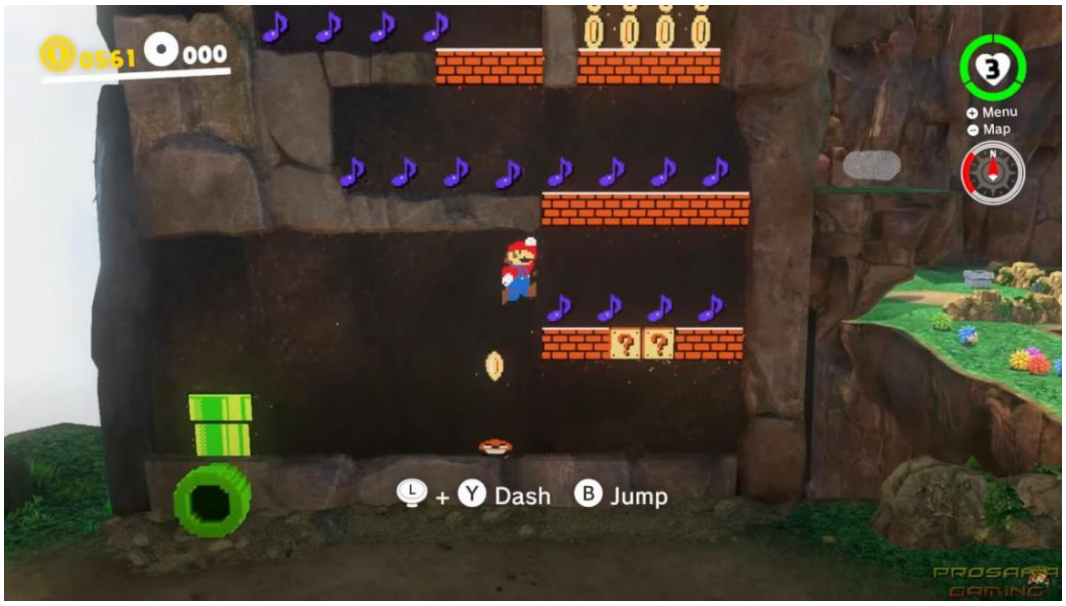


Figure 2 - Super Mario Odyssey

Specifications and Design

Requirements

For this project, I had several requirements that must be met for the app to be developed to a good standard and meet my overall aims. These requirements can be split into 3 groups based on my 3 main objectives for the project as outlined in the introduction.

3D VR Environment

The first objective was to create a 3D environment that the user can be in when using the app with a VR headset. This allows the user to get a sense of presence and immersion as if they are in an arcade playing on an arcade machine. The 3D environment would need to meet these requirements:

1. There must be a simple room to contain the arcade machine and any other background objects.
2. There must be a camera that follows the user’s head movements in VR, both in rotation and position.
3. The room must have some form of lighting.
4. There must be objects that follow the user's hand movements in VR, tracked using the VR controllers. These could be hand models, controller models etc.
5. The room must have a simple arcade machine or another model to house a screen to display the 2D game.
6. There must be object(s) that will be used to interact with the 2D game, for example, bridges, blocks, or stairs.
7. The room must have a surface or location to house these block(s) near to the screen.

2D Side-Scrolling Game

The next objective was to create a 2D game that the user would play on the virtual screen. This game will work standalone, but the levels are designed around the VR interaction which is implemented as part of the final objective. The 2D game will need to meet these requirements:

8. The game must have a player object and texture.
9. The game must have a background. This could be a static or dynamic image.
10. There must be obstacles for the player to avoid.
11. There must be terrain for the player to walk around on.
12. The game must have enemies for the player to avoid or kill.
13. The enemies must deal damage to the player if they come into contact with them.
14. The enemies must move on their own.
15. The enemies must be able to be killed by jumping on them.
16. There must be a camera that follows the player around the levels.
17. There must be an end goal at the end of the level(s) where once the player touches them, it ends the level.

18. At the end of the level, the game must display the user’s score and go back to the main menu
19. A main menu to choose which level to play.
20. A simple control scheme that is easy to use with VR controllers.
21. A user interface (UI) to show the player’s health, score and remaining time.

VR interaction with the 2D game

The final main development objective was to display the 2D game projected onto the screen in the 3D environment and to implement interacting with the 2D game by placing objects into the screen with their hands. To complete this objective these requirements need to be met:

22. The 2D game must be projected onto the screen in the 3D environment.
23. There must be a system to translate the location of 3D blocks that are moved into the screen using VR from the 3D space on the virtual screen to the 2D space of the 2D game so that they can interact with the 2D game.
24. When the camera moves after a 3D block has been placed, the block placed into the screen must follow the apparent location of the 2D block based on the camera movement.
25. When a block is moved by the above feature and reaches the edge of the screen, it must be removed from the screen and return to its original location in the environment.

Non-essential Functionality or Extensions

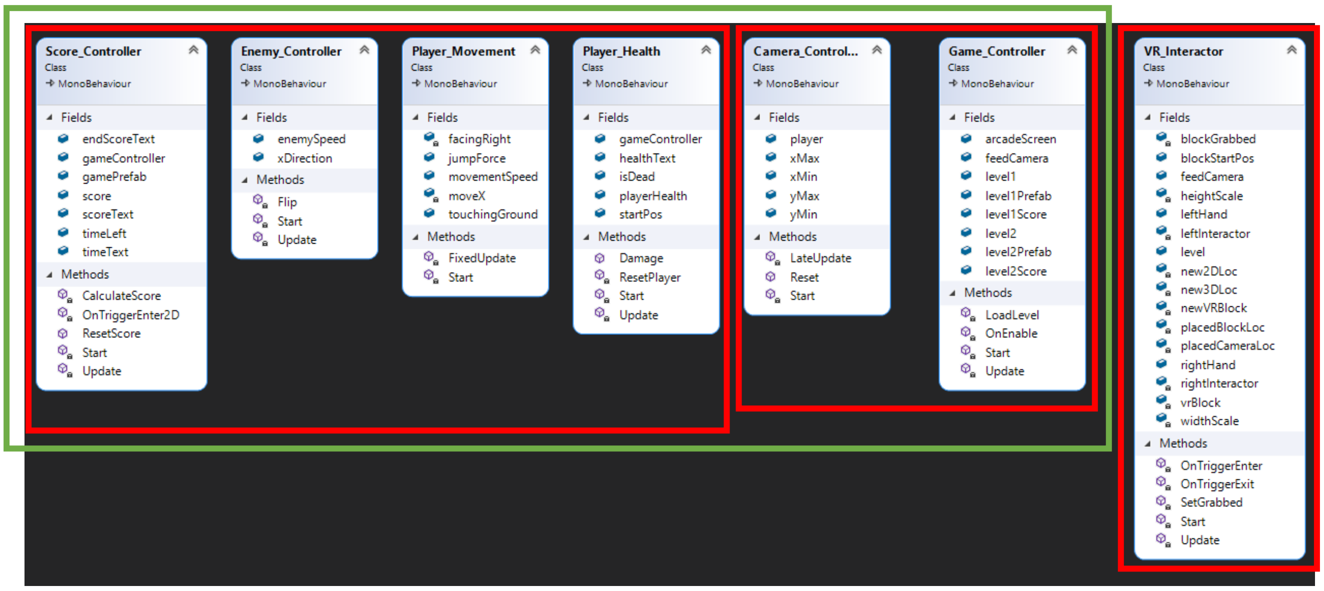
In addition to the main requirements, some functionalities could be desirable to implement but are not necessary for the project to reach its aims. These are:

1. Animations within the 2D game, for example, sprite animations for the enemy and player when they walk, jump, and take damage.
2. Sound effects when actions happen in the 2D game. This would give the player some more feedback within the game.
3. Multiplayer capabilities. Perhaps allowing 2 players to compete at the same time, or one player to deal with the game and one to assist with the 3D blocks.
4. The ability to pause and leave games once a level has started.
5. A system to keep track of high scores on each level.
6. Multiple different levels.
7. Collectable items such as coins in the levels.

Modules

The stages of the project development can be split into different sections or modules based on their functionality in the app. These broadly fall into two groups: the 2D game and the 3D environment with the VR interaction.

Figure 3. below outlines all of the modules in the project.



Score_Controller
Class
→ MonoBehaviour
- Fields
  - endScoreText
  - gameController
  - gamePrefab
  - score
  - scoreText
  - timeLeft
  - timeText
- Methods
  - CalculateScore
  - OnTriggerEnter2D
  - ResetScore
  - Start
  - Update

Enemy_Controller
Class
→ MonoBehaviour
- Fields
  - enemySpeed
  - xDirection
- Methods
  - Flip
  - Start
  - Update

Player_Movement
Class
→ MonoBehaviour
- Fields
  - facingRight
  - jumpForce
  - movementSpeed
  - moveX
  - touchingGround
- Methods
  - FixedUpdate
  - Start

Player_Health
Class
→ MonoBehaviour
- Fields
  - gameController
  - healthText
  - isDead
  - playerHealth
  - startPos
- Methods
  - Damage
  - ResetPlayer
  - Start
  - Update

Camera_Control...
Class
→ MonoBehaviour
- Fields
  - player
  - xMax
  - xMin
  - yMax
  - yMin
- Methods
  - LateUpdate
  - Reset
  - Start

Game_Controller
Class
→ MonoBehaviour
- Fields
  - arcadeScreen
  - feedCamera
  - level1
  - level1Prefab
  - level1Score
  - level2
  - level2Prefab
  - level2Score
- Methods
  - LoadLevel
  - OnEnable
  - Start
  - Update

VR_Interactor
Class
→ MonoBehaviour
- Fields
  - blockGrabbed
  - blockStartPos
  - feedCamera
  - heightScale
  - leftHand
  - leftInteractor
  - level
  - new2DLoc
  - new3DLoc
  - newVRBlock
  - placedBlockLoc
  - placedCameraLoc
  - rightHand
  - rightInteractor
  - vrBlock
  - widthScale
- Methods
  - OnTriggerEnter
  - OnTriggerExit
  - SetGrabbed
  - Start
  - Update

Figure 3 - Module Diagram

In figure 3, all the modules in the left red box are exclusively for the 2D game, being related to the scoring, enemy movement, player movement and player health. These modules are universal for all levels in the game and can be reused in each level.

The modules in the middle red box can be considered as part of the 2D game but are independent of the individual levels of the game, and handle components such as the camera and main menu.

Finally, the red box on the right contains only the module that handles the VR interaction with the 2D game using the 3D blocks. This module is again, independent of the 2D levels.

Design

Main Game with VR Interaction

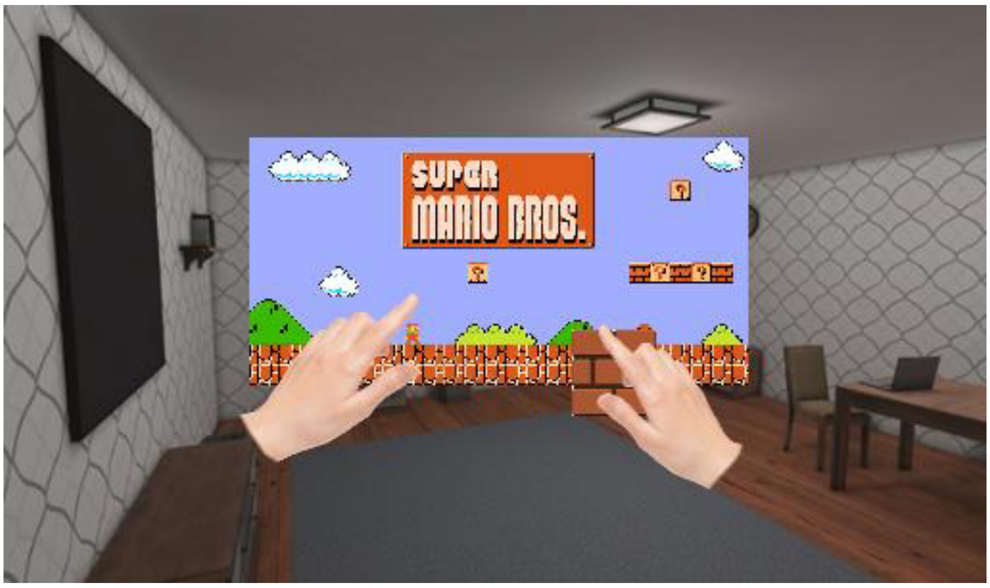


Figure 4 – Main Game with VR Interaction

Figure 4. above shows a mock-up of how I expected the 2D screen and VR interaction to look, as outlined in my initial plan. The mock-up shows the virtual screen with the 2D game projected onto it. It is also possible to see the hand models tracked in the space, and they are grabbing a block that is to be used in the game.

Main Menu

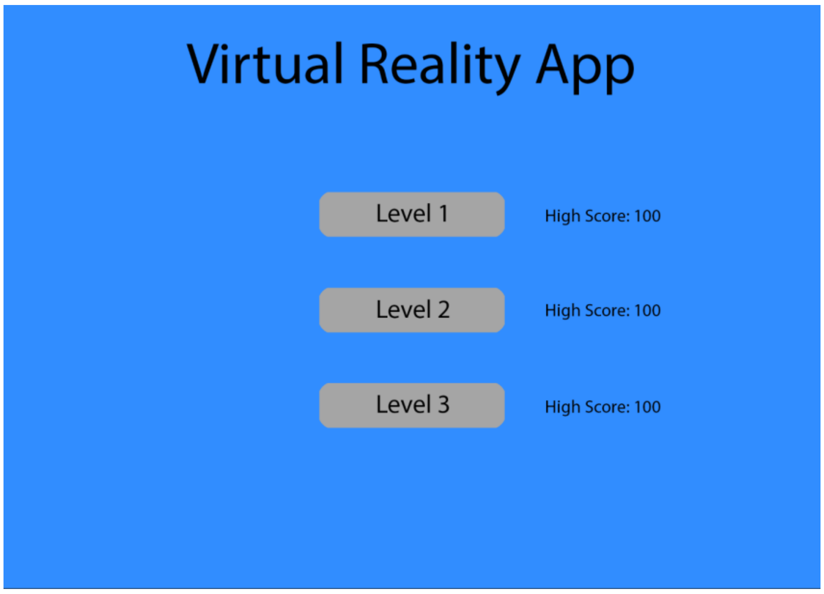


Figure 5 - Main Menu

Figure 5 above details a mock-up created of the main menu that should allow the user to select levels and view the high scores for each level. In this mock-up, the multiple levels and high score display fall under the non-essential features, while the rest of the functionalities are classed as main requirements (main menu).

Implementation

In this section, I will describe how the project is implemented. I will focus on the sections of code that are non-trivial or are critical to the functionality of the project. I will aim to explain briefly how each of these sections works, but an understanding of C# and Unity may be necessary to understand some sections. Explaining these sections is outside the scope of this project, however.

Assets

During the development process, I required the use of a few 3D and 2D models, as creating these myself would have been highly complex and taken too much time out of the already limited time frame for the project. These assets were all from the Unity Asset Store[14], and all allowed free use.

The first asset I used was “Arcade Machines Pack 01 - Lowpoly Pack” by AurynSky[15]. This asset pack includes the 3D model of the arcade machine which I used in my scene.

The next asset I used was “Hero Knight” by Luiz Melo[16]. This was a 2D sprite pack that I used as the basis for the player and enemies.

Finally, I used “Free 8-Bit Pixel Pack” by Super Icon Ltd[17], which was a pack of 2D tiles which I used to create the 2D levels.

3D VR Environment

The first step in development was to create a 3D environment for the user to be placed in and be able to look around in VR.

To do this I started a new Unity project using the new Universal Render Pipeline (URP). This is Unity’s replacement for the old Built-in Render Pipeline. I ended up running into issues with using this render pipeline that I will detail later.

Starting a project with the URP automatically loads in a sample scene with some sample assets, so I created a new empty scene for my game.

I added a floor using a plane and added the model of the arcade machine[15]. I then placed a cube next to the arcade machine to act as a table for the interactive blocks to be held on. Next, I added some textures to the floor and the table using some of the materials from the URP sample scene. Finally, I added a directional light above the scene to provide some extra local light in addition to the skybox lighting.

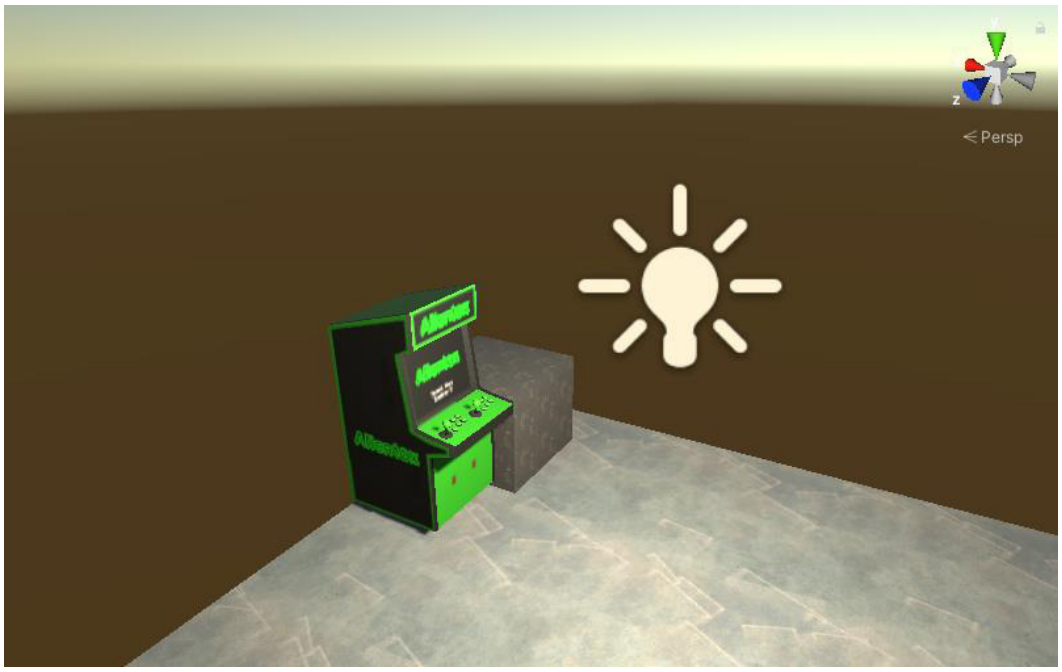
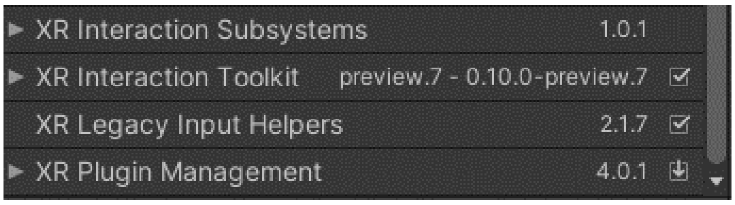


Figure 6 - Scene setup

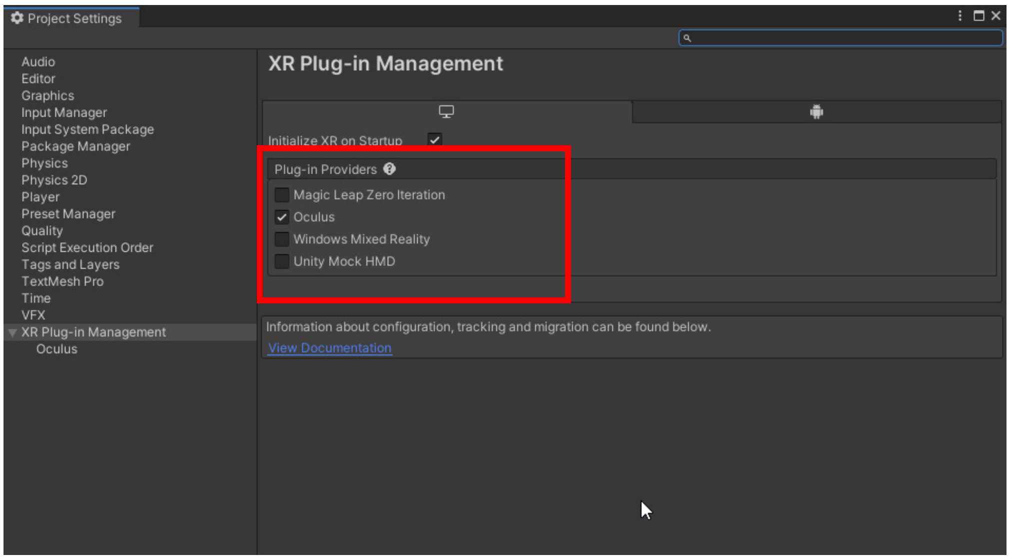
The next step was to set up the Unity packages/plugins and project settings required to enable the VR integration with the Oculus headset. First I installed the XR Interaction Toolkit and XR Plugin Management packages, which enable VR interaction and allow me to select which XR plugins to use, respectively. These are the XR packages that are now installed:



► XR Interaction Subsystems	1.0.1	
► XR Interaction Toolkit	preview.7 - 0.10.0-preview.7	☑
XR Legacy Input Helpers	2.1.7	☑
► XR Plugin Management	4.0.1	⤓

Figure 7 - XR Packages

After this, I set up the XR Plugins in the Project Settings panel to allow integration with the Oculus Plugin:


Project Settings

Audio
Editor
Graphics
Input Manager
Input System Package
Package Manager
Physics
Physics 2D
Player
Preset Manager
Quality
Script Execution Order
Tags and Layers
TextMesh Pro
Time
VFX
XR Plug-in Management
Oculus

XR Plug-in Management

Initialize XR on Startup ✓

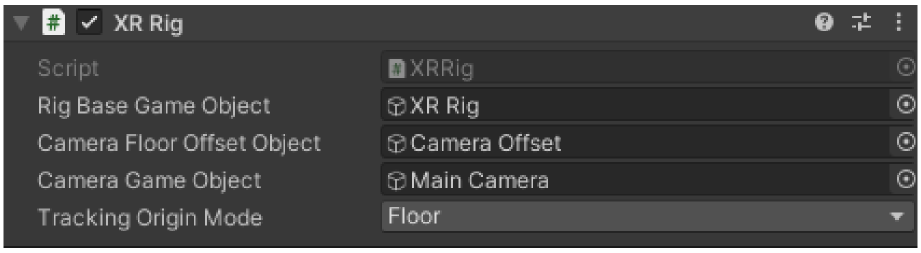
Plug-in Providers

- [ ] Magic Leap Zero Iteration
- [x] Oculus
- [ ] Windows Mixed Reality
- [ ] Unity Mock HMD

Information about configuration, tracking and migration can be found below.
View Documentation

Figure 8 - XR Plugins

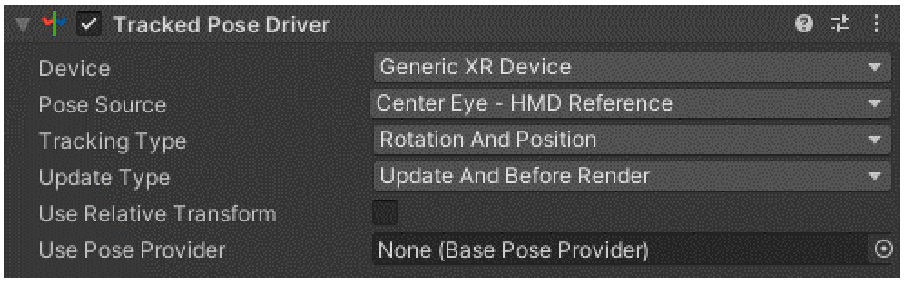
With Unity now set up for VR integration, I then converted the main camera in the scene to a device based XR rig. The XR rig will contain the VR camera as well as the 2 controllers.


XR Rig

Script	XRRig
Rig Base Game Object	XR Rig
Camera Floor Offset Object	Camera Offset
Camera Game Object	Main Camera
Tracking Origin Mode	Floor

Figure 9 - XR Rig

This also sets up the Main Camera to track the headset's rotation and position (6dof) using a Tracked Pose Driver in the camera.


Tracked Pose Driver

Device	Generic XR Device
Pose Source	Center Eye - HMD Reference
Tracking Type	Rotation And Position
Update Type	Update And Before Render
Use Relative Transform	
Use Pose Provider	None (Base Pose Provider)

Figure 10 - Tracked Pose Driver

This takes the position of the centre of the XR HMD and applies it to the camera this Tracked Pose Driver component is attached to. At this point, the VR camera works and you can look around and move correctly in the scene.

Next, I added models to represent the controllers and track them as well. To do this I created two coloured spheres as prefabs and added them to an XR Controller (Device-based) component within both the left and right-hand controller game objects within the XR Rig.

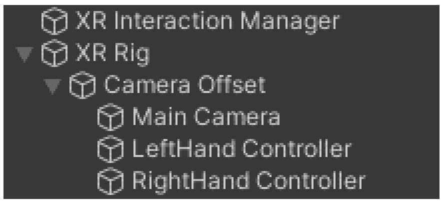
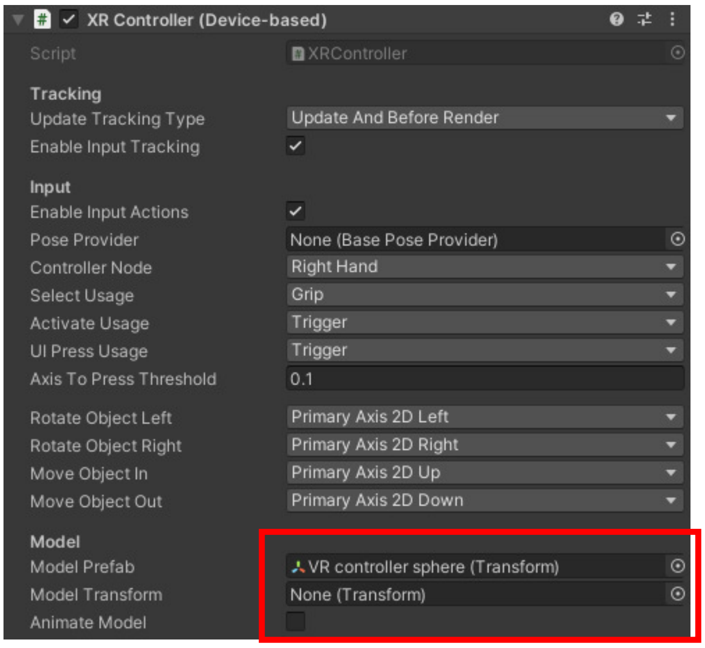

XR Interaction Manager
XR Rig
Camera Offset
Main Camera
LeftHand Controller
RightHand Controller

Figure 11 - Finished XR Rig


XR Controller (Device-based)

Script	XRController
Tracking	
Update Tracking Type	Update And Before Render
Enable Input Tracking	✓
Input	
Enable Input Actions	✓
Pose Provider	None (Base Pose Provider)
Controller Node	Right Hand
Select Usage	Grip
Activate Usage	Trigger
UI Press Usage	Trigger
Axis To Press Threshold	0.1
Rotate Object Left	Primary Axis 2D Left
Rotate Object Right	Primary Axis 2D Right
Move Object In	Primary Axis 2D Up
Move Object Out	Primary Axis 2D Down
Model	
Model Prefab	VR controller sphere (Transform)
Model Transform	None (Transform)
Animate Model	

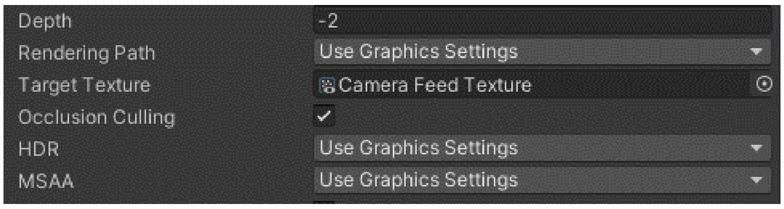
Figure 12 - XR Controller

At this point, the camera follows the VR headset's position and rotation, and the 2 spheres follow both controllers' location and rotation.

Virtual Screen

At this stage, I decided to set up the virtual screen, as I wanted to be able to develop the 2D game with the camera and screen in mind. To do this I created a second camera called the "Feed Camera" and placed it at 40, 2, -10 coordinates. This was because I intended to place the 2D game along the x-axis starting at 40, 0, 0. The camera was set up as an Orthographic camera, so there is no perspective to the image.

To display the Feed Camera's output on a virtual screen I needed to create a material that would go on the plane(screen). I first created a new texture called "Camera Feed Texture". I then set up the Feed Camera to target this texture as an output.



Depth	-2
Rendering Path	Use Graphics Settings
Target Texture	Camera Feed Texture
Occlusion Culling	✓
HDR	Use Graphics Settings
MSAA	Use Graphics Settings

Figure 13 - Camera Feed Texture

Next, I created a new material "Camera Feed Material" and added the texture to the material. Finally, I added this material to a plane I had created within the arcade machine.

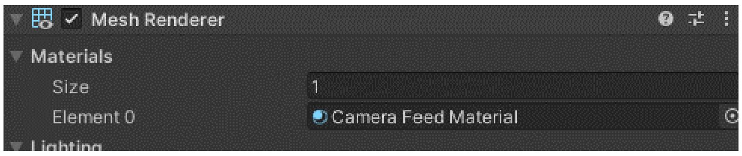

Mesh Renderer
Materials
Size 1
Element 0 Camera Feed Material
Lighting

Figure 14 - Camera Feed on Plane

At this point, I expected this to show the output of the camera, but it did not work at all. I searched for solutions to this online and I did find one person with a similar problem when using the new URP. Without the prospect of identifying a solution, I finally decided to change the project to the Built-in Render Pipeline. This was due to the fact that I concluded there was a bug in the URP, and I did not have a sufficient timeframe to attempt updating my version of Unity to see if that resolved the problem. I did this by altering the graphics settings in the project settings menu to remove the URP asset, which reverts it to the Built-in Render Pipeline.

After switching to the Built-in Render Pipeline, the screen now displays the output from the camera.

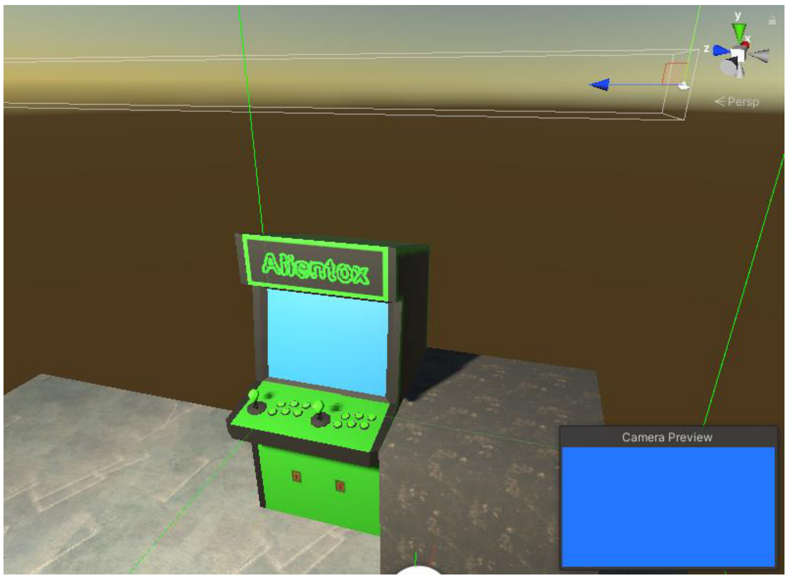


Figure 15 - Camera Feed Working

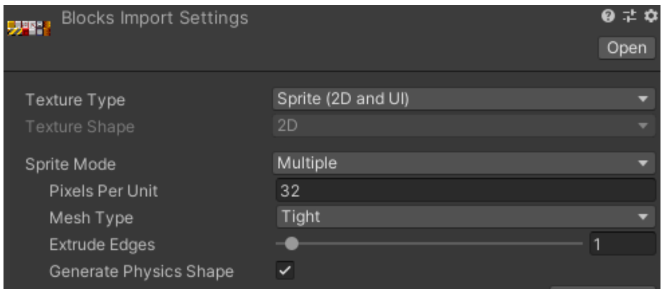
2D Game

The next step was to work on creating the 2D game that the app would be based around.

Level Tilemap

The first step towards the 2D game was to create a surface for the level. To do this, I made use of the tiled pixel assets[17]. To create a level from these tiles, I needed to use Unity's tilemaps. Tilemaps allow you to add tiled assets onto a grid and allow the creation of colliders for these tiles.

Before creating the grid, I needed to alter the block asset's import settings to change the Pixels Per Unit. This tells Unity how many pixels each tile takes up. Without this set, the tiles would be too small for the default grid size and only fill half of a grid square. In my case, this was 32 pixels.


Blocks Import Settings

Open

Texture Type: Sprite (2D and UI)

Texture Shape: 2D

Sprite Mode: Multiple

Pixels Per Unit: 32

Mesh Type: Tight

Extrude Edges: 1

Generate Physics Shape: ✓

Figure 16 - Tile Import Settings

With this set, I added a Tilemap grid and used Unity's Tile Pallet to draw a simple level base called "ground" as shown in figure 17 below.

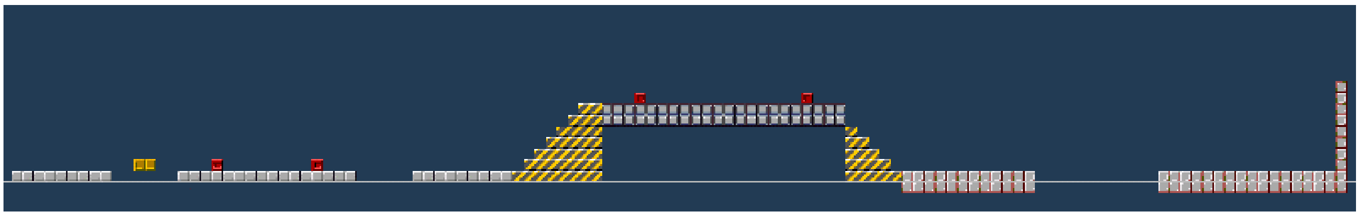


Figure 17 - Level Tilemap

Unity automatically adds a Tilemap Collider, which adds colliders for every block. However, later in development, I noticed that the player would catch slightly on every ground block while running along a flat surface. I detail my solution to this problem in the Bug Fixing section at the end of the Implementation.

Player

To create the player I created a new game object and added a sprite from the Hero Knight asset[16]. I used the "idle 0" sprite, as this would look acceptable without any animation. I added the necessary rigidbody and collider to allow physics and collision. I also constrained the rotation of the rigidbody around the Z-axis, as I did not want the player to rotate at all.

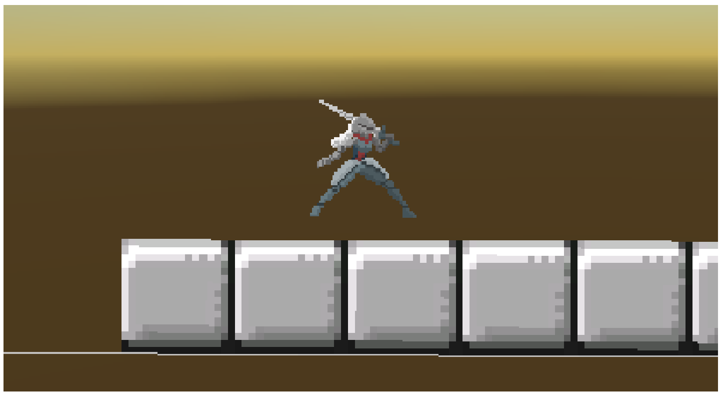
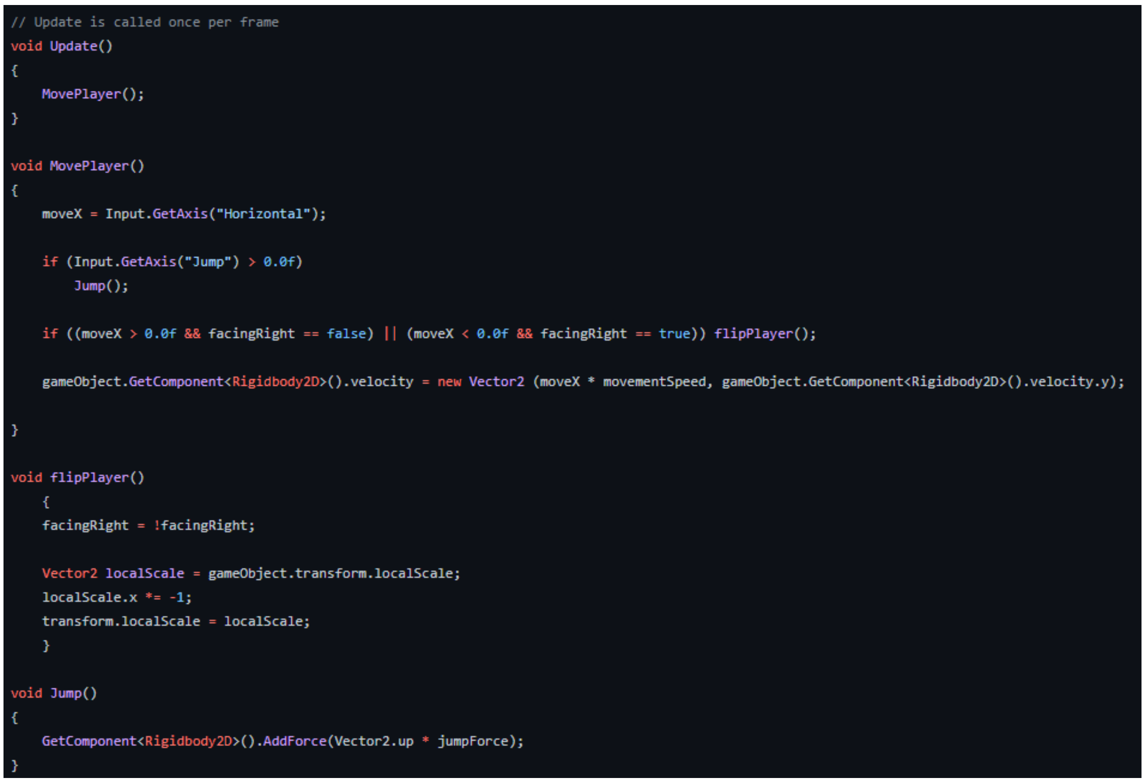


Figure 18 - Player Sprite

Next, I needed to make the player move, so I created a new script "Player_Movement" and attached it to the player game object. Attaching a script to an object in Unity means that the script runs when the player is active, and an instance of the class is created for each of these objects. Therefore, if there were more than one player, it would create a new instance of Player_Movement for each player.



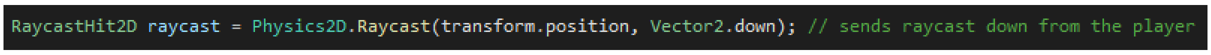
```
// Update is called once per frame
void Update()
{
    MovePlayer();
}

void MovePlayer()
{
    moveX = Input.GetAxis("Horizontal");

    if (Input.GetAxis("Jump") > 0.0f)
        Jump();

    if ((moveX > 0.0f && facingRight == false) || (moveX < 0.0f && facingRight == true)) flipPlayer();

    gameObject.GetComponent<Rigidbody2D>().velocity = new Vector2 (moveX * movementSpeed, gameObject.GetComponent<Rigidbody2D>().velocity.y);

}

void flipPlayer()
    {
    facingRight = !facingRight;

    Vector2 localScale = gameObject.transform.localScale;
    localScale.x *= -1;
    transform.localScale = localScale;
    }

void Jump()
{
    GetComponent<Rigidbody2D>().AddForce(Vector2.up * jumpForce);
}
```

Figure 19 - Player Movement initial draft

This code is relatively straightforward and just takes the horizontal movement from the horizontal axis and multiplies it by the movement speed defined to give the new horizontal velocity. The jumping simply adds an upwards force to the player. Note that this is not the final code, rather an unfinished version. Later versions are documented additionally.

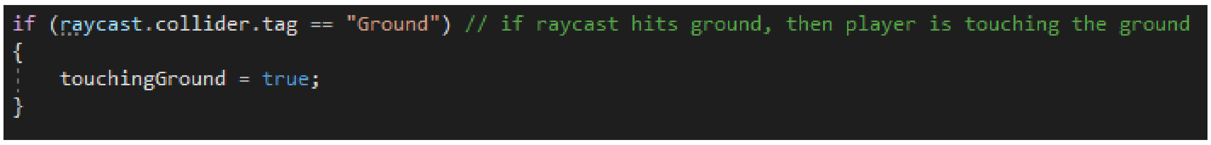
The flipPlayer function when I first implemented it, inversed the player's x local scale to effectively flip the whole object 180 degrees so it faces the other way. I later changed this to use the SpriteRenderer component's flipX function to just flip the sprite, rather than the game object.

At this stage, the player can keep jumping whilst they are in the air, therefore there was a requirement to make a check for if they were touching the ground. Initially, I did this by tagging the ground tilemap with the tag "Ground" and checking on collisions if the tag was ground and if so, setting a variable "touchingGround", but I later decided to switch to using a raycast pointing down. This allowed me to specify distances and would be useful later for detecting when the player jumps on an enemy.



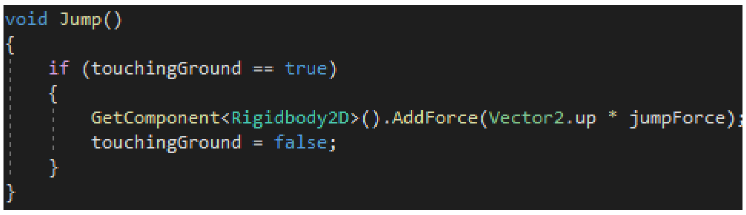
```
RaycastHit2D raycast = Physics2D.Raycast(transform.position, Vector2.down); // sends raycast down from the player
```

Figure 20 - Creating Raycast



```
if (raycast.collider.tag == "Ground") // if raycast hits ground, then player is touching the ground
{
    touchingGround = true;
}
```

Figure 21 - Checking Raycast for ground

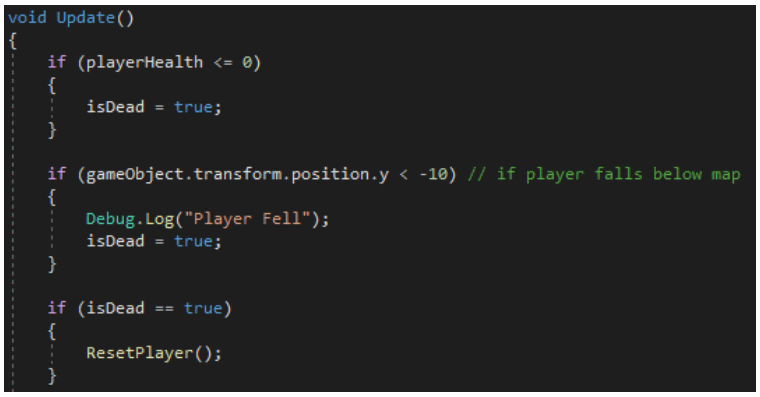


```
void Jump()
{
    if (touchingGround == true)
    {
        GetComponent<Rigidbody2D>().AddForce(Vector2.up * jumpForce);
        touchingGround = false;
    }
}
```

Figure 22 - Jump Touching Ground

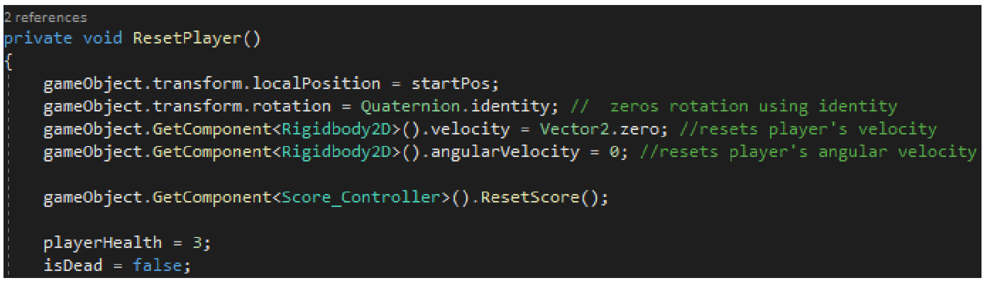
The result was that the player cannot jump unless they are touching the ground.

The next step was handling the player's health and respawning them if they die. To do this I created another script called "Player_Health", again attached to the player object. This would store the player's health, handle damage, and the code to handle the player falling off the map.



```
void Update()
{
    if (playerHealth <= 0)
    {
        isDead = true;
    }

    if (gameObject.transform.position.y < -10) // if player falls below map
    {
        Debug.Log("Player Fell");
        isDead = true;
    }

    if (isDead == true)
    {
        ResetPlayer();
    }
```

Figure 23 - Player Health



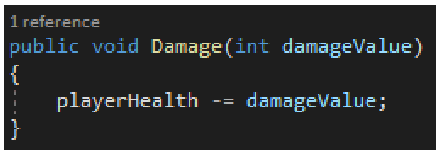
```
2 references
private void ResetPlayer()
{
    gameObject.transform.localPosition = startPos;
    gameObject.transform.rotation = Quaternion.identity; //  zeros rotation using identity
    gameObject.GetComponent<Rigidbody2D>().velocity = Vector2.zero; //resets player's velocity
    gameObject.GetComponent<Rigidbody2D>().angularVelocity = 0; //resets player's angular velocity

    gameObject.GetComponent<Score_Controller>().ResetScore();

    playerHealth = 3;
    isDead = false;
```

Figure 24 - Reset Player

This script simply checks once per frame if the player's health reaches zero, and if so sets isDead to true. If isDead is true, the reset player function is called, which resets the player's position, rotation, velocity, and health. The playerHealth variable starts at 3.

It also checks if the player has gone below -10 on the Y direction, and if so, sets isDead to true. This could also have been done by creating a dedicated collider below the ground to trigger the death if they collide with it. This could be useful if the game needed to go very far down in a level, as otherwise, the player could fall for a long time before dying. This was not necessary for my game, as the levels never go below -10, but nevertheless, it is something to consider if the levels were to expand to go down further.

I also added a public function “Damage”, which I will use later when creating the enemy script to send damage to the player on contact. This just reduces the player’s health by whatever value is passed to it.



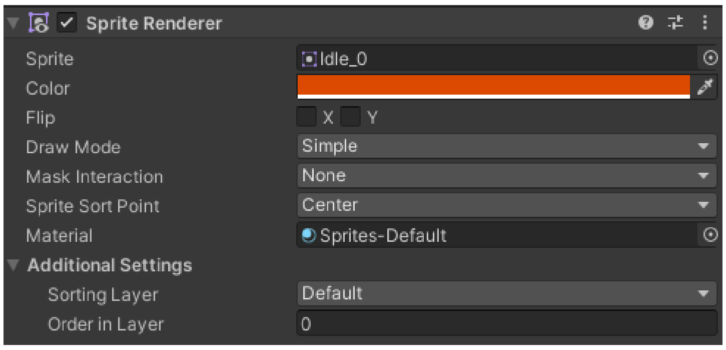
```
1 reference
public void Damage(int damageValue)
{
    playerHealth -= damageValue;
}
```

Figure 25 - Damage function

Now the player will die if they fall below -10 and be respawned at the start position. The reason I reset the player’s location rather than just destroying the player and creating a new one, is that I knew that the 2D camera tracking system would have to be reset later if I did this. This is because the camera will look for a player at the start and changing the instance of the player would mean the original instance was no longer there.

Enemy

To create the enemy, I used the same asset as with the player to create a sprite but coloured it red by changing the sprite colour parameter to make it look distinct from the player sprite. As with the player, I added a rigidbody and collider for physics and collisions and constrained the rotation around the Z-axis.



Sprite Renderer

Sprite	Idle_0
Color	🟧
Flip	X Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	Sprites-Default
Additional Settings	
Sorting Layer	Default
Order in Layer	0

Figure 26 - Enemy Colour

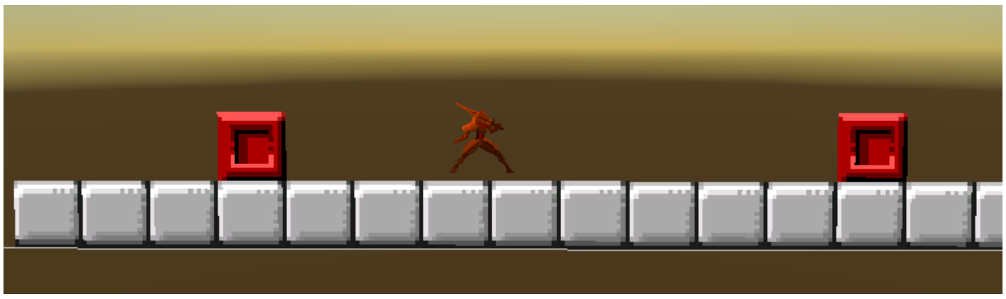
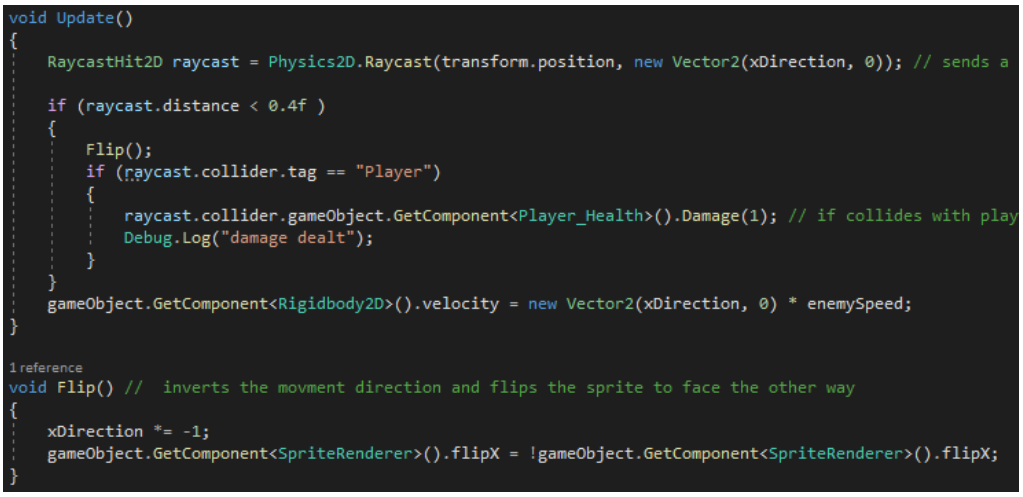


Figure 27 - Enemy Sprite

Then I needed the enemy to move with some degree of autonomy, so I added a script “Enemy_Controller” to the enemy game object.



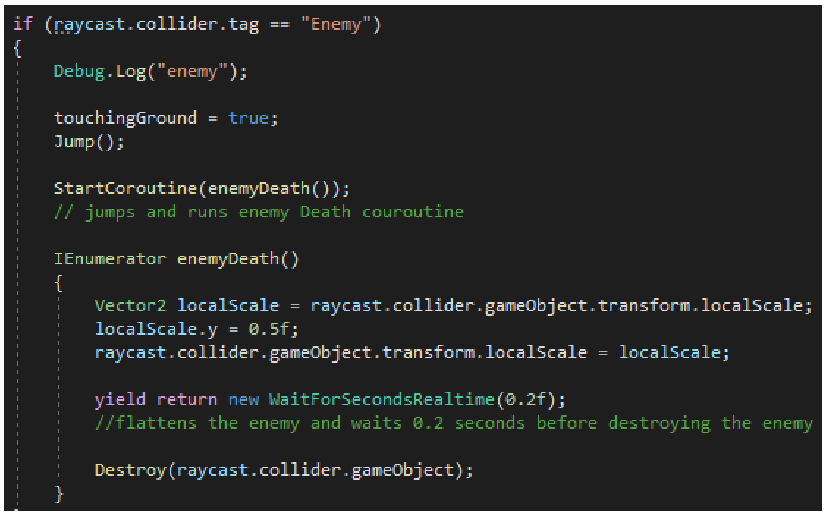
```
void Update()
{
    RaycastHit2D raycast = Physics2D.Raycast(transform.position, new Vector2(xDirection, 0)); // sends a

    if (raycast.distance < 0.4f )
    {
        Flip();
        if (raycast.collider.tag == "Player")
        {
            raycast.collider.gameObject.GetComponent<Player_Health>().Damage(1); // if collides with play
            Debug.Log("damage dealt");
        }
    }
    gameObject.GetComponent<Rigidbody2D>().velocity = new Vector2(xDirection, 0) * enemySpeed;
}

1 reference
void Flip() //  inverts the movment direction and flips the sprite to face the other way
{
    xDirection *= -1;
    gameObject.GetComponent<SpriteRenderer>().flipX = !gameObject.GetComponent<SpriteRenderer>().flipX;
}
```

Figure 28 - Enemy Controller

In this script, once a frame it sends out a raycast in the direction the enemy is facing, if it hits anything closer than 0.4 then it flips the sprite and flips the direction it is travelling. If the collider it hits is a player, then it gets the instance of the Player_ Health class that is attached to the collider's game object and calls the Damage function from Player_Health for 1 damage.

This means that whenever the enemy gets to a collider, it will flip and start travelling the other way. This also happens when it hits a player, and damage is dealt.

At this point, I wanted to have the ability to kill the enemies by jumping on them. I did this by adding a check in the Player_Movement's raycast:



```
if (raycast.collider.tag == "Enemy")
{
    Debug.Log("enemy");

    touchingGround = true;
    Jump();

    StartCoroutine(enemyDeath());
    // jumps and runs enemy Death couroutine

    IEnumerator enemyDeath()
    {
        Vector2 localScale = raycast.collider.gameObject.transform.localScale;
        localScale.y = 0.5f;
        raycast.collider.gameObject.transform.localScale = localScale;

        yield return new WaitForSecondsRealtime(0.2f);
        //flattens the enemy and waits 0.2 seconds before destroying the enemy

        Destroy(raycast.collider.gameObject);
    }
```

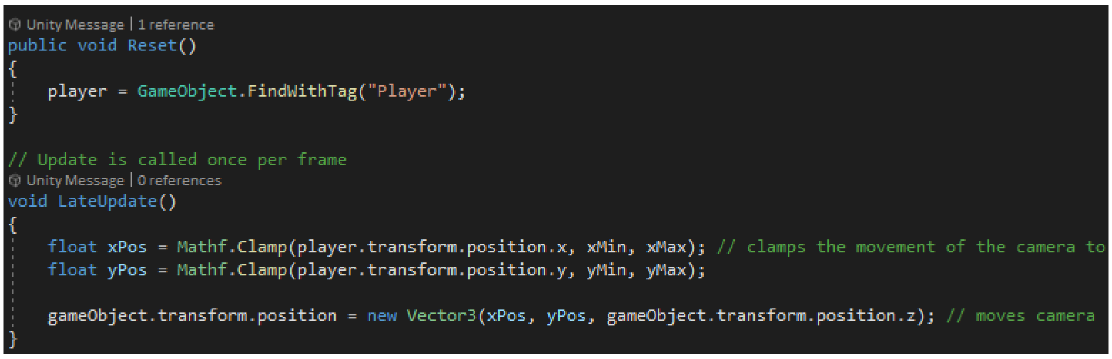
Figure 29 - Enemy Death

This checks the collider for the tag "Enemy" and if it is, it calls the jump function and starts a coroutine enemyDeath. This coroutine flattens the enemy object to half its height, to appear squashed and waits 0.2 seconds before destroying the enemy.

Camera Controller

The next part of the 2D game I created was a script to make the camera follow the player around the game. Originally, I had decided to have the camera as part of the level prefabs and respawn it every level to avoid having to connect the camera controller with the player every level as the player changed. As I was intending to create a Main Menu and this would need a camera to attach the canvas to, this would make this more difficult as I would need another camera. Additionally, changing the camera would unlink the Camera Feed Texture which is needed to display on the arcade screen. For these reasons I decided to uncouple the camera with the level prefabs, so it is now a permanent part of the 3D game, independent of levels.

To move the camera around with the player, I attached a script “Camera_Controller” to the Feed Camera.



```
Unity Message | 1 reference
public void Reset()
{
    player = GameObject.FindWithTag("Player");
}

// Update is called once per frame
Unity Message | 0 references
void LateUpdate()
{
    float xPos = Mathf.Clamp(player.transform.position.x, xMin, xMax); // clamps the movement of the camera to
    float yPos = Mathf.Clamp(player.transform.position.y, yMin, yMax);

    gameObject.transform.position = new Vector3(xPos, yPos, gameObject.transform.position.z); // moves camera
}
```

Figure 30 - Camera Controller

This script sets the X and Y position of the camera to be equal to the player’s X and Y position, provided they are within the min and max values set in the corresponding variables. This works as the camera is parallel to the 2D game which is placed along the X-axis. Additionally, the camera is an Orthographic camera, therefore has no perspective. This means that it doesn’t matter where the camera is placed on the Z-axis, it will always show the same sized output. In this case, the script just maintains whatever the Feed Camera’s X position was previously.

I also added a function “Reset” to find the player game object using Unity’s “FindWithTag” function. Note that this function is inefficient, so should only be run when needed. In this case, the Reset function will only be run once when the levels are started, so this is not a problem, but it would be a poor use of resources to run this once a frame for example.

The camera can now follow the player around the level.

Main Menu

The next component is the main menu, which will allow the user to select levels and view their high scores. To do this I needed the 2D game to be part of a prefab, so the selected level can be instantiated by the main menu.

After putting the 2D game into a prefab, I created a new game object called “Level Menu” which will hold the UI Canvas object. Placing the Canvas within another game object meant I could disable the canvas to hide the menu, without disabling the Level Menu object. This allows me to find the Level Menu object to re-enable the canvas when the level ends and the menu needs to be visible again.

I designed the menu using a background, title, 2 buttons and 2 text fields:

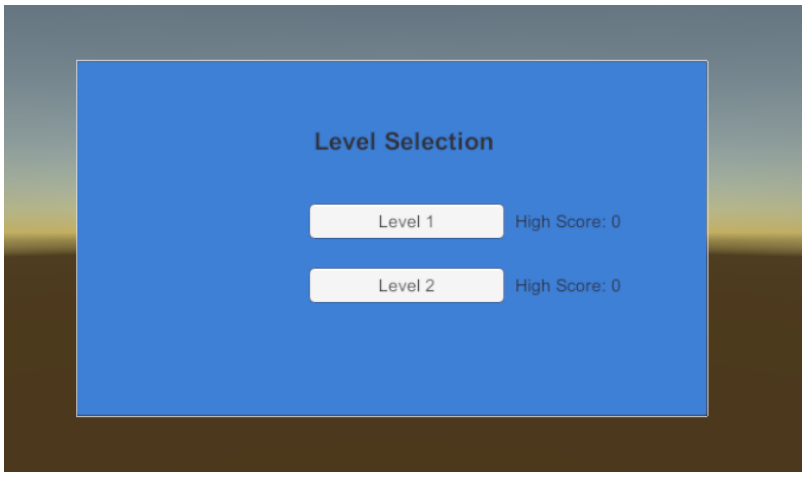
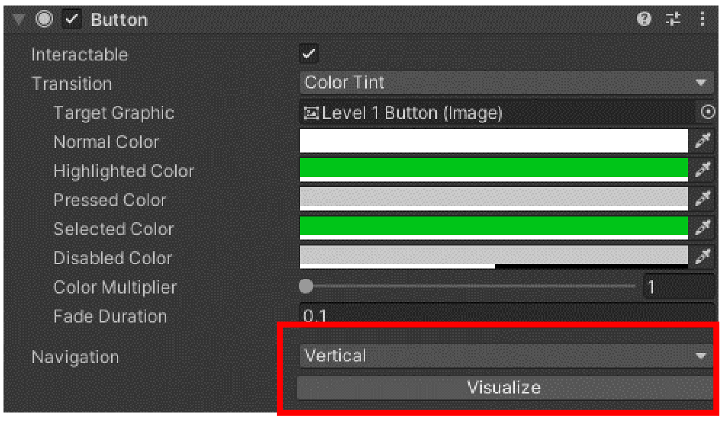


Figure 31 - Main Menu

By setting the Navigation setting on the Buttons to "Vertical", it allows me to use the built-in event system to navigate the menu using the default vertical buttons on either keyboard (for testing) or the joystick on the VR controller.



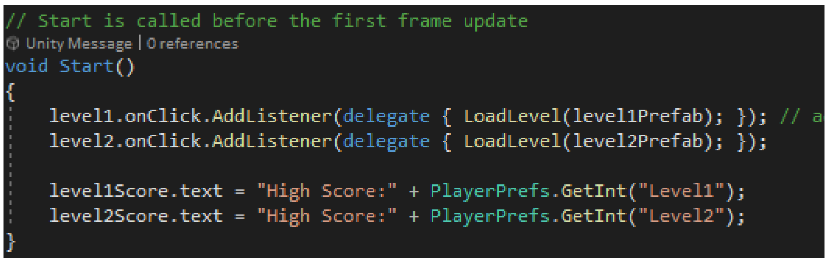
Button

Interactable	✓
Transition	Color Tint
Target Graphic	Level 1 Button (Image)
Normal Color	⬜
Highlighted Color	🟩
Pressed Color	▨
Selected Color	🟩
Disabled Color	▨
Color Multiplier	1
Fade Duration	0.1
Navigation	Vertical

Figure 32 - Button Navigation

After this, I created a new script called "Game_Controller" to handle the main menu and loading of levels.

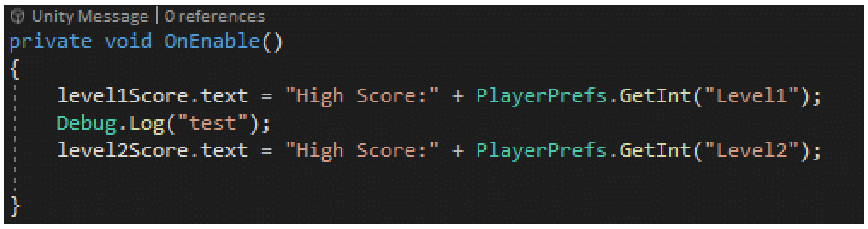
First, the script adds listeners to the onClick events for both of the level buttons (this could be expanded to however many levels are used). These listeners will run the lambda expression which calls the LoadLevel function with the corresponding prefabs for the levels which are defined in the Unity editor. It also sets the high score text to a PlayerPrefs variable. PlayerPrefs are designed to store player preferences such as settings, so isn't a very secure way of storing the high scores, as it is reasonably easy to modify these values. However, this is not a concern in this project, as it is more a proof of concept. I set these high scores later when I set up the scoring and end game



```
// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
    level1.onClick.AddListener(delegate { LoadLevel(level1Prefab); }); // a
    level2.onClick.AddListener(delegate { LoadLevel(level2Prefab); });

    level1Score.text = "High Score:" + PlayerPrefs.GetInt("Level1");
    level2Score.text = "High Score:" + PlayerPrefs.GetInt("Level2");
}
```

Figure 33 - Game Controller Start

LoadLevel instantiates the level prefab passed to it in the position 40, 0, 0. It then finds the Canvas component in its children objects and sets the world camera of that canvas to the Feed Camera. This is a canvas used later on for the in-game UI. Next, the script calls the Reset function in the Feed Camera's Camera_Controller to find the player object for the camera to follow. Finally, it deactivates the canvas.



```
Unity Message | 0 references
private void OnEnable()
{
    level1Score.text = "High Score:" + PlayerPrefs.GetInt("Level1");
    Debug.Log("test");
    level2Score.text = "High Score:" + PlayerPrefs.GetInt("Level2");

}
```

Figure 34 - Game Controller OnEnable

The OnEnable method is called when the canvas is re-enabled and resets the high score text in case the high score has changed since the last game.

In-game UI

Before adding in a timer, scoring and end goal, I needed to add a UI for the game to display the player's score, health, and remaining time. For this, I added a canvas to the 2D level prefab, with text fields for the score, health and time placed at the top of the screen. I also added another text field that I will use for displaying the score at the end of the level. Here the End Score Text is not visible as it is disabled.

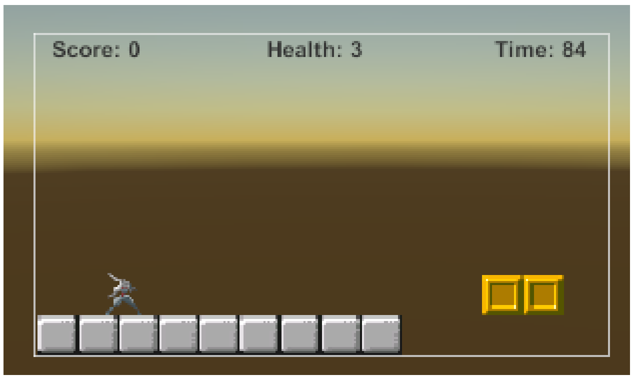


Figure 35 - In-game UI

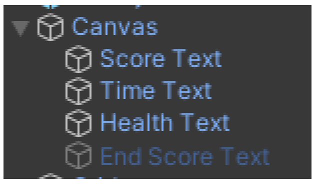
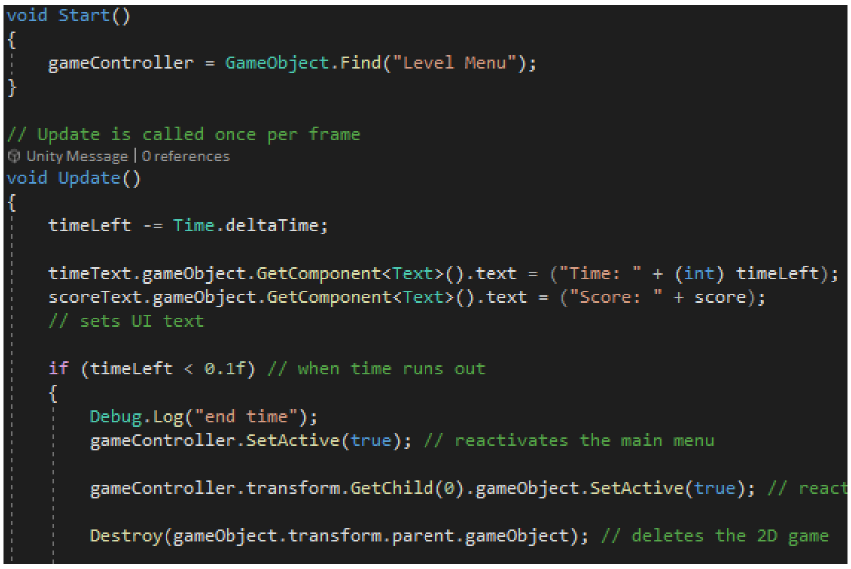


Figure 36 - UI Canvas

Then, to manage the score and timing of the game, I added a new script to the player called "Score_Controller".



```
void Start()
{
    gameController = GameObject.Find("Level Menu");
}

// Update is called once per frame
Unity Message | 0 references
void Update()
{
    timeLeft -= Time.deltaTime;

    timeText.gameObject.GetComponent<Text>().text = ("Time: " + (int) timeLeft);
    scoreText.gameObject.GetComponent<Text>().text = ("Score: " + score);
    // sets UI text

    if (timeLeft < 0.1f) // when time runs out
    {
        Debug.Log("end time");
        gameController.SetActive(true); // reactivates the main menu

        gameController.transform.GetChild(0).gameObject.SetActive(true); // react

        Destroy(gameObject.transform.parent.gameObject); // deletes the 2D game
```

Figure 37 - Score Controller Update

This script updates the time left every frame by the time since the last frame (deltaTime). This is so the timer is independent of the game's frame rate and counts in real-time. It also updates the time and score text to reflect any changes. Next, if the time has run out, it reactivates the main menu and the canvas then deletes the 2D game prefab, ready for the next level to be launched by the main menu.

After this, I added a new object to the tilemap in the level called "End Goal" with the tag "End Level". This will serve as the target that the player must reach to end the level. Again, I used the pixel art tiles for this.

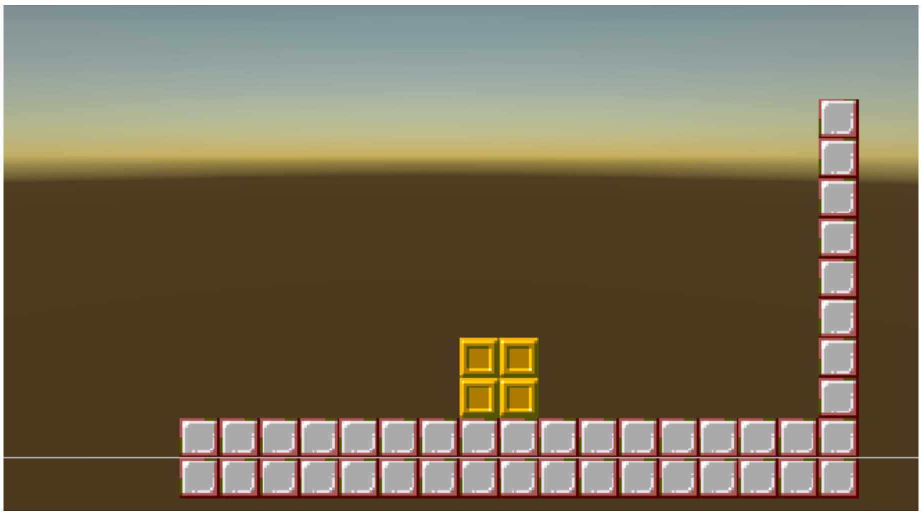
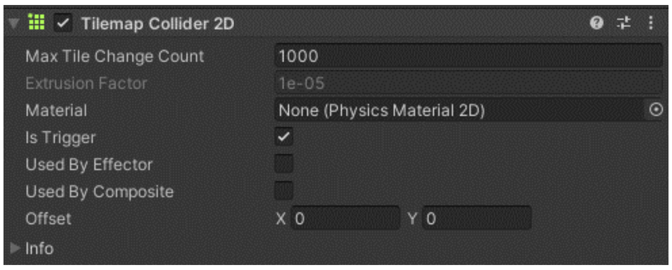


Figure 38 - End Goal

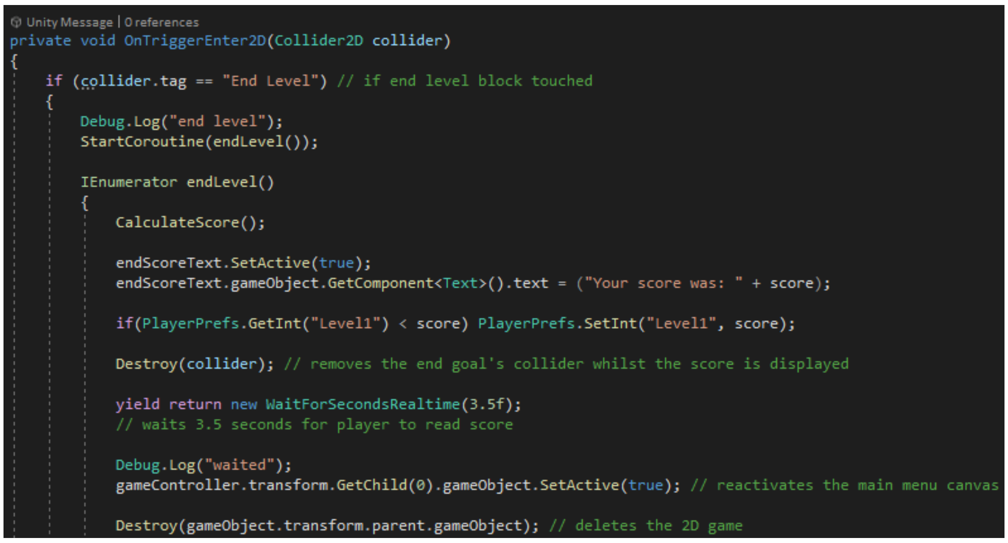
To detect when the end goal is reached, I set the end goal's collider to be a trigger. This will allow me to detect when it collides with the player, without the player bumping into the goal. In other words, the player can walk into the end goal and it will trigger.


Tilemap Collider 2D

Max Tile Change Count	1000
Extrusion Factor	1e-05
Material	None (Physics Material 2D)
Is Trigger	✓
Used By Effector	
Used By Composite	
Offset	X 0 Y 0
Info	

Figure 39 - End Goal Trigger

I then added an OnTriggerEnter2D method to the Score_Controller. This checks if the trigger was the end goal and if so, starts the coroutine endLevel. EndLevel calculates the score and displays it on the End Score Text. It then saves the score as a high score if it is higher than the saved high score. Next, it removes the end goal's collider, in case the player runs out of the goal and back in again. This stops the player from being able to gain points repeatedly at the end of the level. Then it waits for 3.5 seconds, re-enables the main menu canvas, and destroys the 2D game ready for the next level.



```
Unity Message | 0 references
private void OnTriggerEnter2D(Collider2D collider)
{
    if (collider.tag == "End Level") // if end level block touched
    {
        Debug.Log("end level");
        StartCoroutine(endLevel());

        IEnumerator endLevel()
        {
            CalculateScore();

            endScoreText.SetActive(true);
            endScoreText.gameObject.GetComponent<Text>().text = ("Your score was: " + score);

            if(PlayerPrefs.GetInt("Level1") < score) PlayerPrefs.SetInt("Level1", score);

            Destroy(collider); // removes the end goal's collider whilst the score is displayed

            yield return new WaitForSecondsRealtime(3.5f);
            // waits 3.5 seconds for player to read score

            Debug.Log("waited");
            gameController.transform.GetChild(0).gameObject.SetActive(true); // reactivates the main menu canvas

            Destroy(gameObject.transform.parent.gameObject); // deletes the 2D game
```

Figure 40 - End Goal OnTriggerEnter

At this stage, the 2D game functioned as I intended. The movement and jumping are not quite as smooth as in a finished commercial game, but it was still very playable.

VR Interaction

Grabbable Block

The final step in the implementation was to develop the interaction between the 2D game and the 3D environment using VR.

To do this, I first added a block to the 3D scene which would act as a bridge in the game. I added a texture to this from the Unity URP sample materials and added a rigidbody and collider. I decided to make the block' rigidbody kinematic. This means that forces such as gravity and collisions will not affect the block at all, and the only thing that can change the position of the block are scripts. This may not have been necessary, but it simplified the development later and the block not having gravity doesn't affect the functionality in any way. It also makes it a bit easier for a novice user to use, as if they 'drop' the block, then it won't fall to the floor.

I also set the block's collider to act as a trigger. This allows the grab component used next to tell if there is a collision with an interactable object.

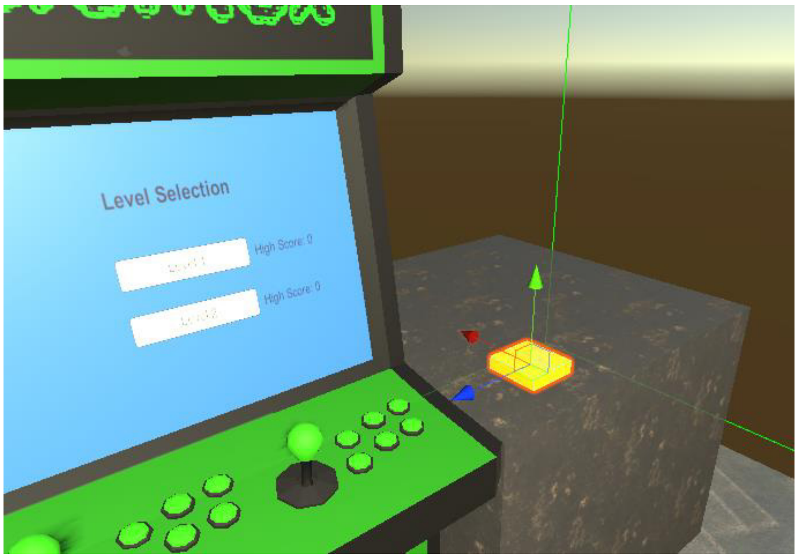
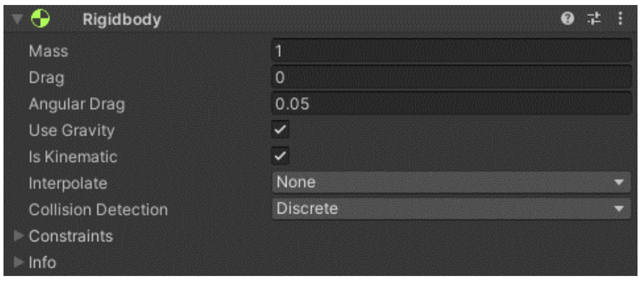


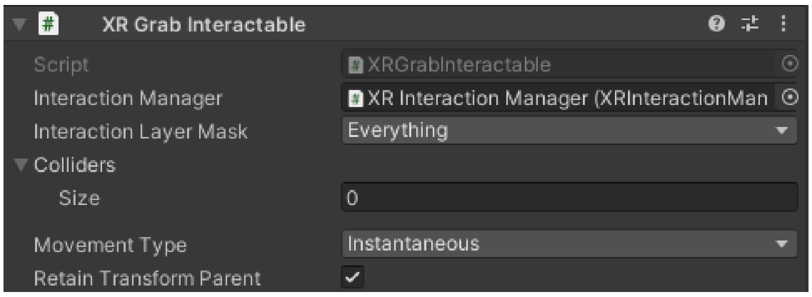
Figure 41 - Bridge Block


Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	✓
Interpolate	None
Collision Detection	Discrete
Constraints	
Info	

Figure 42 - Bridge Block Kinematic

Next, I needed to make the block be able to be 'grabbed' by the player using the VR controllers. To do this I added the component XR Grab Interactable to the block. This will allow the block to be grabbed by an XR Interactor. I later found out that I should set the movement type in the Grab Interactable to "Instantaneous" from the default "Kinematic", as otherwise, this caused problems when moving the block within the screen later.


XR Grab Interactable

Script	XRGrabInteractable
Interaction Manager	XR Interaction Manager (XRInteractionMan
Interaction Layer Mask	Everything
Colliders	
Size	0
Movement Type	Instantaneous
Retain Transform Parent	✓

Figure 43 - XR Grab Interactable

Then I needed to add an XR Direct Interactor to both VR controller objects. This will detect any trigger colliders that enter the spheres I added earlier to represent the controllers/hands. If these trigger objects contain the XR Grab Interactable component, it will allow the player to grab them using the grip button on the controllers.

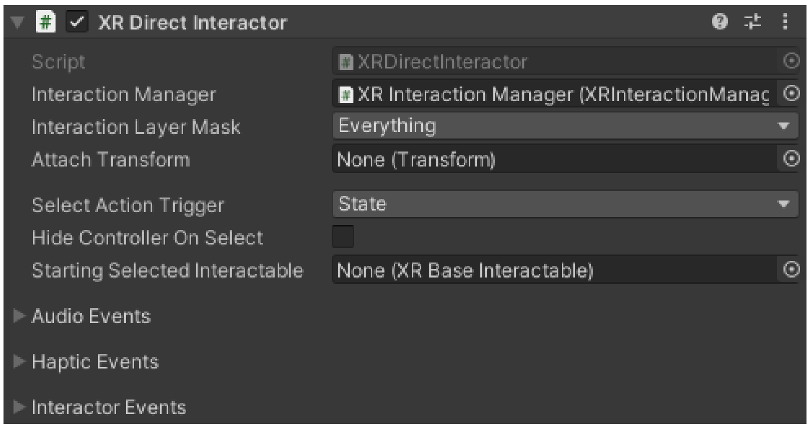

XR Direct Interactor
Script XRDirectInteractor
Interaction Manager XR Interaction Manager (XRInteractionManag
Interaction Layer Mask Everything
Attach Transform None (Transform)
Select Action Trigger State
Hide Controller On Select
Starting Selected Interactable None (XR Base Interactable)
Audio Events
Haptic Events
Interactor Events

Figure 44 - XR Direct Interactor

With this done, the player can grab the block with either hand using the grip button. The block stays where the player left it once they let go. This is also another benefit to making the block kinematic, as when the player lets go, it does not then collide with their hands, which could push the block out of reach.

VR Interaction Code

The final aim in development was to make the block appear in the 2D game when you push it into the screen in the 3D space. Finding a solution to this was not trivial and so I needed to find a simple way to translate the virtual screen space in 3D, into the '2D' space of the game. Note that the 2D game is actually in the 3D space, just along the X-axis starting at 40, 0, 0.

The solution I came up with was to work out 2 ratios for the difference between the size of the virtual screen, and the camera size, then use this ratio to translate the location based on the position of the camera at the time.

To do this, I created a new script on the screen object called "VR_Interactor" which will contain all the code for the interaction:

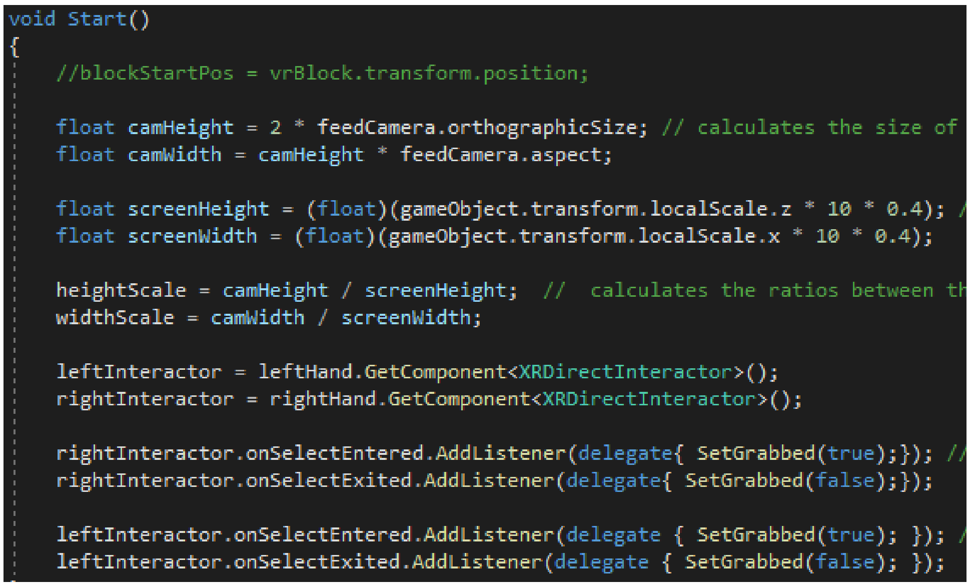

```
void Start()
{
    //blockStartPos = vrBlock.transform.position;

    float camHeight = 2 * feedCamera.orthographicSize; // calculates the size of
    float camWidth = camHeight * feedCamera.aspect;

    float screenHeight = (float)(gameObject.transform.localScale.z * 10 * 0.4); /
    float screenWidth = (float)(gameObject.transform.localScale.x * 10 * 0.4);

    heightScale = camHeight / screenHeight;  //  calculates the ratios between th
    widthScale = camWidth / screenWidth;

    leftInteractor = leftHand.GetComponent<XRDirectInteractor>();
    rightInteractor = rightHand.GetComponent<XRDirectInteractor>();

    rightInteractor.onSelectEntered.AddListener(delegate{ SetGrabbed(true);}); //
    rightInteractor.onSelectExited.AddListener(delegate{ SetGrabbed(false);});

    leftInteractor.onSelectEntered.AddListener(delegate { SetGrabbed(true); }); /
    leftInteractor.onSelectExited.AddListener(delegate { SetGrabbed(false); });
```

Figure 45 - VR_Interactor Start

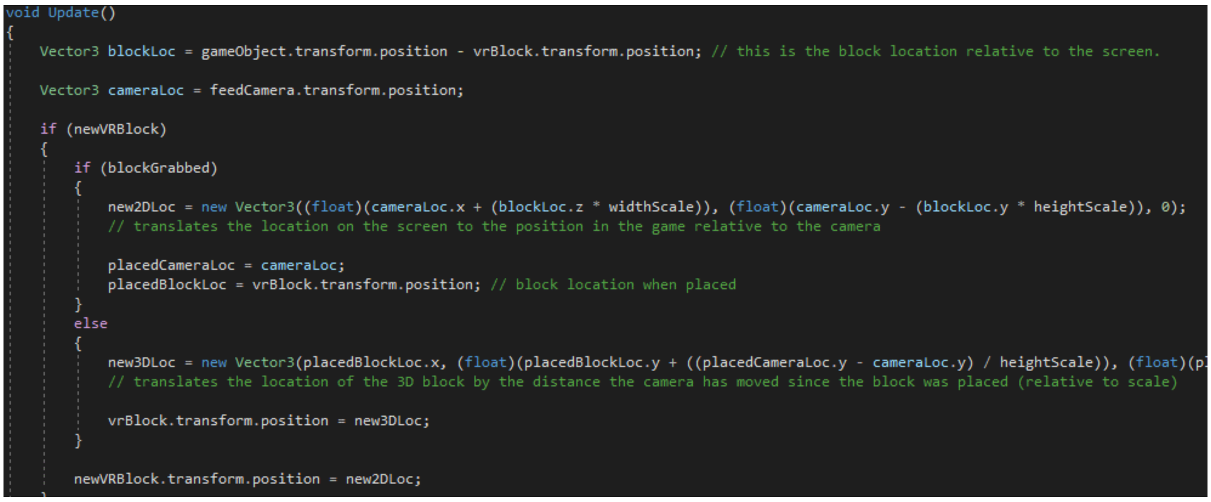
First, when the script is first loaded, it calculates the height and width of the camera. This is calculated using the camera's size and aspect ratio. I took these values from the camera rather than setting them manually, as it gives the option to change the camera size later without having to change this code at all.

Next, it calculates the height and width of the virtual screen by multiplying the scale of the plane that makes up the screen by 10 and 0.4. 10 because the screen is made of a plane, which is always 10x10 units at a scale of 1, and 0.4 as this is the scale of the arcade machine the screen is the child object of, so the plane has also been scaled by 0.4.

Then it calculates the width and height ratio between the camera and the screen.

After this, it adds listeners to the onSelectEntered and onSelectExited functions of the XR Direct Interactor components in the left and right controllers. These are lambda functions which call SetGrabbed to set a variable "blockGrabbed" which stores whether the block is currently being grabbed. This will be used later to check whether the block is still being moved by the player or has been placed in a position and let go of.

Next, I added the code which will translate the position of the block into the 2D space:

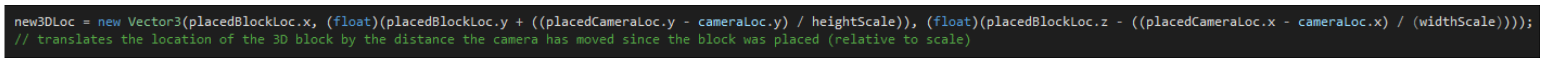


```
void Update()
{
    Vector3 blockLoc = gameObject.transform.position - vrBlock.transform.position; // this is the block location relative to the screen.

    Vector3 cameraLoc = feedCamera.transform.position;

    if (newVRBlock)
    {
        if (blockGrabbed)
        {
            new2DLoc = new Vector3((float)(cameraLoc.x + (blockLoc.z * widthScale)), (float)(cameraLoc.y - (blockLoc.y * heightScale)), 0);
            // translates the location on the screen to the position in the game relative to the camera

            placedCameraLoc = cameraLoc;
            placedBlockLoc = vrBlock.transform.position; // block location when placed
        }
        else
        {
            new3DLoc = new Vector3(placedBlockLoc.x, (float)(placedBlockLoc.y + ((placedCameraLoc.y - cameraLoc.y) / heightScale)), (float)(p
            // translates the location of the 3D block by the distance the camera has moved since the block was placed (relative to scale)

            vrBlock.transform.position = new3DLoc;
        }

        newVRBlock.transform.position = new2DLoc;
```

Figure 46 - VR_Interactor Update

This code is run every frame, and first calculates the position of the 3D block relative to the screen. It then stores the camera's location. If newVRBlock exists, which will be a reference to the new block in the '2D' space of the game, then it checks if the block is being grabbed. If it is still being grabbed, then it defines a Vector3 (3d position) "new2DLoc" as the camera's current location plus the block's relative location multiplied by the ratio calculated at the start. Note that the X value is calculated from the block's Z location on the screen, as the screen is at 90 degrees to the 2D game space, therefore the Z-axis corresponds to the X-axis in the camera plane. It then stores the world (not relative) position of the camera and the block when they were placed.

If the block has been placed (newVRBlock exists) but the block isn't being grabbed anymore, then it defines a new position "new3DLoc":



```
new3DLoc = new Vector3(placedBlockLoc.x, (float)(placedBlockLoc.y + ((placedCameraLoc.y - cameraLoc.y) / heightScale)), (float)(placedBlockLoc.z - ((placedCameraLoc.x - cameraLoc.x) / (widthScale))));
// translates the location of the 3D block by the distance the camera has moved since the block was placed (relative to scale)
```

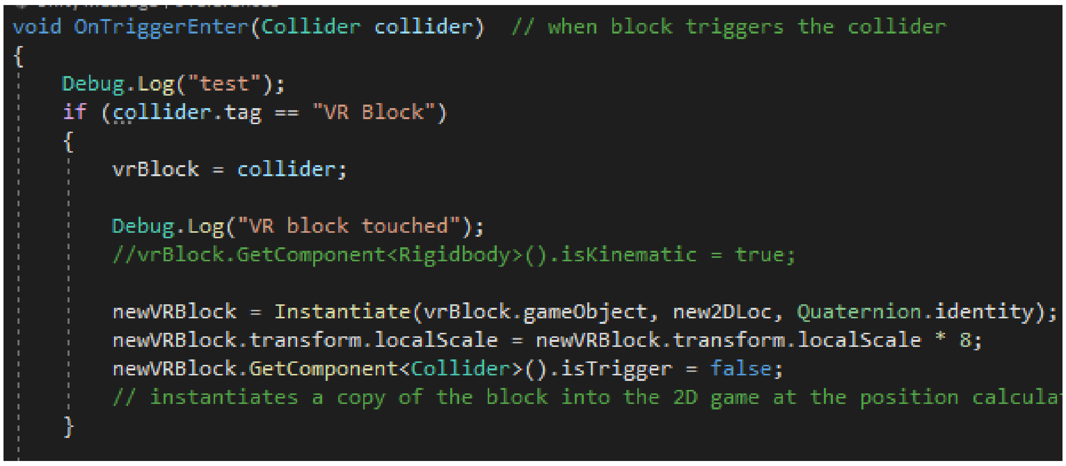
Figure 47 - new3DLoc

This calculates a new location for the 3D Block so that if the camera moves, the location of the block in the screen moves to reflect this. This is calculated by first taking the difference between the camera location when it was placed and the current camera position and dividing it by the ratios calculated earlier. It then adds this to the placed block location.

Finally, it sets the location of the 2D Block in the game and the 3D block in the screen to their new locations.

When I coded this initially, I had a bug where once the camera moved, the block would shoot off, but in the right direction. I eventually realised that I was adding to the actual location of the 3D block once a frame, rather than the placed location. This meant that the position kept getting added to every frame and it kept moving. The solution to this was just to store the location the block was placed at with the placedBlockLoc variable.

The next function I developed was the detection of the block being pushed into the screen. Initially, I tried doing this using Unity's OnCollisionEnter and Exit functions, but I ran into a bug where when you pushed the block into the screen, it would keep 'entering' and 'exiting' every frame, rather than just once when the block enters and once when it leaves. Instead, as described earlier, I set the block's collider to be a trigger and used Unity's OnTriggerEnter and OnTriggerExit to handle this, as this worked with no problems.

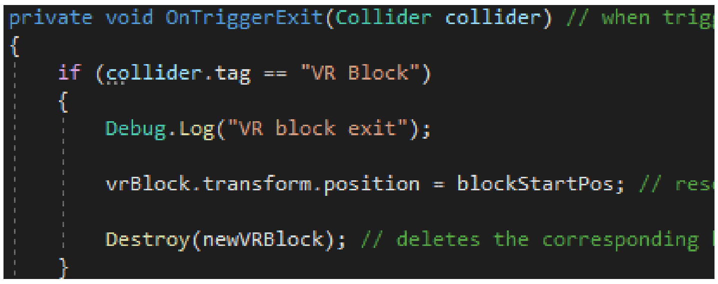


```
void OnTriggerEnter(Collider collider)  // when block triggers the collider
{
    Debug.Log("test");
    if (collider.tag == "VR Block")
    {
        vrBlock = collider;

        Debug.Log("VR block touched");
        //vrBlock.GetComponent<Rigidbody>().isKinematic = true;

        newVRBlock = Instantiate(vrBlock.gameObject, new2DLoc, Quaternion.identity);
        newVRBlock.transform.localScale = newVRBlock.transform.localScale * 8;
        newVRBlock.GetComponent<Collider>().isTrigger = false;
        // instantiates a copy of the block into the 2D game at the position calcula
    }
```

Figure 48 - VR_Controller Trigger Enter

This code checks if the trigger was a VR Block, and if it is, sets the variable vrBlock to the collider that triggered. It then instantiates a copy of the block at the position defined in new2DLoc in the code in figure 47. Then it scales up the new block by 8 to better match the scale and stops its collider from being a trigger.

Finally, I added the code to handle when the block is pulled out of the screen and delete the block in the 2D game:



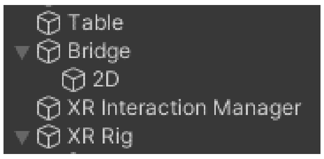
```
private void OnTriggerExit(Collider collider) // when trig
{
    if (collider.tag == "VR Block")
    {
        Debug.Log("VR block exit");

        vrBlock.transform.position = blockStartPos; // res

        Destroy(newVRBlock); // deletes the corresponding
    }
```

Figure 49 - VR_Controller Trigger Exit

This resets the 3D block back to its original position on the table and destroys the new block in the 2D game to reflect the block being removed from the screen.

This mostly functioned as I expected, in that the block created in the 2D game followed the placement of the 3D block. However, the player in the 2D game could not collide with the block, so couldn't use the block to jump on.

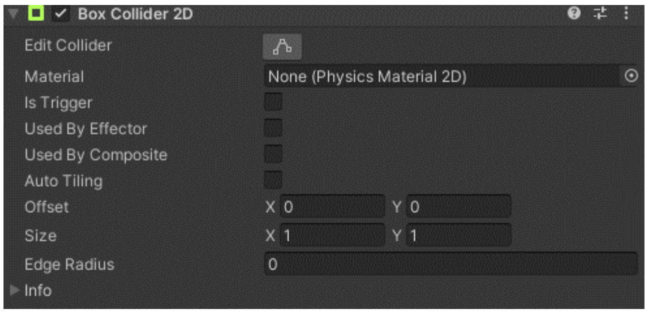
I determined that this is because although 2D and 3D objects in unity share the 3D space, they use different physics engines, physX for 3D and Box2D for 2D. To get around this, I added a 2D box collider to the block object. This doesn't affect the block in 3D but allows 2D objects to collide with it. Normally unity does not allow 2D and 3D components on the same object, however, I solved this by creating a child of the block called "2D" just for the 2D collider:



```
Table
▼ Bridge
    2D
XR Interaction Manager
▼ XR Rig
```

Figure 50 - Bridge 2D Child

I could then add a 2D collider to this child object:



Box Collider 2D	
Edit Collider	
Material	None (Physics Material 2D)
Is Trigger	☐
Used By Effector	☐
Used By Composite	☐
Auto Tiling	☐
Offset	X 0 Y 0
Size	X 1 Y 1
Edge Radius	0
Info	

Figure 51 - Block 2D Collider

Now the 2D player can jump on the block when it is placed into the screen, and the block is removed from the 2D game when the 3D block is pulled from the screen:

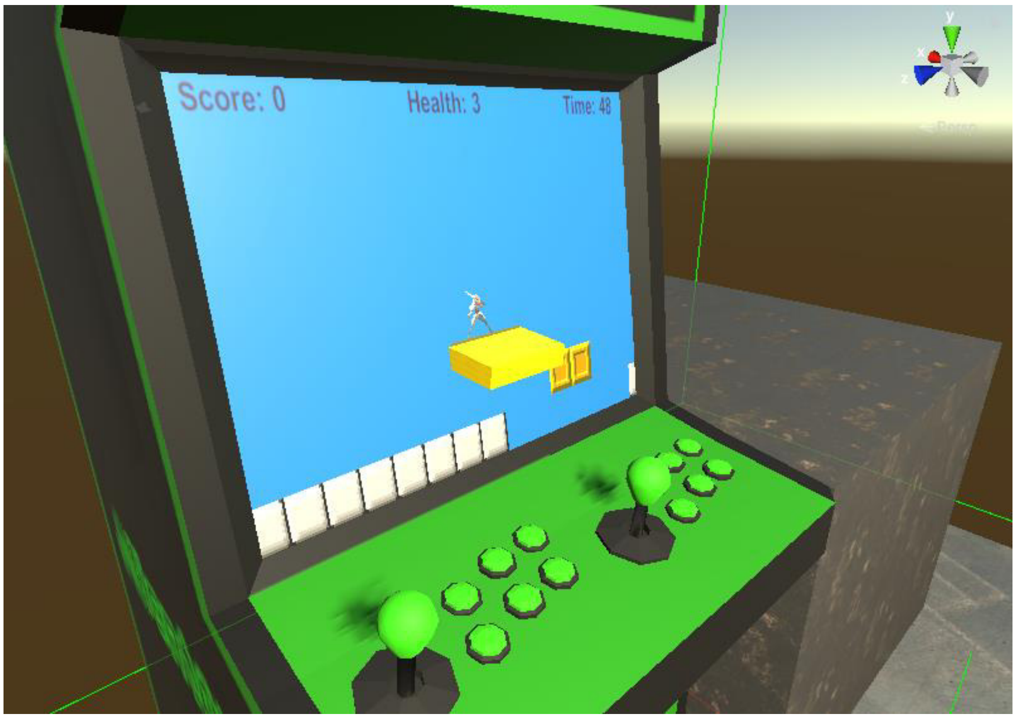


Figure 52 - Interactive Block Working on Virtual Screen

Here is the block that is created in the ‘2D’ game:

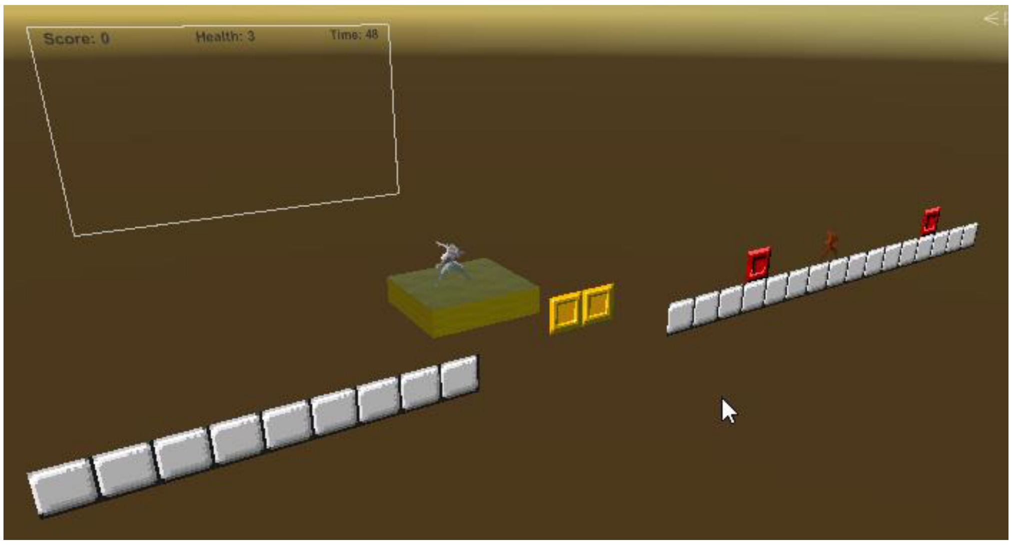


Figure 53 - Interactive Block Working in 2D Game

The placed 3D block also follows the apparent position of the ‘2D’ block on the screen when the camera is moved:

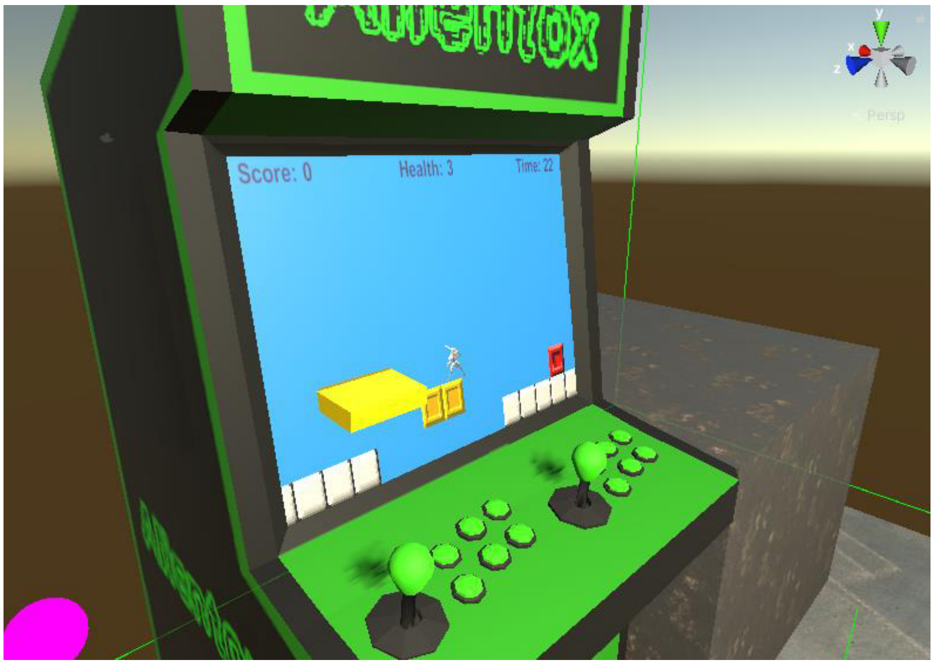


Figure 54 - 3D Block Moves with Camera

Here is an example of the block being used in the game:

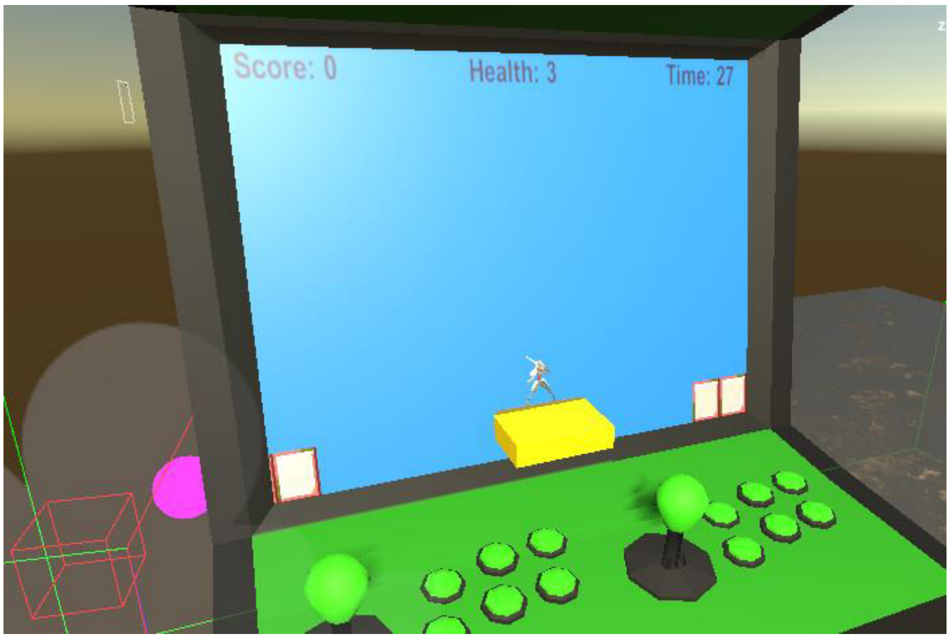
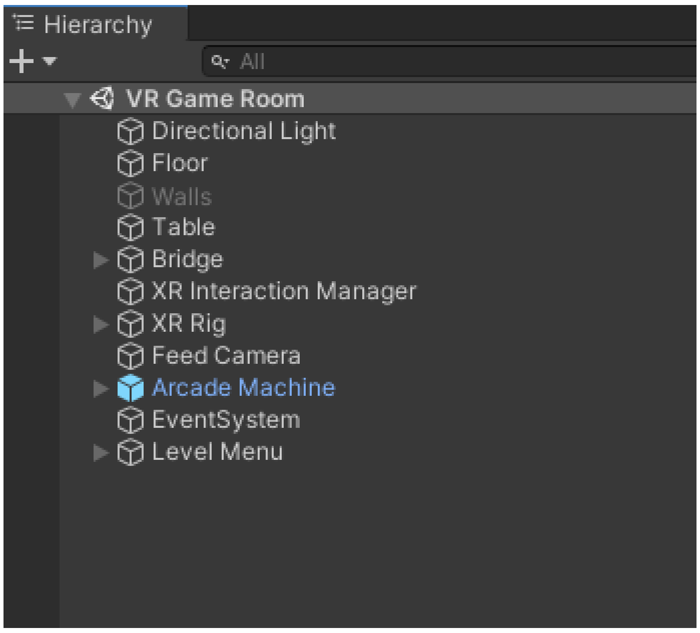


Figure 55 - Interactive Block Example Use

In figure 55 above, the player cannot cross the gap by jumping as it is too far, so the user must use the interactive block to create a platform in the middle to jump to.

Unfortunately, it is not easy to show the full functionality of this through pictures alone. I created the above photos by placing the blocks using the VR headset, then pausing the game through the Unity Editor to take screenshots.

Here Is a screenshot of the Unity project hierarchy at the end of development:


Hierarchy

VR Game Room
- Directional Light
- Floor
- Walls
- Table
- Bridge
- XR Interaction Manager
- XR Rig
- Feed Camera
- Arcade Machine
- EventSystem
- Level Menu

Bug Fixing

During and after development, there were some changes I made to make the game a little easier to control. None of these would change whether my requirements were met but were notable as they made the game more consistent and thus slightly easier to play.

2D Player's Collider

Initially, the 2D player only had a capsule collider to handle collisions with the level and objects. I used a capsule collider so that the collider could be rounded at the corners so that the corner wouldn't catch on the edge of blocks. This successfully worked most of the time, but there were some instances where the player would not be able to jump if they tried while touching something next to them. For example, in figure 56 below the player cannot jump if they hold the 'right' direction at the same time, as there is friction between the right side of the player's collider and the block:

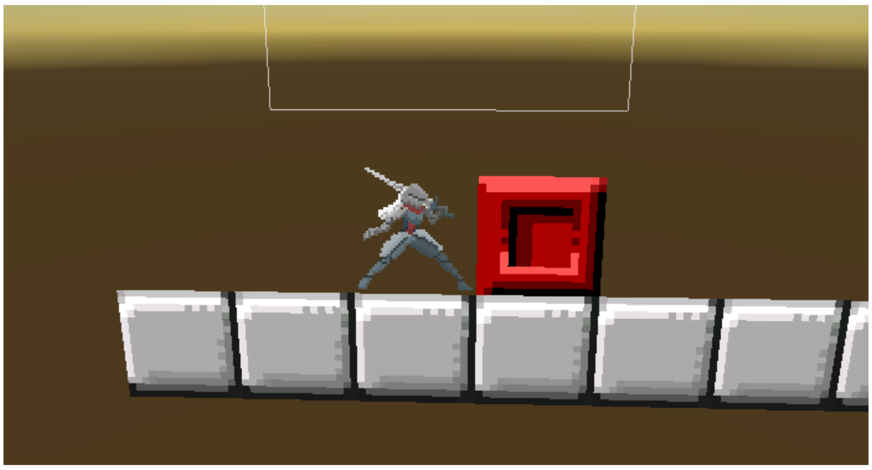


Figure 56 - Player Getting Stuck Bug

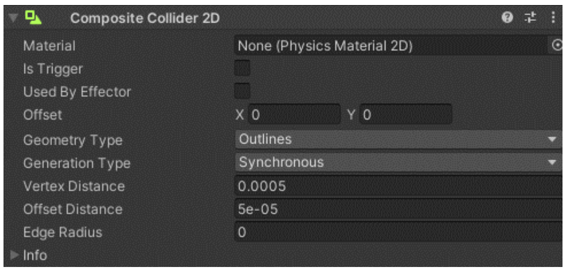
The solution I found was to change the collider to a box collider and add a component called "Platform Effector 2D". This component allows you to alter the behaviour of the collider such as whether there is friction or bouncing on the sides of the collider. This component is specifically designed for 2D platformer colliders.

With this component added, and the side friction and bounce disabled, the player collider now has no friction on the left and right sides and so the player can hold a direction and run at the same time. This also fixed the occasional bug I had where if the player hit the side of a block when in the air from jumping, if they held down the direction of the block, they would stick to the side of the block rather than sliding off it. This was caused by the same problem with the friction on the side of the collider.

2D Level Collider

Earlier when I created the level, I used a Tilemap Collider, which adds colliders for every block, however later in development I noticed that the player would catch slightly on every ground block while running along a flat surface. This was because the collider was made up of multiple smaller ones, and the player's collider would snag on the edge of each collider as it went over it.

The solution I found for this was to add a Composite Collider 2D component to the ground Tilemap. This takes the Tilemap's multiple colliders, and merges any connecting blocks into single polygon colliders, thereby smoothing out the colliders, fixing the issue.


Composite Collider 2D

Material	None (Physics Material 2D)
Is Trigger	
Used By Effector	
Offset	X 0 Y 0
Geometry Type	Outlines
Generation Type	Synchronous
Vertex Distance	0.0005
Offset Distance	5e-05
Edge Radius	0
Info	

Figure 57 - Tilemap Composite Collider

Results and Evaluation

This section will aim to show to what extent the app has met the aims and requirements I set out before starting the project in the Specification section. I will demonstrate this by detailing the results of both my own testing and the results from user testing I carried out. I will then give a critical appraisal of the project as a whole, and to what extent I feel it was successful.

Testing

To test the app myself, I tested the game against my original requirements set out in the specification by repeatedly playing the app and determining if each requirement was met. I feel that these tests will give a good indication of where the app has and has not met my overall aims for the project.

The results of my own testing are as follows:

Main Requirements

These tests are against the main requirements set out in the specification. These requirements are categorised as main requirements because they are integral to meeting the objectives of the project.

Table 1.

Test Number	**Requirement Description**	**Pass/Fail/ Partial**	**Comments**
3D VR Environment			
1.	There must be a simple room to contain the arcade machine and any other background objects.	Pass	There is a basic room with the arcade machine and table
2.	There must be a camera that follows the user's head movements in VR, both in rotation and position.	Pass	The camera can follow the user's head movements, both in rotation and position.
3.	The room must have some form of lighting.	Pass	There is a directional light in the room, in addition to the skybox lighting.
4.	There must be objects that follow the user's hand movements in VR. Tracked using the VR controllers. These could be hand models, controller models etc.	Pass	There are spheres that represent the user's hands in VR.
5.	The room must have a simple arcade machine or another model to house a screen to display the 2D game.	Pass	There is an arcade machine model which contains the screen.
6.	There must be object(s) that will be used to interact with the 2D game, for example, bridges, blocks, or stairs.	Pass	There is one object that can interact with the 2D screen, although more could be added by tagging an object with "VR Block".
7.	The room must have a surface or location to house these block(s) near to the screen.	Pass	There is a table for holding the objects, although it is not strictly needed, as the objects do not have gravity

2D Side-Scrolling Game			
8.	The game must have a player object and texture.	Pass	The player object has a static sprite texture.
9.	The game must have a background.	Partial	The background is just a static colour
10.	There must be obstacles for the player to avoid.	Pass	The game has several gaps to jump over, stairs to climb and level 2 has a 'stepping stone' section. Additionally, there are also enemies.
11.	There must be terrain for the player to walk around on.	Pass	The game has a base level terrain composed of tiled blocks.
12.	The game must have enemies for the player to avoid or kill.	Pass	The game has multiple (identical) enemies.
13.	The enemies must deal damage to the player if they come into contact with them.	Pass	The enemies deal 1 damage value if they contact the player when facing towards them.
14.	The enemies must move on their own.	Pass	The enemies' movement is controlled by a script that moves them in one direction until they hit something, at which point they turn around.
15.	The enemies must be able to be killed by jumping on them.	Pass	The player can kill enemies by jumping on them from above.
16.	There must be a camera that follows the player around the levels.	Pass	The feed camera follows the player around the levels and is bounded by the clamp variables.
17.	There must be an end goal at the end of the level(s) where once the player touches them it ends the level.	Pass	The levels have an end goal tagged "End Level". If the players hit this, then the level ends.
18.	At the end of the level, the game must display the user's score and go back to the main menu	Pass	When the user reaches the end of the level, their score is displayed for 3.5s and then is returned to the main menu.
19.	A main menu to choose which level to play.	Pass	The game has a main menu where you can select between 2 levels.
20.	A simple control scheme that is easy to use with VR controllers	Pass	The 2D game is controlled using the left joystick to move left and right,

			and the 'A' button on the right controller to jump.
21.	A UI to show the Player’s health, score and remaining time	Pass	3 text fields display the player's health, score and remaining time.
VR interaction with the 2D game			
22.	The 2D game must be projected onto the screen in the 3D environment.	Pass	The 2D game is projected onto the screen by using a feed texture from the camera applied to the screen.
23.	There must be a system to translate the location of 3D blocks that are moved into the screen using VR from the 3D space on the virtual screen to the 2D space of the 2D game so that they can interact with the 2D game.	Pass	When a block is pushed into the screen, its location is translated into a location in the 2D game space.
24.	When the camera moves after a 3D block has been placed, the block placed into the screen must follow the apparent location of the 2D block based on the camera movement.	Pass	When the camera moves, any blocks that have been placed are moved so that they appear to follow the location of the 2D block on the screen.
25.	When a block is moved by the above feature and reaches the edge of the screen, it must be removed from the screen and return to its original location in the environment.	Partial	When a block reaches the edge of the screen, it is removed and returns to the table to the right of the player. The block does go over the edge of the screen before this happens though, as the block's collider doesn't trigger the exit function until the innermost edge leaves the screen.

Non-essential functionalities:

These tests are against the non-essential functionalities or extensions to the app. These functions were determined to not be crucial to the overall success of the project but could potentially further the project with their inclusion.

Table 2.

Test Number	Functionality Description	Pass/Fail/ Partial	Comments
1.	Animations within the 2D game, for example, sprite animations for the enemy and player when they walk, jump, and take damage.	Fail	The only animation implemented was the enemy getting flattened when jumped on by the player and this was a crude implementation.
2.	Sound effects when actions happen in the 2D game. This would give the player some more feedback within the game.	Fail	The game has no sound.
3.	Multiplayer Capabilities. Perhaps allowing 2 players to compete at the same time, or one player to deal with the game and one to assist with the 3D blocks.	Fail	There is no functionality for playing with more than one person at a time.
4.	The ability to pause and leave games once a level has started.	Fail	There is no pause functionality. The user must wait until the game's timer runs out, or they must reach the end of the level to go back to the main menu.
5.	A system to keep track of high scores on each level.	Partial	The high scores are not independent of each level, as the script that controls the score doesn't know which level it is controlling. This means the high scores only work for level 1.
6.	Multiple different levels	Pass	The game has 2 different levels
7.	Collectable items such as coins in the levels.	Fail	There are no collectable items

User Questionnaire

The aim of the project was to develop a virtual reality (VR) application that demonstrates a novel way of interacting with a 2D video game in a VR environment. At its core, the application is intended for recreational use and as such, in addition to the technical success or failure of the project, users of the app needed to enjoy interacting with it. Furthermore, since the enjoyment of the app is subjective and may vary between users, I felt that it would be key for the success of this aspect of the app to be based on user feedback.

To gather user feedback, I designed a questionnaire with the intention of testing the application on a small number of participants (Appendix 1.). The number of participants I could test with was limited, as due to current Covid-19 restrictions, they had to be members of my household.

Ethical approval was sought and obtained from The School of Computer Science and Informatics Ethics Board for the inclusion of participants undertaking the questionnaire.

There was a small risk to participants using a virtual reality environment that they may suffer from motion sickness or may be at risk to themselves or their environment should they fall or collide with objects in their immediate vicinity. Participants, therefore, received the following advice included in the participant information sheet (see supplementary materials)

"Accessing a virtual reality environment means that users of the app will have a visual experience which often does not match the physical experience. In some users, this can affect their balance and rarely causes motion sickness. Potential participants who have a disability which affects their balance, suffer from severe motion sickness or Meniere's disease or who have an ear infection, are advised not to take part. Participants using the app are advised to remain standing still to avoid the possibility of losing their balance or bumping into objects in the "real" environment."

Potential participants confirmed they were happy to take part by signing a consent form (see supplementary materials).

Participants were asked to access the app and enter the virtual reality environment wearing the VR headset. They were then asked to play the game for 5-10 minutes before completing the questionnaire.

The questions were designed to specifically address the user's overall enjoyment of the experience, the acceptability of the design and their suggestions for further development and for the purpose of questionnaire analysis, the questions were categorised as such.

The results are summarized in the table below:

Questionnaire Results

Table 3.

No.	Question	Question Type	Mean Average Score (1-5) where 1 is not at all and 5 is extremely	Comments
1.	Did you enjoy using the game?	E	4.75	All participants scored 4 or 5
2.	Did you think the game was immersive?	E	4.75	All participants scored 4 or 5
3.	Would you be more likely to play the game if it had more levels or variation (assuming you had a VR headset)?	D	4.25	All participants scored 4 or 5
4.	Has the game made you more interested in VR games, or VR in general?	E	4.25	
5.	Did you think that the VR aspect of the game improved the otherwise purely 2D game?	D	4.5	All participants scored 4 or 5
6.	Did you find any obvious or otherwise serious bugs while you were playing the game?	D	No	All participants responded "no"
7.	If you noticed any bugs, how much did they affect your enjoyment of using the game?	D	NA	All participants responded "no" to question 6
8.	Did you like the VR environment?	D	4.5	All participants scored 4 or 5
9.	Would you be likely to play a game like this again?	E	4.25	All participants scored 4 or 5
10.	What did you like about the game?	F	Free type	Themes: - Degree of difficulty was good - Positive feedback regarding the use of VR environment
11.	What would you improve about the game?	F	Free type	Themes: - Additional functionality and more 3D shapes - Sound effects
12.	Do you have any other comments about the game?	F	Free type	Themes: - Great use of consistent 3D physics in the 2D environment - Suggestions for additional functionality

Key:

E = Question related to experience

D = Question related to game design

F = Question related to future development

The results clearly demonstrated that the application scored highly on experience and design. Participant suggestions for further enhancement included:

- Additional levels in the game.
- The inclusion of sounds and animation.
- The ability to see further ahead in the 2D game.
- “power-ups”, so when successful with a level, you gain more blocks to use in other levels.
- The ability to use buttons on the arcade machine.
- Additional 3D objects to insert into the 2D environment to enhance the capability of the game player.

The following are some examples of quotes from participants:

“VR is definitely more immersive than a standard “keyboard” controlled game.”

“I thought the fact that the 3D object when applied to the 2D game still retained the functionality that the VR 3D control gave was very innovative/immersive”

“Good integration of block into 2D game. Consistent 3D Physics”

Due to the pandemic and social distancing, the user group was limited to a small group of 4 users in my household. In the event of further development of the application, feedback would be sought from a group of at least 20 participants to allow an analysis of a wider demographic.

I noted that 3 out of the 4 participants had never previously accessed a virtual reality environment. This may have had the potential to impact their subjective analysis of their enjoyment and whether the VR component of the game significantly enhanced the enjoyment of a 2D arcade game. If repeating the questionnaire, it may be pertinent to include an additional question as to whether the participant had ever experienced a virtual reality environment in the past and analyse the results, comparing the responses from those participants with experience and those without.

I also noted that for question 6, no participants reported noticing any bugs. This may have been because they either did not notice any bugs or that they did not know what a ‘bug’ was in this game.

The raw responses from the questionnaires can be found in the supplementary materials.

Evaluation

I believe that the app has achieved most of the functionality I set out in the specification. This is evident in that all but 2 of my main requirements were met when tested. This means that most of the base functionality was present in the app.

One area that I identified for improvement in the app is the block being reset when it leaves the screen. This, therefore, appears inferior in design to a finished commercial game, as the block should not in any way leave the arcade machine before its position is reset. Currently, I think this is one of the most significant parts of the app which makes it look unfinished or amateur and could be confusing to users who were not expecting this behaviour.

Another main requirement that was only partially met was the background. I initially intended to have a dynamic background that would give a sense of depth to the levels by moving with a parallax effect behind. This was not implemented, although this is an aesthetic shortcoming rather than a functional one.

In terms of the non-essential functionalities, my app was not as successful. I think that this was mainly down to time constraints during development, rather than any inherent problem during development, as I feel that all the additional features which I did not implement would be relatively straightforward to add at a later date given more time. These additional features were also the last things I planned to do because they did not impact the success of the app. For this reason, I prioritised the more important features.

I do consider that there is a possibility for bias in both my own testing and the user testing in the form of the questionnaire. This is because the requirements by which it was tested were created by me, and in the case of my own tests, tested by myself as well. Despite this, I think that the number of requirements tested and met gives a strong justification for the overall success of the development of the app.

The group of participants could also have been more varied in terms of demographic and experience in VR or games in general. The group were also all connected to me, the researcher, so this could skew the results. Given the restrictions on the number of participants I could use, this was unavoidable.

I think that the core functionality of the app was received well by the participants, with all of them commenting that they liked the interaction method with the block. The aim of the project was to develop an app that demonstrated this method of interaction with the 2D game using VR, so is encouraging to see the participants commenting positively on this element of the app specifically.

Overall, I think that the development of the app went well, however, a large portion of the time developing was spent learning the Unity engine and C# language as I went along, and the time spent on this could perhaps have been better spent on some of the additional features or perfecting the main functionality. I also think that some of the scripts could potentially be refactored to reduce class dependencies, but due to a lack of time at the end of development, I did not get to do this.

Future Work

Although as stated in the Results and Evaluation section, as intended, I implemented almost all the main features of the app, there are still many features I would have liked to have added. These were not included primarily because of the inherent time constraints in the 12 weeks I had to complete the project, rather than because they were too difficult, or not possible.

If given more time, the first thing I would add to the app would be tweaking the VR Block's behaviour. One of the things that I noted from the evaluation was that when the block leaves the screen, it does not actually reset until the whole of it fully leaves the screen. This leaves it partially sticking outside the arcade machine, which does not look professional or realistic. There are a few ways this could be remedied. One would be to simply expand the borders around the screen or the actual walls of the arcade machine. This would give a small buffer for the block to go into before it goes far enough to be removed. Otherwise, a more complex collider on the block could check when the outer edge leaves the screen.

I also would have liked to make the rotation of the 2D block mirror that of the 3D block in the screen. This was not necessary and did not constitute a requirement, so I decided to focus on the position of the block instead. I would have liked to complete this, however, and it should only require similar calculations to the code for translating the position.

Another feature I think would add greatly to the experience of using the app is adding sound effects, and this was mentioned by several of the participants in the questionnaire. This would not be difficult to implement and would only require the scripts to also call and run these sound clips when actions happen.

Similarly, animations on the player and enemies would help significantly towards making the app look more finished, polished, and professional and would not be too complex to add as the player sprite I used for the player and enemy already has animations for many actions. This would just need integration into the player and enemy control scripts.

Another additional feature that I felt would be valuable after reflecting on the responses from the participants, was the possibility of an in-game tutorial or prompts on objects such as the interactive block, for example. This would make the game much easier to pick up, as I had to tell the participants how to use the app prior to them putting on the headset.

Finally, the enemies in the game are relatively simplistic in their movements and do not have any intelligence towards the player's location. An improvement to the game could be if the enemies had a more sophisticated AI to make them follow the player rather than just bouncing off objects they hit.

Conclusions

In this project, the goal was to create a virtual reality app that could demonstrate a novel method of interacting with traditional 2D games in a 3D VR environment.

The first objective I set out in my introduction was to create a 3D VR environment in which the arcade machine sits. I believe that this aspect of my app was implemented well, as I met all my requirements for this section. This objective was the simplest to implement, as it was all developed inside the Unity editor GUI, not requiring any code.

The second objective was to create a base 2D game to be displayed on the virtual arcade screen. I think that as I showed in my testing of the 2D game, I met almost all the main requirements set for the 2D game, and those that I did not complete were not crucial to the gameplay.

The final objective was to allow a user to interact with both the 2D and 3D environment by using the VR controllers to pick up the 3D blocks and interact with the 2D screen. This objective was the most challenging, both technically and timewise, as this was something that I had never seen done before in a game, and I could not find anyone online who had attempted it. With that said, my final solution was not overly complex, and a lot of the time spent developing it was due to my inexperience with Unity and C#. However, I do think that overall, I achieved most of this objective, as I met all the main functionalities set out, apart from the small visual imperfection when the block sticks out of the edge of the arcade machine before it is reset.

Because overall, all 3 of my objectives were met, I am confident in saying that I met the project's overarching goal of creating a virtual reality app that could demonstrate a novel method of interacting with traditional 2D games in a 3D VR environment.

I believe that my user questionnaire furthers the evidence for my conclusion that my goal was met, as all 4 participants reported enjoying the game, and finding it immersive, all scoring 4 or 5 for these questions. Perhaps more importantly, all participants thought that the VR aspect of the game improved the otherwise purely 2D game.

While I did not complete many of the non-essential functionalities, I do not think this detracted from the overall success of the project. As I set out in the specification, these were always meant as additional features that could improve either the app's ease-of-use or visuals. There was simply not enough time to develop these in the timeframe and this is something to reflect on and learn for future work or projects.

In conclusion, I believe that the project was successful at reaching its main goal and was highly complimented by all the participants after testing.

Reflection on Learning

This project was very challenging at times during development but was equally very rewarding once completed. Given the opportunity again, I would structure my work plan very differently.

While I planned to write the implementation section of the report alongside development, I quickly failed to comply as I soon became immersed in numerous iterations of programming. Unfortunately, I mostly wrote the section retrospectively. This was very difficult, as some of the functions I was writing about I had completed 2 months beforehand, and I could not remember a lot of what I had done. This resulted in much more time spent, as I had to go back and work out exactly what I had done. This inevitably impacted the quality of the report slightly, as some sections were not as detailed as I would have liked due to a lot of my time having been spent reviewing my work. If done again, I would have stuck more closely with my initial aim of recording all my development as I went along. Perhaps the use of a simple notebook to jot down key points of coding along the way would have been appropriate.

One aspect of the project that I felt helped more than I thought was the initial plan. I used this more than I thought I would to look back on and check that I was implementing the app as I had described in it. Without this, I feel I may have strayed more from my initial vision for the app.

I also think that my method of learning Unity and C# as I went along may not have been the best approach. I could have instead dedicated more time at the beginning of the project to go through tutorials to teach me how to use them. I spent a large amount of time at the beginning of the project getting familiar with the software despite not using a formal tutorial. I have learnt that preparation and planning are key, with the result that I would approach learning new software or programming languages very differently in the future.

I think that my method of self-testing iteratively during development was extremely helpful in informally evaluating sections of the code. This allowed me to add a section of code and immediately evaluate whether it met the requirement I was attempting to fulfil. Essentially, I was performing quality control of sorts to my design which meant that bugs were identified promptly, and a solution identified. This was obviously supplemented by the formal testing in the results and evaluation section.

Overall, I believe that while I managed to complete the project with relative success, there are several areas I can look to improve in the future. Chiefly, time management, and my approach to documentation during development.

These learning areas will stay with me for a long time, and I think the project is overall one that I have a positive reflection on.

References

1. BARNARD, D. *History of VR - Timeline of Events and Tech Development*. 2019 22/05/2021]; Available from: https://virtualspeech.com/blog/history-of-vr.
2. Howard IP, R.B., *Binocular vision and stereopsis.* New York: Oxford University Press, 1995.
3. *The Stereoscope*. The Times 1856 22/05/2021]; Available from: https://www.thetimes.co.uk/archive/article/1856-10-31/10/7.html?region=global#start%3D1856-01-01%26end%3D1985-01-01%26terms%3Dyour%20stereoscope%26back%3D/tto/archive/find/your+stereoscope/w:1856-01-01%7E1985-01-01/1%26prev%3D/tto/archive/frame/goto/your+stereoscope/w:1856-01-01%7E1985-01-01/1%26next%3D/tto/archive/frame/goto/your+stereoscope/w:1856-01-01%7E1985-01-01/3.
4. Weinbaum, S. *Pygmalion's Spectacles*. 1935 22/05/2021]; Available from: https://www.gutenberg.org/files/22893/22893-h/22893-h.htm.
5. Brockwell, H. *Forgotten genius: the man who made a working VR machine in 1957*. 22/05/21]; Available from: https://www.techradar.com/uk/news/wearables/forgotten-genius-the-man-who-made-a-working-vr-machine-in-1957-1318253.
6. *INVENTOR IN THE FIELD OF VIRTUAL REALITY*. 22/05/2021]; Available from: https://www.uschefnerarchive.com/morton-heilig-inventor-vr/.
7. *Virtual-Reality - Education and training*. 22/05/2021]; Available from: https://www.britannica.com/technology/virtual-reality/Education-and-training.
8. *VPL Research Jaron Lanier*. 22/05/2021]; Available from: https://www.vrs.org.uk/virtual-reality-profiles/vpl-research.html.
9. Oculus. *Oculus Rift: Step Into the Game*. 22/05/2021]; Available from: https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game.
10. Zhang, S. *Can VR Really Make You More Empathetic?* 22/05/2021]; Available from: https://www.statista.com/topics/2532/virtual-reality-vr/#dossierSummary.
11. *Unity Manual - XR*. 22/05/2021]; Available from: https://docs.unity3d.com/Manual/XR.html.
12. *Tetris Effect*. 22/05/2021]; Available from: https://www.tetriseffect.game/.
13. *Super Mario Odyssey*. 23/05/2021]; Available from: https://www.nintendo.com/games/detail/super-mario-odyssey-switch/.
14. *Unity Asset Store*. 23/05/2021]; Available from: https://assetstore.unity.com/.
15. *Arcade Machines Pack 01 - Lowpoly Pack*. 23/05/2021]; Available from: https://assetstore.unity.com/packages/3d/props/arcade-machines-pack-01-lowpoly-pack-73020.
16. *Hero Knight*. 23/05/2021]; Available from: https://assetstore.unity.com/packages/2d/characters/hero-knight-167779.
17. *Free 8-Bit Pixel Pack*. 23/05/2021]; Available from: https://assetstore.unity.com/packages/2d/environments/free-8-bit-pixel-pack-79530.

Appendix 1

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYD

Virtual Reality App Questionnaire

Dear Participant,
You will now have spent 5 – 10 minutes using the app and playing a game in the virtual reality environment. Please respond to the following questions to the best of your ability. The questionnaire should take no longer than 5 minutes to complete. All participant details and answers will be treated confidentially.

1. Did you enjoy using the game?
(Answer by circling 1-5, where 1 is not at all and 5 is extremely)

1	2	3	4	5

2. Did you think the game was immersive?
(Answer by circling 1-5, where 1 is not at all and 5 is extremely)

1	2	3	4	5

3. Would you be more likely to play the game if it had more levels or variation (assuming you had a VR headset)?
(Answer by circling 1-5, where 1 is not at all and 5 is extremely)

1	2	3	4	5

4. Has the game made you more interested in VR games, or VR in general?
(Answer by circling 1-5, where 1 is not at all and 5 is extremely)

1	2	3	4	5

5. Did you think that the VR aspect of the game improved the otherwise purely 2D game?
(Answer by circling 1-5, where 1 is not at all and 5 is extremely)

1	2	3	4	5

6. Did you find any obvious or otherwise serious bugs while you were playing the game?
(Answer yes or no)

Yes	No

7. If you noticed any bugs, how much did they affect your enjoyment of using the game? (Answer by circling 1-5, where 1 is not at all and 5 is extremely)				
1	2	3	4	5

8. Did you like the VR environment? (Answer by circling 1-5, where 1 is not at all and 5 is extremely)				
1	2	3	4	5

9. Would you be likely to play a game like this again? (Answer by circling 1-5, where 1 is not at all and 5 is extremely)				
1	2	3	4	5

10. What did you like about the game? (Please write or type your answer in the box below)

11. What would you improve about the game? (Please write or type your answer in the box below)

12. Do you have any other comments about the game? (Please write or type your answer in the box below)

Many thanks for taking part.