Just A Normal Squash VR Game – Using a Muscle Recognition and Linear Accelerometer Approach

Submitted May 2017, in partial fulfilment
of the conditions of the award of the degree Bsc Computer Science

Yousef Alaw

4222792

Supervised by: Peter Blanchfield

School of Computer Science and Information Technology
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____

Date _____/_____/_____

Abstract

Recent development in technology has made it possible for virtual reality to give more immersion to the user and soon we will achieve true VR, which is the complete immersion of the user in a world where he/she will have all their senses convinced that they are in another reality. However, the difficulty in achieving this lies with the use of controllers (such as Xbox, ps4, pc, controllers.), as this ruins the atmosphere of what is supposed to be VR.

That is why the aim of this project is to provide a way to use VR without the use of physical controllers. Instead the plan is to use the Myo armband (which is a muscle gesture recognition hardware) to act as a controller, the user will put these on their arms and by using certain gestures they can control the in-game player.

Myo armbands have had a great impact on recent use of technology, especially in medicine which will be further discussed later, showing how useful the Myo armband can be is another aim of this project.

By combining both VR and Myo armbands the user should have a more positive outlook on the VR experience. A squash game will be built using Unity to demonstrate the capabilities of combining both the VR headset and Myo armbands. Within the squash game the user will be able to move around using the accelerometer of the phone and can control the player using the Myo armbands, essentially the project can be split into 2 sections; the VR and the game physics, the controller and movement of player.

Acknowledgements

I would like to start by thanking my supervisor, Dr. Peter Blanchfield, who allowed me to build a project that I was fascinated by, because of the flexibility I was given, this project was made possible.

I am also grateful to the three people who will be marking my project on the demonstration day, past its original deadline.

Finally, I would like to thank Dr. Ender Özcan who has been supportive and gave me the courage to finish this project, despite having to deal with great difficulties.

Contents

Chapter 1. Background

1.1 Introduction

The intention of this project is to create a highly realistic VR game which uses the user's arm to control the player and the phone's accelerometer to control the player's movement. The game created will be a squash game, following regular squash rules [Rul1], Though originally the designed game was to be an FPS (First Person Shooter) this changed when talking to my supervisor, as he suggested that a squash game will be more exciting and it would demonstrate the Myo armbands capabilities fully, as accuracy is required to detect certain muscle motions that the user imposes.

1.2 Motivation

This project defines a new way for users to interact with the digital world, whether it be for gaming, interactive movies, or even for medical use, consider these following cases:

Alternative Method for Gaming

Up until now gaming has been dominated by the use physical controllers such as the Xbox, ps4, etc. there have been several attempts to popularise the idea for hand motion controllers (Wii devices) but the prior option remains dominant. However, using both VR and muscle gesture recognition hardware there can be a new successful alternative method for gaming where the users truly feel immersed.

Medical Use

Allowing surgeons to control information in real time without touching anything is very important, as usually while operating the surgeons require additional support from other people to help show vital information on the screen so that the surgeons know what is currently happening. Myo armbands change this as it gives the surgeons total control which reduces miscommunication amongst two doctors which can occur during an operation.
Furthermore, doctors, junior doctors and even surgeons can use both the VR headset (360°) and Myo armbands to help simplify their tasks or even give them the ability to experiment certain operation before practicing them. Junior doctors can greatly benefit from this project which can vastly improve the training for doctors.

Overall

This project has a lot of applications and can help a lot of people out, the idea is very fixable too and can be used with different devices. It also signifies our current technological standpoint, which has increased significantly over the past few years. The current biggest limitation must be cost, as having to buy VR headset and the Myo armbands are quite costly, though this is not an extreme concern as prices decrease over years when it becomes simpler to design and implement such devices.

1.3 Description of the work

The project will take the form of a VR squash game built using Unity. The user starts in a squash court where they can move in 360°, and play against an enemy AI opponent (possible to change if time becomes an issue), the rules of the game are the same as a normal squash game [Rul1]. However, to make the game more interesting and interactive there will be several events that will be triggered as the user progresses through the game. The game will be played through a first person point of view where the user can see only the player's hands, this was done to keep things as real as possible, it also synergises well with the Myo armbands as well as the VR headset.

1.4 Related Work

A lot of similar work has been done before on virtual reality and while it is true that Myo armbands have only recently been released there has been a lot of applications made to it. Virtual reality is a very old concept in fact it was not until 1990's when Sega released the Sega VR and popularised the idea of VR, consumers liked the idea and even came up with more futuristic ideas of what VR will be

like, sadly VR did not progress at a rapid state as with other successful technologies. 2012 was when the oculus rift showed what VR still has a lot of potential and is worth developing.

This project uses various of different concepts that have been researched on before, for example allowing the user to move based on their actual movement using the phone's accelerometer is a common issue known by researchers, one solution is to have multiple sensors [Acc1] [Acc2] on the human body and track the body's full motion (when the player steps forward move forward), an impractical solution for my project. Other methods include calculating the displacement of the device (phone) and based on its acceleration output fit in the data of acceleration to get the speed of the device (Z axis) which you can then use the speed to move the device. Phone accelerometers have yet to be perfected, they have the basic tools for getting information by tilting the device or even the gravitational influence on the device but lack in accurately getting the devices position change, like mentioned above linear accelerometer is a way for achieving this (other than GPS) however it is very reliable.

Although this projects shares a lot of similarity with other concepts, it is quite unique in the fact that the combination of VR and Myo have only recently been researched and developed on, other comparable systems that exist include systems like the HTC VIVE [Hv1], Oculus touch, and other examples include in researches that combine both VR with hand motion devices such as Wii remote, ps4 move, etc. Everything happen to use a form of controller, this is mainly because controllers that have direct input are very accurate, e.g. move analogue stick up to move forward, The Myo armbands have only recently been developed and so the application with it to VR is still relevantly new, more time is required for it be reliable. The squash game that will be developed will be simple enough so that the accuracy problems does not occur or at the very least reduced significantly, this is due to the few controls that need to be bind with the Myo armband gestures.

Although similar systems do exist, most fail to capture the purpose for this project, which is the use of muscle gesture recognition instead of hand motion controllers or direct input controllers. Using this method instead the project becomes rather novel, and so it must be said that a lot of problems may occur however the finding out and determining whether this method (VR & Myo armbands combination) is feasible or not will be the second objective for this project.

1.5 Insight of the problem

Using the phone's accelerometer sensor, the player will be able to move around based on their real-time movement (closely related to GPS system used in Pokémon Go), however this may prove to be challenging as currently smartphones do not have the sensor required to achieve this accurately. Even with using a gyroscope sensor which is very like an accelerometer it cannot be done accurately, that is why Pokémon Go decided to use a GPS system instead, however doing something like this is not very practical for a squash game, one solution could be to use a linear accelerometer. Another issue is that the Myo armbands come with a few gestures which limits the user's interaction with the game, this may reduce the immersion experience, however the game in principle is quite simple therefore the player's movement should be limited regardless the involvement of the Myo armbands. Finally, the enemy AI, this will be the toughest to implement as the AI must not only play in increasing difficulties level but also can mimic human like decisions, to make the game 'fair'. As the user progresses through the levels environments will begin to change and certain events will be triggered, together the game should feel very immersive and fun.

Chapter 2. Design

2.1 Introduction

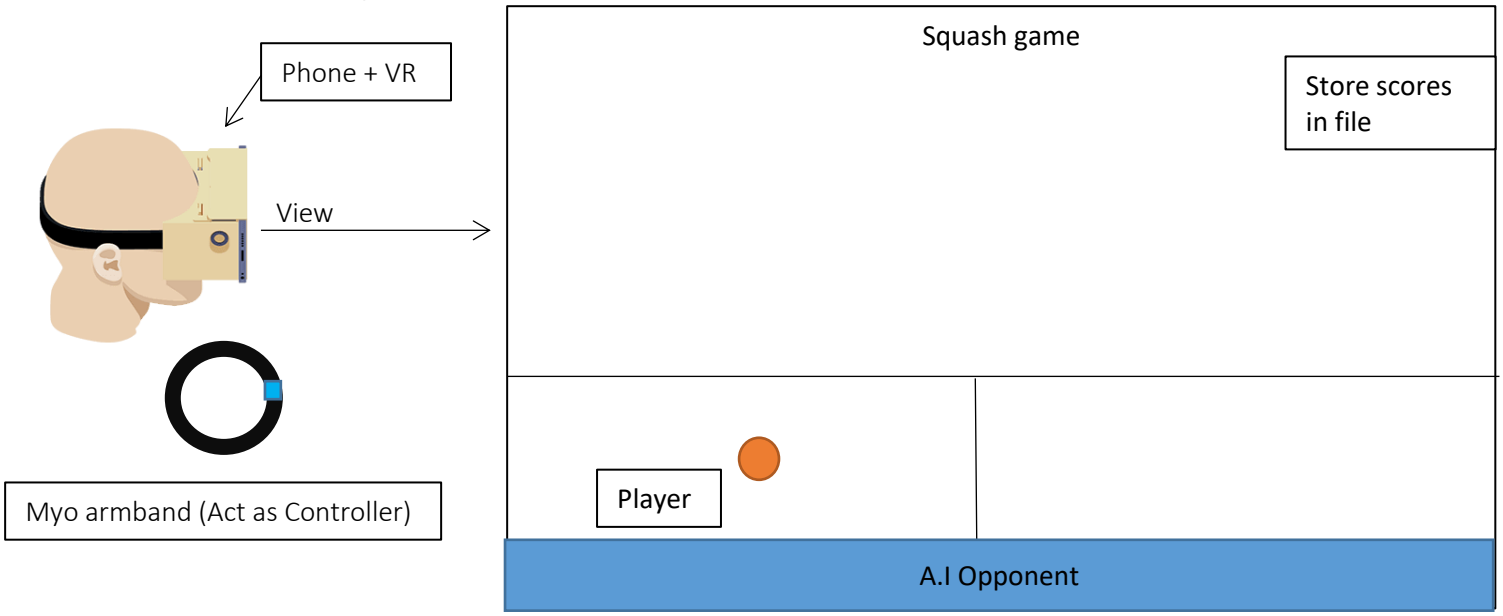
In this Chapter the design of the game will be discussed, understanding how the Myo armband interacts with a device is important as it will be mentioned later, there are certain limitations depending on which device is used, the fact this is a VR game brings forth other obstacles that need to be dealt with, such as the size of the environment, the size of the player, movement of the player, the pace of the game, the difficulty of the game, and so on. When it comes to the actual game mechanics there isn't new that needs to be looked over, as the game has a fairly simple goal and sub-goals, this will be explained below.

2.2 Gameplay

The game will begin with a start menu, which also explains the rules of the game, it will also ask for the user to connect their Myo devices with the phone (this is done by tapping the Myo armband against the phone), once the user is ready they can start the game by double tapping their finger against their thumb, this will signal to the Myo armband to start the game (by binding to the control). The player is then matched against an easy AI enemy which will increase in difficulty as the user progresses through the game.

The AI will start with serving the ball and his goal is to try to win the game, by hitting the ball at different angles to make the user miss the ball.

The winning condition is met once the first player who reaches 9 points at which point the time and date and rank is saved in a scoring system which will also be created and can be viewed by the player at the end of the game.



The game uses a first person (looking at the scene as if you are the character) perspective to control a 3D character. This was done to give a more immersive feel to the game, as the goal is to try to convince the user that they are controlling the arms of the character. Figure 2.1 shows the character in a holding stance (when holding the bat).

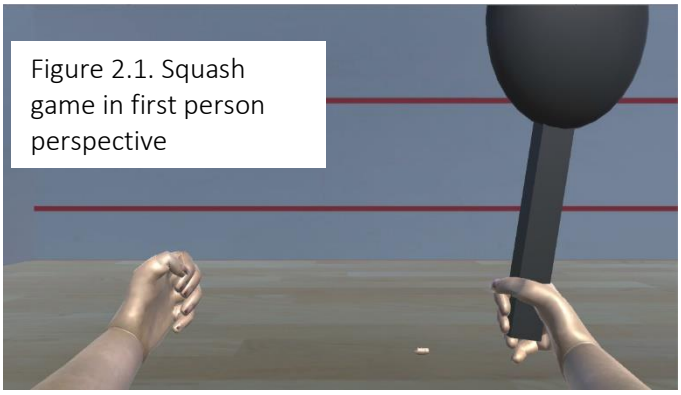


Figure 2.1. Squash game in first person perspective

2.3 Environment

As mentioned previously the scale of the game objects need to be carefully considered as this is a VR game, and having object sizes such as ball being too small will cause a lot of confusion to the user, therefore to make things clearer bigger size objects will be used. The character model will also be slightly large compared with the environment as this will make movements easier. Since the player movement are done based on the linear accelerometer of the phone, having bigger steps taken per direction is ideal as linear acceleration is not an efficient method for gathering continuous input, therefore to work around this larger steps have been added. The efficiency of each method of movement will be discussed later.

As the environment is made from many cube objects (walls) and a plane object (floor) the collision detection is simple as the Unity physics engine already has this implemented, However, to have the user and the A.I interact with the environment, a trigger event was added and based on objects getting triggered certain events were carried out.

2.4 The player and the A.I

A.I follows a very simple reactive architecture, the idea is basically to allow the user to hit and receive ball shots, the A.I will do this by remembering who's turn it is, and when it is the A. I's turn it will hit the ball at a certain velocity further away from the user, this way the user will need to move more to receive the ball and if the user misses the ball then the point will go to the A.I. Though originally the plan was to create a smart A.I that would mimic human like behaviour to give the player an equal chance to win, as it is rather easy to program the AI to always hit the ball in the most favourable position where the user will be unable to hit the ball back or have a hard time, therefore decreasing the difficulty of the enemy AI is important, One way of doing this Is by adding a rand() value which can decrease the accuracy of the enemy AI's goal (where ball will land), and the rand() value can decrease over time (as player gets better) close to the actual value which will give the impression that the enemy AI is increasing in difficulty. Another way which this can be done is by using ma-chine learning algorithms such as neural network to allow the enemy AI to learn and become better over time, so initially the game will be rather easy but over time as the AI learns the player will have a difficult time (hard mode). Although this method is the most appealing and efficient way of doing things (since the enemy AI is performing somewhat human like behaviour), it requires myself to do more research on the topic to be able to implement such AI on the game. However, understanding the Myo technology took longer than anticipated and therefore this did not work out.

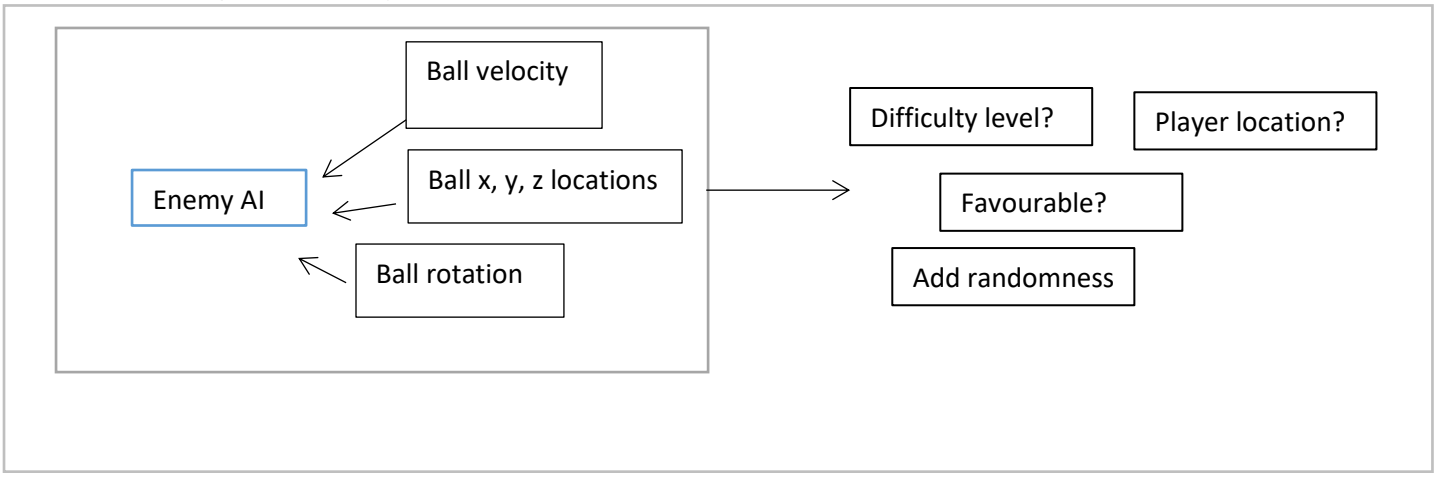


Figure 2.2: Diagram showing the basic inputs that enemy AI requires.

2.5 User Control and Equipment

The user will be controlling the character in game using both the phone and the Myo armbands, for the movement of the character, the input will be the z and y axis of the phone's linear accelerometer, the code in figure 2.3 shows an example of how this would look like. The linear accelerometer values start at 0 and depending on how fast the velocity of the phone (against a certain direction) the values (x, y, z) will increase or decrease, e.g. pushing the phone backwards at a high velocity could give a z-axis value of 8. This method can have a big flaw, which is that it is not very accurate while putting on a VR headset as the user will need to constantly push the device slightly at different axis to get a valid input, this can be very annoying, however, this can be reduced by using lower y and z axis values as the device will only need to be pushed at very suttle force, the disadvantage of this is that movements may be made where it was not the user's intentions. Most phones have a built-in accelerometer and so this method is highly accessible to most devices, and allowing the user to play inside is another reason why picking this method over GPS was used.

There are several actions that the character can make, these are; swing racket in forward direction(smash), swing the racket in back direction (back swing), grab action (which grabs the ball) and release action (to release the ball), these have each been animated using Blender and the model of the arm has also been modelled using blender, and each animation is played based on when certain actions have been made, the Myo armband will serve as a controller and based on the hand gestures made certain actions will be made, for example waving your hand in will call the swing function which in turn will play the swing animation, the figure 2.4 shows the code snippet for this example, and figure 2.5 shows the animator window for each action.

```
if (zval >= -3.15 && canmove)
move(transform.position + vector3.forward)

where zval is the linear acceleration z value
```

Figure 2.3: Code snippet that moves the character forward when phone has been pushed forward (z-axis).

```
        switch (myoPose)
        {
            case MyoPose.WAVE_OUT:
                isSwingOn = true;
                animeR.SetBool("isSwing2", true);

                StopSwing();
                break;
```

Figure 2.4: Code snippet that plays the swing animation when Myo detects the wave out motion.

The Myo armbands are highly accurate which makes the gameplay satisfying as the user will feel as if they have really hit the ball, the main issue with the armbands lies with the fact that there is yet no clear way of adding two devices to a phone, this is caused since the Myo devices use Bluetooth to connect to the device and trying to connect to devices causes an interference to occur which does not allow for multiple devices to be connected to the phone, other platforms such as a computer will not have this issue as each Myo device has its own connector which allows for the pc to understand which device is currently being used (even both simultaneously).

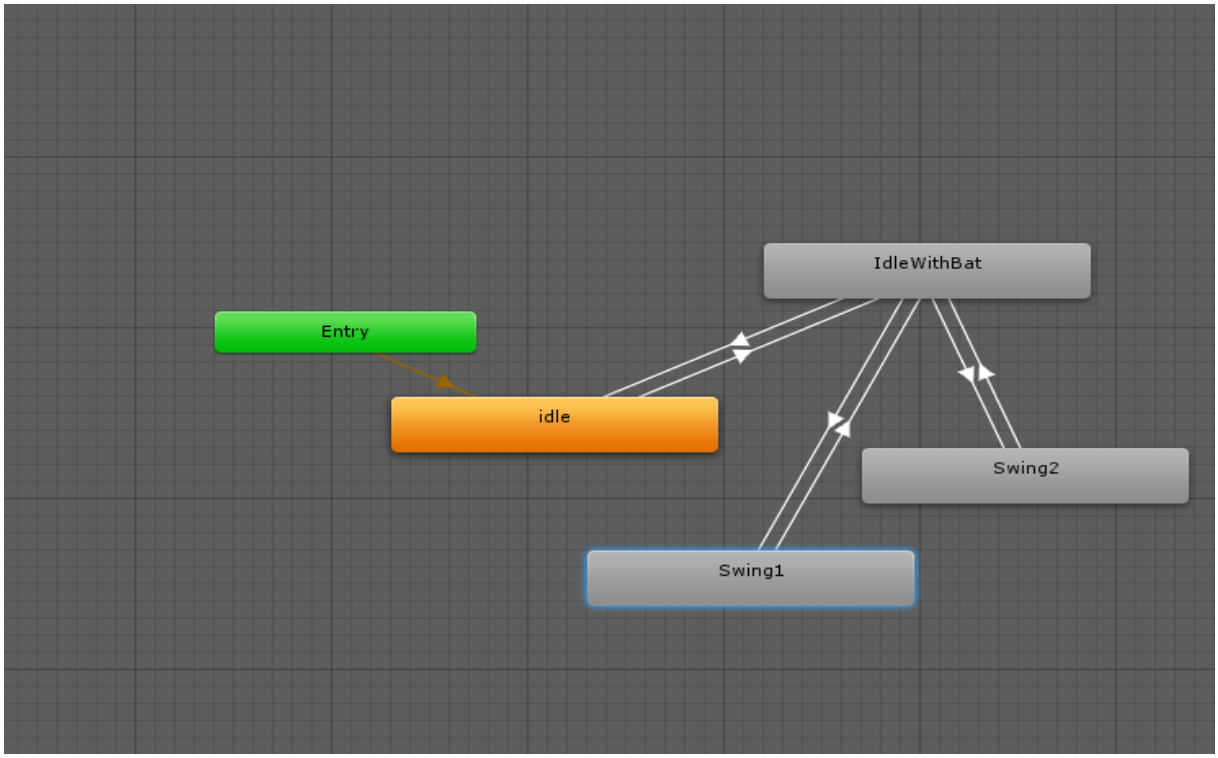


Figure 2.5: Animator window for different actions that are all linked together.

2.6 Libraries and tools used

Numerous of tools and libraries were used throughout the design and development phases to both simplify complicated tasks (connecting the Myo devices to the phone) and to re-use existing tested and popular libraries, rather than creating my own.

SuperSensor library was used to gain access to all the phones accelerometer properties, it works by using the Android SDK to call the methods such as get linear accelerometer, Unity itself does not allow for direct access to these properties. SuperSensor was selected as it provides the linear accelerometer property that is required and has a simple syntax, for example to get the axis values you simply write Sensor.Instance.OnLinearAccelerationChanged += (value){…}, The fact that this library has access to the phone's GPS system is another great reason to keep it as it means future development on the application can be done easily.

Myo Unity Android Library was also used to allow for the direct connection between phone device and the Myo armband, this library is still in development as Myo is still a new technology that is been worked on every day, future development such as allowing the use of two or more Myo devices will be added too. This was the only library that was available to use to allow for the development on an android device, other methods that were available included using a computer instead of a phone, however this removes the great portability use that this project provides.

Photoshop was used to draw graphics for the game. Photoshop is a great image editor as it provides a lot of flexibility to how much you can edit on an image, it was developed using c++. The textures were all done using photoshop and were exported as jpeg format (low memory usage yet provides high quality images).

GoogleVR SDK [GVR0] was used to provide a virtual reality environment, since the used VR platform was the Google Cardboard, this SDK was needed, GoogleVR SDK also allows for development on other platforms such as Daydream. GoogleVR provides a low-latency way of working with VR mobile games, which is key for an immersive game.

The meshes and models were all done using Blender. Blender is a free open source 3D creation suite, able to model, rig and handle animations, coded in C, C++ and Python. Having a huge community allows for free models and help which is primarily the reason Blender was used.

Finally, everything was put together and developed using Unity, which is a very popular game engine, and though it is not fully optimized compared with doing it using c++ and OpenGL, it is still highly useful and runs smoothly. Unity is highly popular for the fast development of games, it has an already built in physics engine, scripts can be attached easily to each component and other libraries that were used all support Unity which makes it a great intermediate application.

2.7 Game Objects

Game objects are used to represent each individual component of the game world. A game object in unity is essentially a container, adding pieces to the game object will allow for larger more complex objects such as a character or a ball.

The game objects used are listed below, the description of each game object has been given and how they are expected to behave.

Court made from many cube objects, defining each side of the wall and a plane which is used as the floor.
Player The most complicated Game object, made from many sub-game objects also contains a camera inside, the player is a controllable representation of the character in the game world.

- **Bat** A sub group of the player hand object, held by the right hand of the player, made by the combination of a cube object and flat sphere object (very simple representation).

Canvas To show U.I. such as the score, buttons in the game world.
Ball Made using a sphere object and is the main method to define the game.
A.I agent Created using a big cube object which is set to invisible and on trigger (only activated once a condition is met)

The main interactions are between the player, A.I opponent, walls and the ball the A.I is programmed to only perform an action once every other turn, and will only interact with the ball game object. The player object makes use of multiple objects; the bat, arm models and the ball. The figures below show each object.

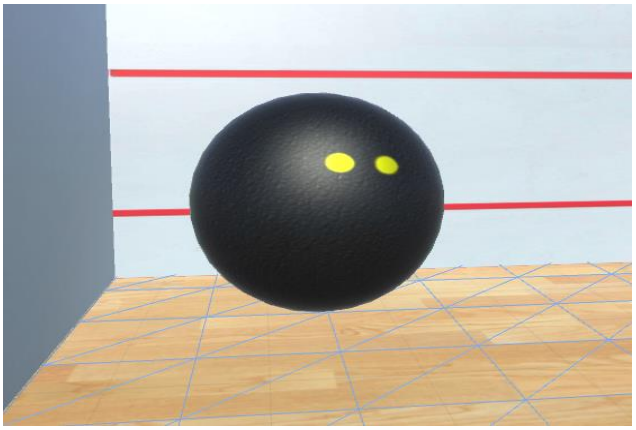


Figure 2.6: A squash ball designed in Unity

Figure 2.7: The green line shows the A.I agent

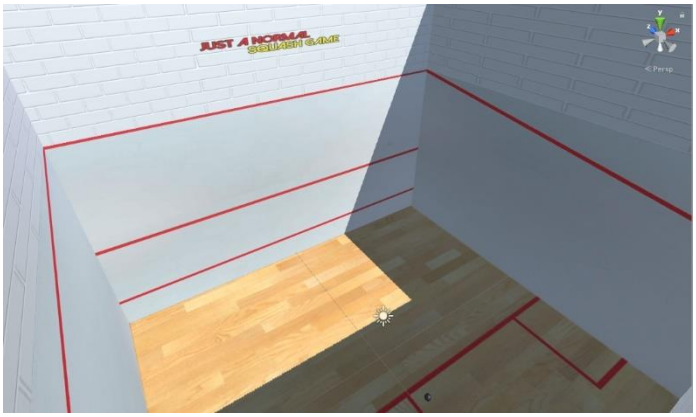


Figure 2.8: The court, created using cubes and a plane

Figure 2.9: the UI interface

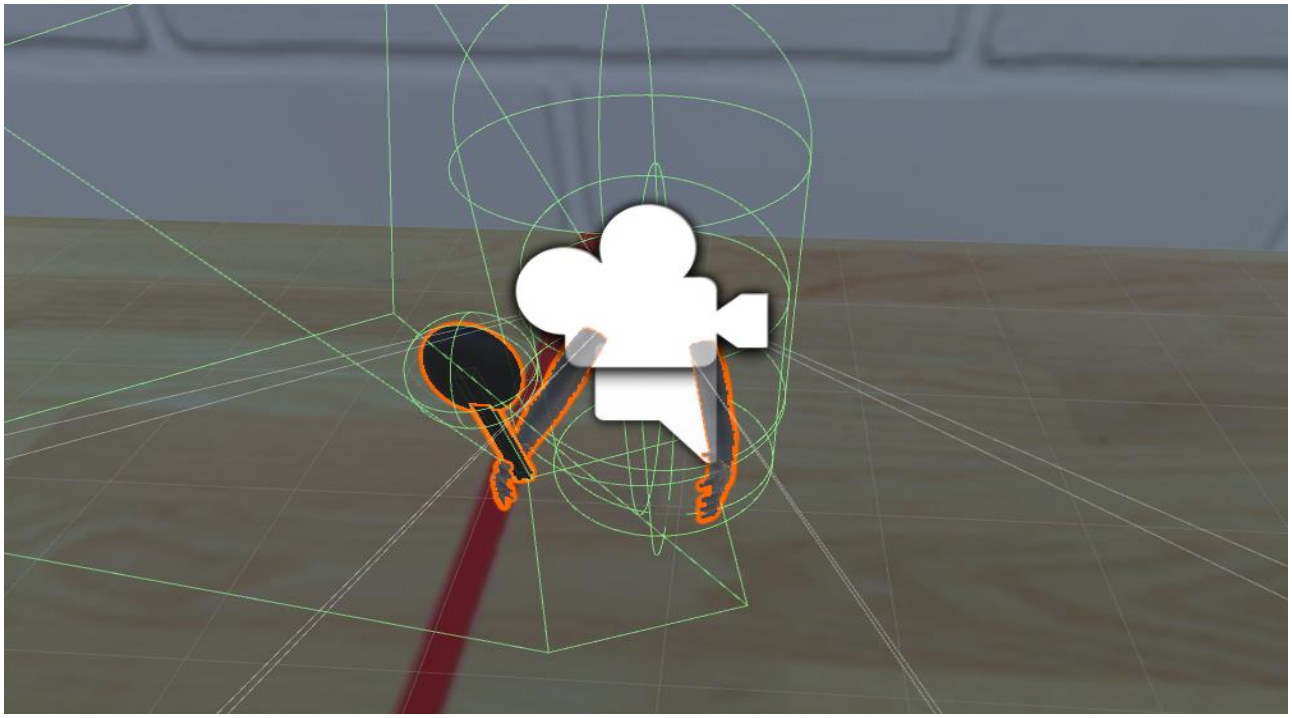
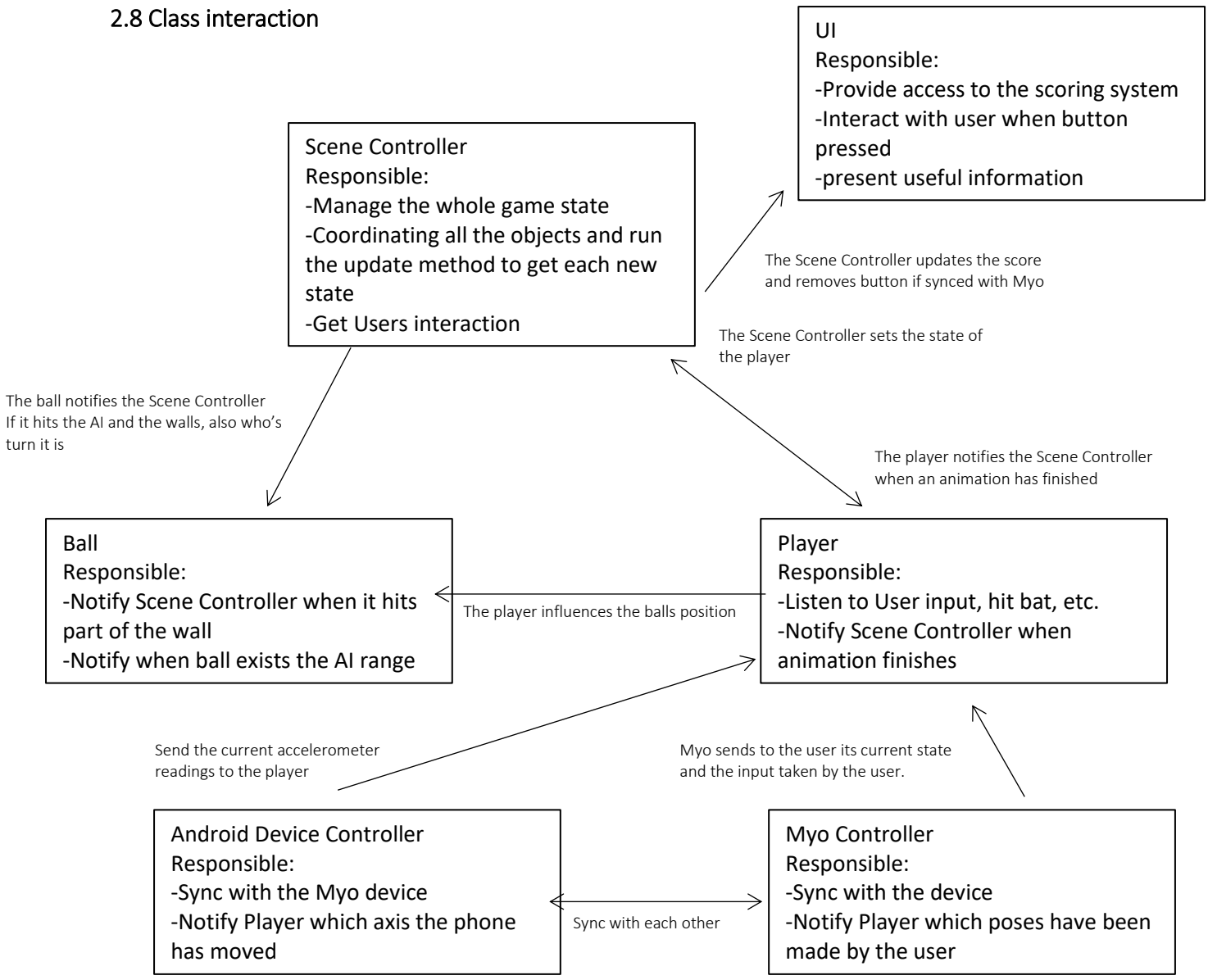


Figure 3.0: Player object, encapsulated, composed of the arm models, the camera, the bat, and a trigger event object.

2.8 Class interaction



Chapter 3. Implementation

3.1 Introduction

this chapter describes how the designs were taken forward and implemented. Furthermore, the implantation of the external tools (Myo armbands, phone accelerometer and GoogleVR) will be discussed, also how the overall development of the application went and what difficulties were faced along the way.

3.2 Programming Languages Chosen

Unity was the most appropriate platform to use for this project for the following reasons:

1. Was compatible with all SDK that is required for this project, i.e. android SDK, Myo SDK, Google VR, FPS character libraries.
2. Easy to use and can create models (maps, game level) all in one platform.
3. Compatible with multiple platforms (PC, IOS, Android).
4. Overall can be interconnected with external programs/libraries easily.
5. Already built in game engine that may require few small adjustments.

Therefore, by using Unity there were two options to either create the scripts using JavaScript or C#, both languages were similar in what they provided for the project. Learning Unity (following tutorials), they used C# and being comfortable with this language my project will be written in C# also.
As described before there were some uses of Java (Android SDK) to create a prototype application for using the phone's linear accelerometer property, which helped greatly in understanding how accelerometers work.

3.3 Modelling and Animations

As mentioned previously the tool used to create the models and animations was Blender. One advantage of using blender is that the ability to create models is made very easy as there are free online models can easily be imported into Blender and amended in any way possible, and the animations work by registering initial keyframes and then changing the position, rotation and scale of

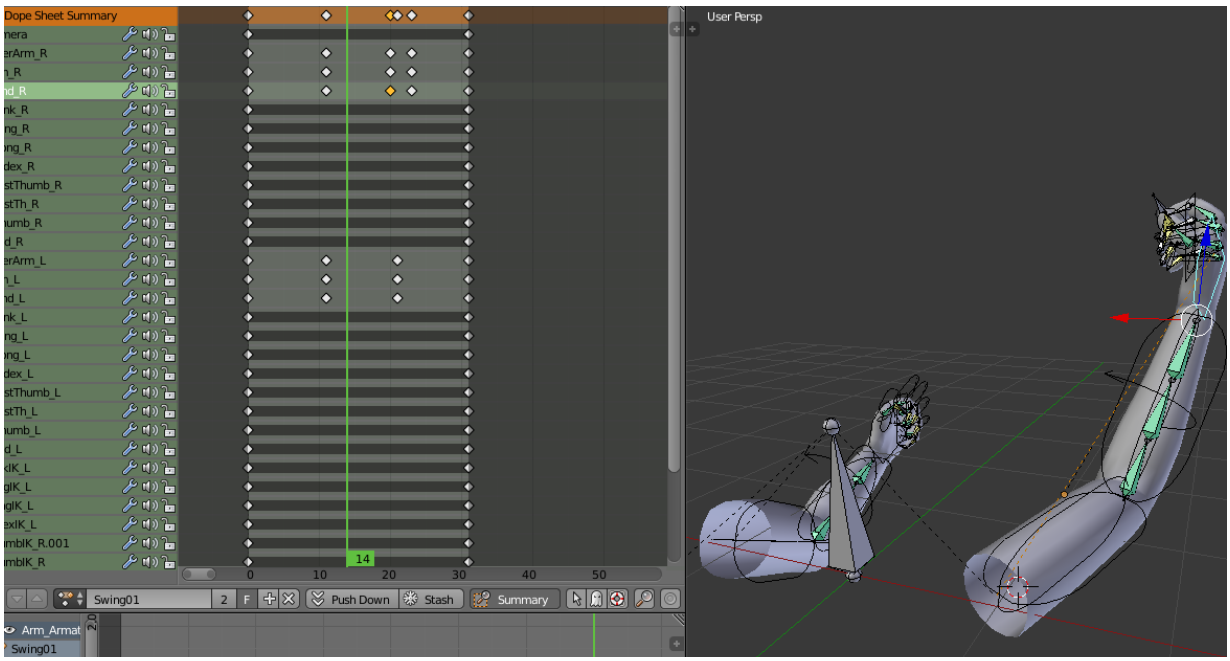


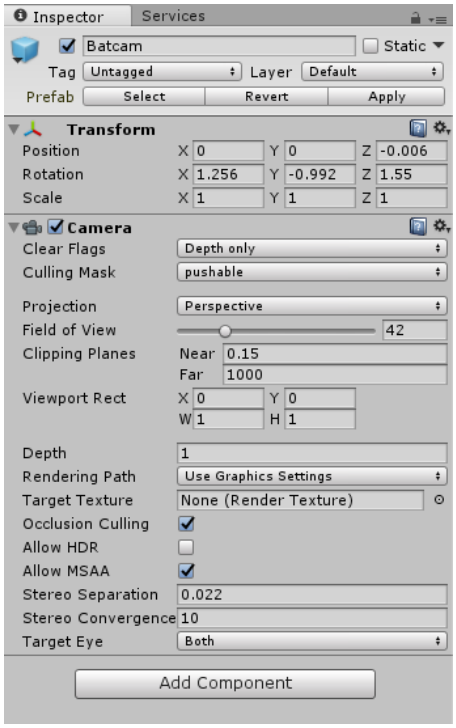
Figure 4.0: Keyframes added based on time, the dope sheet summary on the left is each object that makes the arms and how they've been modified per each keyframe placed.

the objects individual components (such as hand) and placing another keyframe, the figure 4.0 demonstrates this. Models were downloaded from free model sites [Mod1] and amended to suit my game, though it was possible to create the models from scratch, the process was lengthy and being able to change already made models, allowed for more time to work on the game mechanics and the connections between the controllers and the player.

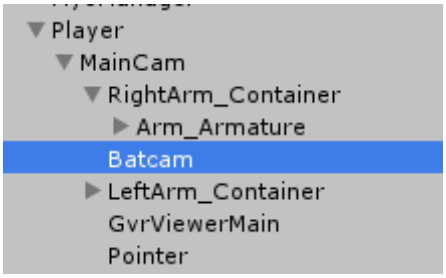
The arm models originally came as 2 arms combined, this was not ideal for the application as if there needed to be integration for 2 Myo armbands the models will carry out the same actions from both Myo armbands. Instead the arms needed to be separated, this was done by recreating the left arm. Having the application to be ready for future implantation was key for this project, as it makes it easier for future development on the game.

3.4 Integrating 3D Models into Unity

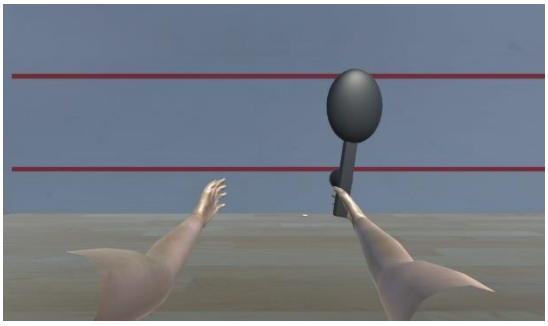
Importing the models into Unity was the next step, Unity makes it easy to play with the models and link up together each animation with the tool Mecanim (as shown in figure 2.5). A very important implantation for models was to create 2 cameras one which focuses on the general scene and the other which only focuses on the arms, bat and the ball. This was done to prevent objects not going through the arms or bat, it also made it very simple to insure clipping does not occur, the figure 4.1 shows the clipping settings as well as the additional camera added.



(A)



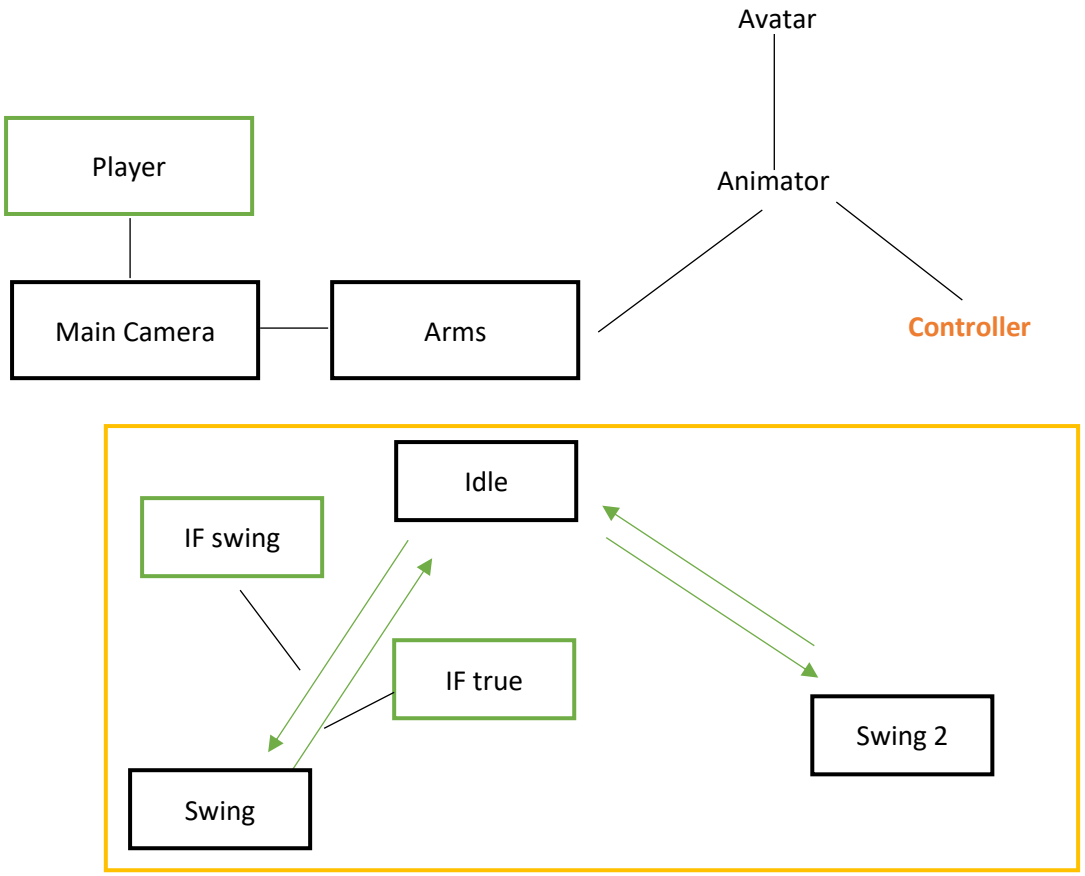
(B)



(C)

Figure 4.1: Second camera settings (A, B), as well as an example of bad clipping (C).

Unity's Mecanim tool is very powerful, it is essentially an animation state machine system, where by different states are made that play animation based on certain transition logic which is defined by the programmer. The Mecanim diagram below is a very simple design to show how the game player transitions between different states.



3.5 Gameplay

The main objective of the game is to win against the A.I opponent, this can be achieved by hitting the ball in a location where the A.I cannot return the ball, edges on the opposite side of the opponent are usually a very good location to aim for. Once won a point that player will become the new server and add a point to their score, matches are usually best of 3 or 5 games, and each game has the scoring up to 11, first player to achieve the score will win that round. This section will be split into 3 parts, firstly the discussion of the A.I agent and how the agent knows when it's their turn, secondly the player and what actions are available to him/her, and finally how the overall gameplay should flow and what is to be expected from a typical game.

3.5.1 A.I. Agent

AI-controlled opponents are generally used as support characters or as opponents. Most games however tend to create AI agents to challenge the user, example is the Quake game, Quake III Arena's Bots [QU3] which has one of the best AI agent for a game which has been reported to beat the best players in Quake, however because it is more of a bot which gets the possible

Figure 5.1: Code snippet showing how the AI figures out who's turn it is.

```
void OnTriggerEnter(Collider other)
{
    if (other.tag == "walls") {
        if(turn == 1) //AI turn
```

```
if (turn % 2 == 0) //every other turn
        {
            ++turn;
            turn %= 4; //reset
        }
```

future movements of the user by simple tracking algorithms, and use many external information to make prediction, you can call it a smart computer as opposed to an AI, they aren't necessarily intelligent. The squash game AI opponent will also follow a very simple solution, it will be fed data such as the location of the player, the ball velocity and adding a bit of randomness to each shot. The A.I figures out who's turn it is using the code below (figure 5.1)
The way the AI commits actions is every time the ball is within its box collider, when this happens Unity calls the onTriggerEnter function which is where the code can be written to compute the A. I's decisions.

3.5.2 Player

As mentioned before, the player has a few actions available to it, and based on the input, certain actions will be made. However, the player's goal in the game is to win as much as point as possible (same as the AI). Traditionally, these types of games tend to focus on rapid movement and aiming skills, not intelligent computer opponents. One great improvement for this application is to introduce multi-player gameplay, this way the immersion and excitement of the game is amplified even more. The players input and actions will be discussed in the player input section.

3.5.3 Overall Gameplay flow

The rule of the game has been explained above, and the roles of each player has also been explained. The figure 5.2 shows how the environment appears from a top-down view, the player can move anywhere in the green box once served from his side (each side has a serving box, and they most serve diagonally). An example of how the game is expected to be played out is the following:
AI servers the ball -> hit against the wall at the front and bounce back to the side of the player.
Player can then hit the ball -> anywhere against the wall and give a tricky shot to the AI.
AI tries to return the ball -> if misses then player gains 1 point, if not then players turn.
This is repeated until first player to reach 11 points.

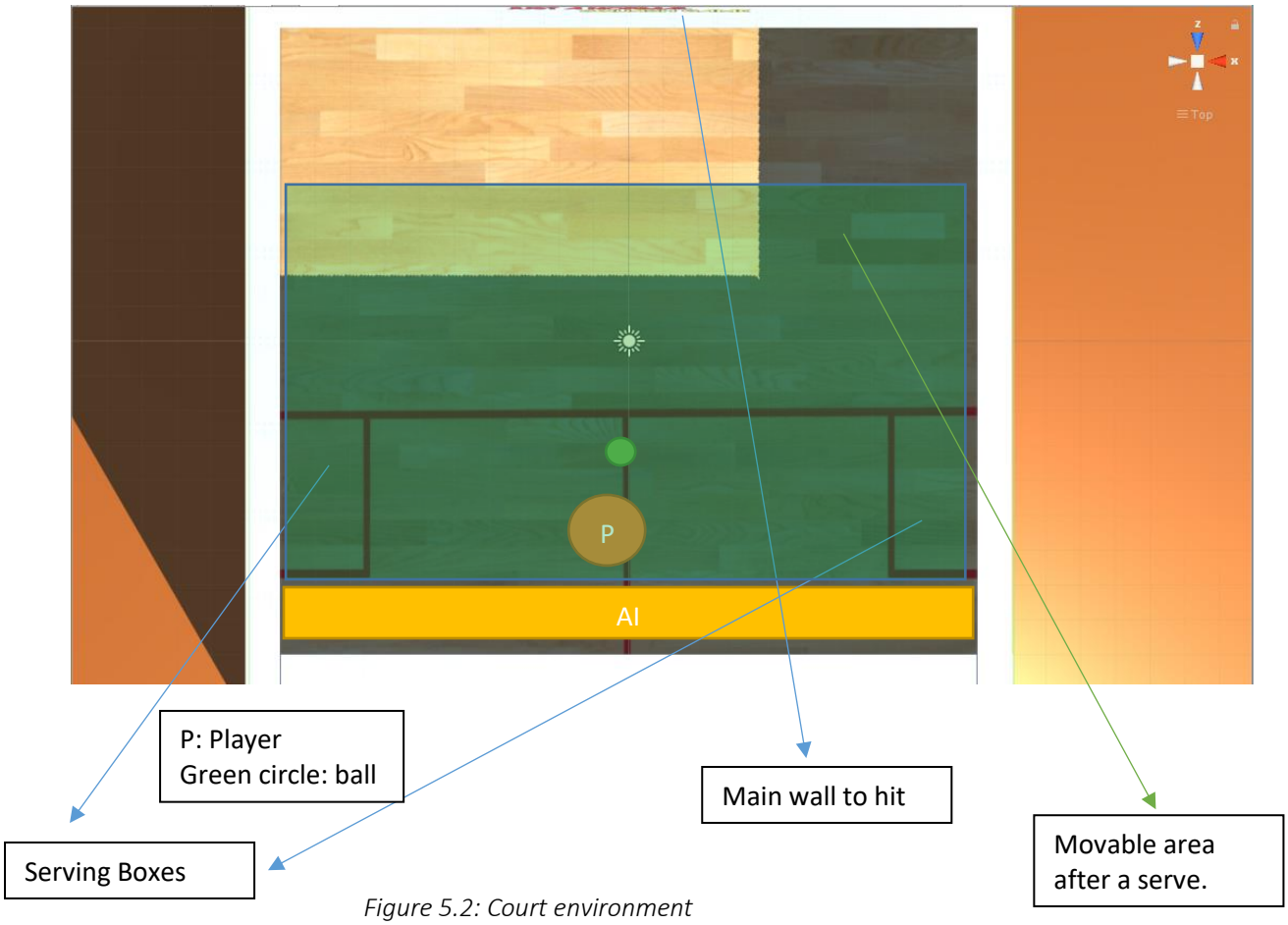


Figure 5.2: Court environment

3.6 Player Input

The player input is split into 2 sections; the Myo armband, which handles the player's arm actions and the android device which handles the player's movements. The Myo manager oversees synchronising the Myo armband to the android device and understanding which poses have been made by the user, to sync between each device the code below is used, this is already written by the Myo SDK.

```
	public static void Initialize()
	{
		#if UNITY_ANDROID && !UNITY_EDITOR
		if(!isInitialized){
			aJavaObject = new AndroidJavaObject ("com.strieg.myoplugin.Myo-
Starter");
			using (AndroidJavaClass ajc = new AndroidJa-
vaClass("com.unity3d.player.UnityPlayer")) {
				using (AndroidJavaObject a_aJavaObject = ajc.GetStatic<An-
droidJavaObject>("currentActivity")) {
					aJavaObject.Call ("launchService", a_aJavaObject);
				}
			}
			if (aJavaObject!= null)
				isInitialized = true;
		}
		#endif
	}
```

The first part is to initialise the Myo armband, this method is called in the start method of the arm controller class which I wrote, MyoManager.Initialize(), once called the Myo SDK will create a new android object and call the Myo plugin. Once initialised the next method can be called which checks for the Myo device, to sync the devices together the Myo armband must be tapped with the android device, usually a button is made and the user must tap the button and at the same time bump the two devices together, however, the game will be in VR therefore it makes more sense to constantly check if the device is tapped against one another by calling the function AttachToAdjacent() in the update function of the Arm Controller class. The code below shows how this is done.

Firstly: call the function in the update method.

```
if(MyoManager.GetIsAttached()){
…
}
else if (!MyoManager.GetIsAttached()){
	MyoManager.AttachToAdjacent();
}
```

Secondly: the AttachToAdjacent function in the Myo SDK

```
public static void AttachToAdjacent(){
	#if UNITY_ANDROID && !UNITY_EDITOR
	if(isInitialized) jo.Call ("attachToAdjacent");
	#endif
}
```

When synchronised the Myo armband will light up blue, and the player will be able to move the character's arms, the movement of the arms are calculated using the rotations of the Myo armbands which are then converted into vectors that can be read by the Unity game engine and applied to the rotations of the arm models. Again, the code below was used to do this.

```
void UpdateRotation()
 {
   UpdateCurrentMyoRotationsWithinRange();
        //new arm x rotation
        newArmHolderRotationLocal.x = Mathf.Lerp(newArmHolderRotationLocal.x, current-
MyoX + currentArmHolderX, Time.deltaTime * armXYSmoothing);
       //new arm y rotation
        newArmHolderRotationLocal.y = Mathf.Lerp(newArmHolderRotationLocal.y, current-
MyoY + currentArmHolderY, Time.deltaTime * armXYSmoothing);

 }
```

The UpdateCurrentMyoRotationsWithinRange function creates a Vector3 variable of the current Myo rotations, the rotations are taken from the Myo SDK using the this piece of code MyoManager.GetQuaternion().eulerAngles;
Once converted, the x axis and y axis of the arm object are updated and the setArmRotation function is called which sets the new rotations of the arms.

```
  void SetArmRotation()
   {
        foreArmBone.transform.localEulerAngles = newArmRotationLocal;
        rightArm.transform.localEulerAngles = newArmHolderRotationLocal;
        leftArm.transform.localEulerAngles = newArmHolderRotationLocal;
   }
```

Next, understanding how the Myo armband reads the gestures of the user and executes an action depending on the pose made. Again, the Myo SDK was used to read the poses, below is a series of case states by which certain actions are carried out depending on the Myo pose detected.

```
switch (myoPose)
       {
           case MyoPose.WAVE_OUT:
               isHitOn = true;
               animeR.SetBool("isSwing2", true);
               StopSwing();
               break;
           case MyoPose.DOUBLE_TAB:
               isSwingon = true;
               animeR.SetBool("isSwing1", true);
               StopHit();
               break;
           case MyoPose.WAVE_IN:
               stopBat();
               break;
           case MyoPose.FIST:
               onUseGrip ();
               isGripOn = true;
               isBatOn = true;
               animeR.SetBool("isBatOn", true);
               animeL.SetBool("isBatOn", true);
               StopHit();
               StopSwing();
               break;
           case MyoPose.REST:
           case MyoPose.UNKNOWN:
               Invoke("StopHit", 2);
               Invoke("StopSwing", 2);
               break;
       }
```

MyoPose is a method within the MyoUnity library, which is used to check against the current pose made. For example, when detected that the user is making a wave out pose, the case will enter the MyoPose.WAVE_OUT state and set the isSwing2 value to true which will then play out the swing animation.
The player object is surrounded in a capsule collider object and the bat is also surrounded by a box collider (as shown in figure 5.3)

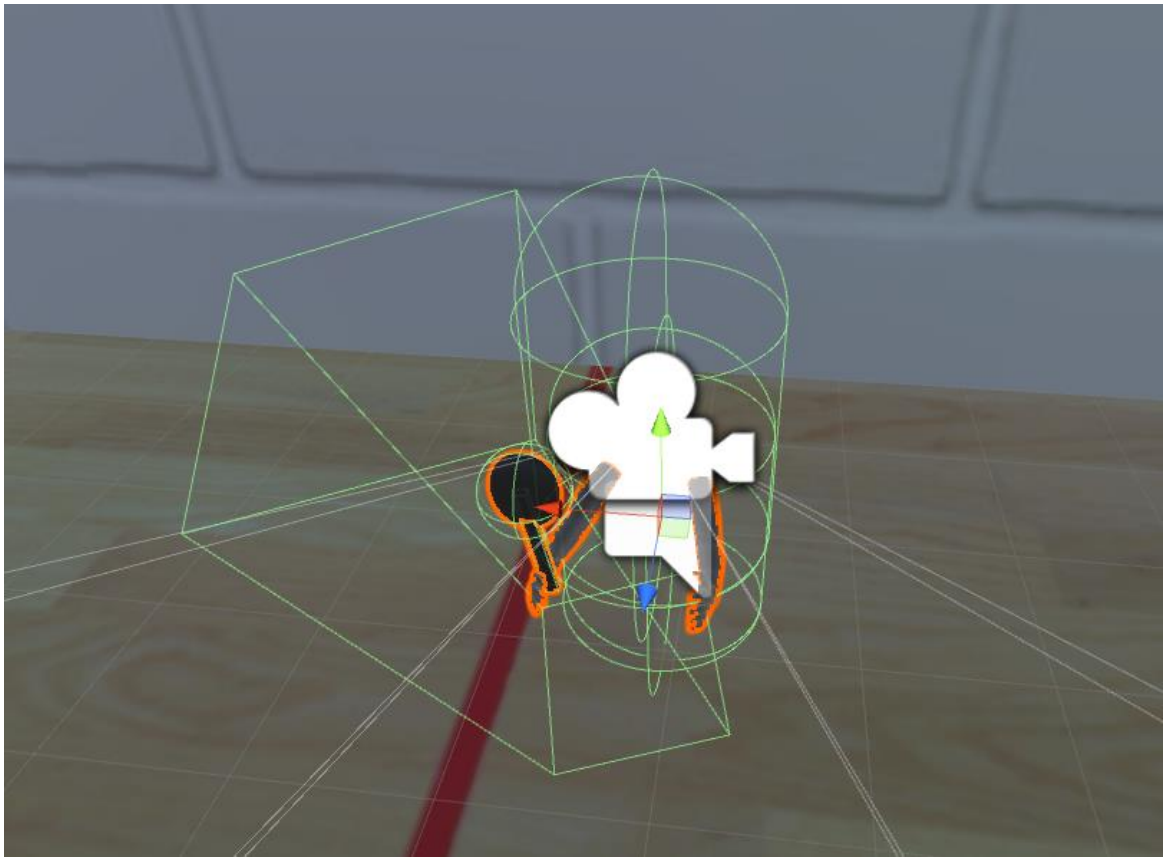


Figure 5.3: Player Game object in a capsule collider and bat in a box collider.

Having the bat in a box collider is important as it means that collision detection can be determined and then calculations can be carried out to understand where the ball's destination should be.

One more ability the player has is being able to hold the ball, this was mainly added for showing off the Myo armbands and as a safe guard in case the ball somehow is at a point beyond reach for the user or the AI, being able to hold the ball means the user can restart that round. The object is grabbed and added to a list of game objects, a list was used for easier future implementations if more game objects were to be added. Once added the grip is set to active and the FixedUpdate function will check if it is active, and if active the ball object will be equal to the left-hand object. The code below (figure 5.4) demonstrates this.

To accurately pick up objects or hit the ball a script was added to the player's main camera object, where a pointer was made, to understand what the player is currently looking at, the pointer's position was then used to determine what the user was currently looking at

```
if (canLookAt)
        {
            palmLoc.LookAt(target);
            newRot.y = palmLoc.localEulerAngles.y + 60;
            palmLoc.localEulerAngles = newRot;
        }
```

Code snippet: getting the palm location for the character.

```
void UseGrip()
    {
        if (!isGripActive)
        {
            RaycastHit hit;

            if (Physics.SphereCast(starthand.transform.position, gripRadius, GetDirec-
tion(), out hit, gripRange, CheckForGrip))
            {

                centre.transform.position = hit.point;

                Collider[] cols = Physics.OverlapSphere(hit.point, 0.6f, Check-
ForGrip);

                foreach (Collider col in cols)
                {
                    GameObject hitObj = col.gameObject;
                    object_hold.Add(hitObj);

                    hitObj.GetComponent<Rigidbody>().useGravity = false;
                }

                if (object_hold.Count != 0)
                {
                    isGripActive = true;
                }
            }
        }
    }
```

Figure 5.4: UseGrip function used to check if user wants to hold the ball.

```
if (isGripActive)
      {
          foreach (GameObject obj in object_hold)
          {

              obj.transform.position = centre.transform.position;
              obj.GetComponent<Rigidbody>().transform.position = starthand.transform.position;
          }
      }
```

This is added to the fixed update method of the Arm Controller class.

Finally, the player's movement can be discussed. This was done using the phone's linear acceleration property. SuperSensor library [SuS1] was a great asset to doing this as it provided all the android device's acceleration properties, gravity values, etc.
Inside the player object the unity FPS script was added and amended to remove the standard keyboard input, instead use the accelerometer as input.
In the start method, an instance of the sensor is started and the type of desired type of acceleration and the sensor delay time is added as the parameters.

```
Sensor.Instance.Start(Sensor.Type.LINEAR_ACCELERATION,Sensor.Sampling.SENSOR_DELAY_GAME);
```

Once initialised, the sensor is ready for use, in the GetInput method several if statements have been added to determine which direction the character should move, the code below shows this.

```
Sensor.Instance.OnLinearAccelerationChanged += (value) =>
{
    yval = value.y;
    zval= value.z;
    if (yval >= 3.15 && canMove)
    {
      Status_Text.text = "Left";
      StartCoroutine(Move(transform.position + Vector3.left));
      canMove = false;
    }
    if (yval <= -3.15 && canMove)
    {
      Status_Text.text = "Right";
      StartCoroutine(Move(transform.position + Vector3.right));
      canMove = false;
    }
    if (zval <= -3.55 && canMove)
    {
      Status_Text.text = "forward";
      StartCoroutine(Move(transform.position + Vector3.back));
      canMove = false;
    }
    if (zval >= 3.55 && canMove)
    {
      Status_Text.text = "backwards";
      StartCoroutine(Move(transform.position + Camera.main.transform.forward));
      canMove = false;
     }
 };
```

Though this method for movement seems like a bad idea for this type of game where fast and responsive input is required, the main point of using such a method was to demonstrate the capabilities of having 0 physical controllers where direct input is required (keyboard, mouse, gamepads, etc.) and only using muscle recognition hardware and user's current position in the real world. If there was more time for this project the use of both GPS and accelerometers would have been used to provide for a precise accurate representation of the user's movement intentions. A similar application that comes to mind is the Pokémon GO app [POK1] which uses this method for movement to track the player's location using the longitude and latitude and converting them to positions in the game world, it also uses accelerometer readings to provide even more accuracy.

3.7 VR Implementation

Using the GoogleVR SDK, it was fairly simple to implement the VR component for the game, the difficulty that was faced was with the compatibility issue, due to the GoogleVR needing version 24/19 for the android device (depending on which platform was to be used, Cardboard or Daydream), the Myo SDK required version 18, so when trying to work with the Myo armband while using version 24, there were issues that did not allow for the Myo armbands to synchronise with the device, The solution will be discussed in a later chapter.
Once the GoogleVR SDK was imported into Unity the GvrViewerMain object needed to be dragged into the player object, figure 5.4 shows the settings used for the GoogleVR to work correctly. Everything else after that was straightforward, just run the application and the VR was enabled.

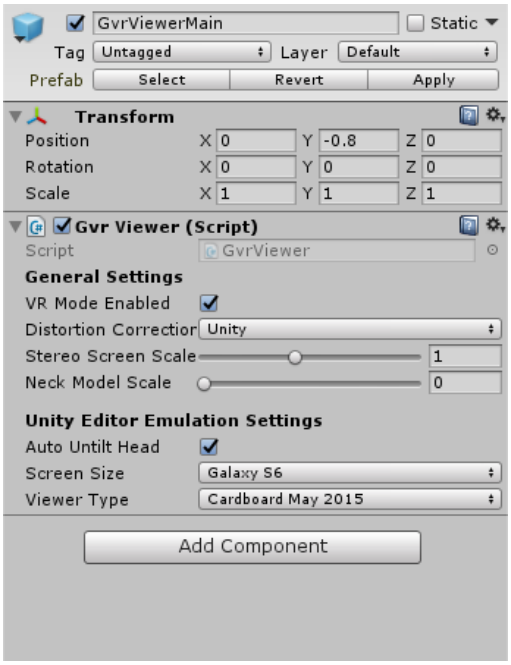


Figure 5.4: VR settings.

Chapter 4. Testing

4.1 Introduction

This chapter will detail the approaches taken to carry out the testing of the application. Because of the nature of the application the testing methods available are limited, black box testing with the use of test cases is a good approach. Series of tests will be carried out; the tests will mainly be compared against the system specifications described in chapter 4.2. Other objectives include the general gameplay of the application, the accuracy of the input method used and how the AI deals with certain situations.

4.2 System Specifications

To have a successful project, the system needs to be able to meet these following requirements (functional and non-functional):

Functional Requirements

1. Player Movement: The user will be able to move the player using the phone's accelerometer, i.e. the player should move based on the user's movement.
2. Myo controllers: The user will be able to control the player's arms based on their own arm movements, e.g. to swing a racket the user will need to swing the arms forward.
3. Enemy AI: The enemy AI will be functional and have at least a basic gameplay knowledge.
4. Game Objects: The game object physics should work and resemble closely how they do in the real world.
5. Narrator: The system narrator should function correctly and provide useful information to the user as well as interacting well with them.
6. Accuracy: The game will need to be accurate in the following criteria:
    - Follow the user's arm movement.
    - Ball location should be accurate based on where the user hit it.
    - Player movement should highly represent the user's actual movement i.e. should player should not move back when user moves forward.
    - AI have accurate shots and play fairly.

Non-Functional Requirements

1. Accessibility: The game should be very easy to understand and use, and the narrator should do a good job at helping the user to understand how the game works.
2. Quality: It is important to have a good quality and eye catching design for a game, as it is a big factor to influence whether the user ends up enjoying the game, therefore there should not be any rendering issues nor lags or delay within the game, especially a VR game, having delays could cause headaches which can ruin the user's experience. Other than render quality the game environment should be pretty and wide to all the user to feel immersed within a different world.
3. Portability: One of the major advantages of this project is that it is highly portable as it uses a mobile phone as the main platform for the game, and a VR headset as well as the Myo armbands, although it is currently impractical to think of commercial use for such a project, once the Myo armbands are more well-known this should not be a problem. The project should not use any other source of input as it will ruin the portability benefit which it provides.
4. Resource devices: The system makes use of both a VR headset and Myo armbands, it is important to make sure that these devices are calibrated well for most users, to provide high accuracy.

4.4 Testing Myo Accuracy

Testing for the Myo device was done to see how accurate the hardware was, though some tests were carried out by myself (carried out 266 tests), there was also a test that had already been carried out by another individual (1876 tests) which proved as a great asset to compare alongside the tests that were conducted by myself. See Appendix A for the results, there will be data sheets as well included in the project dissertation folder. Overall the results from the Myo tests carried out came as expected (see evaluation for testing results explanation).

4.5 Testing Accelerometer accuracy

Performing tests on an android device accelerometer is not ideal, as there are many android devices each differ in its accelerometer hardware which will lead to ambiguous and inaccurate testing results.

4.6 Test Cases

The main form of testing involved in creating a set of test cases, which firstly checked the application against its requirements and secondly checked against self-created scenarios, pretending to play as the player is one way of testing, however also asking few people to play the game gave a more of an accurate reading (see Appendix A).

Chapter 5. Evaluation

5.1 Introduction

This chapter will evaluate the whole project, by going over the testing results, the overall performance of the application and the difficulties faced. Decisions taken will also be explained.

5.2 Testing Results

Overall the system was tested using the test cases. I excepted to have numerous of fail results, however though there were some fail cases, overall the game played well and the main objective of the application was met, this was to create a VR game by which the user does not need to use a keyboard, mouse or a gamepad to play the game, rather use their body to perform certain actions (such as moving forward or swinging the racket). Though I was disappointed that the game is unable to finish correctly as the AI would always win.
What surprised me the most was the accuracy of the Myo armband, achieving an average accuracy rate of 90.5% (combing both my result and the results taken from allingeek) [AIG1], for a muscle recognition software this was a great result to achieve, which shows the potential that this hardware has.

Though no tests were performed on the accuracy of the android device accelerometer, I could notice from my own device that most of the time the device did perform accurately, however when the device is worn by the user using the VR headset the accuracy drastically decreased and it also became highly unpractical.

While many test cases were done, a better solution would be to use unit testing, however Unity does not come with such a library so I was unable to do this, and creating my own program that does this for me would have been time consuming and difficult.

5.3 Performance

The application performed well overall. The android device which this application was tested on was the Google pixel XL, which has high specifications, 1.6GHz quad-core processor, 4GB RAM and a 32GB internal storage, running android version 7.1 (to date). Of course, running this game on such a device has little to no effect on the performance, however the same cannot be said for devices with lower specifications following GoogleVR's minimum specification requirement, the following is needed:

1. Must have at least 2 physical cores
2. Must support sustained performance mode
3. Must support Vulkan hardware level 0 and 1
4. Must support HEVC and VP9 and be compatible to decode at least 1920 x 1080 at 30fps

Using the GoogleVR is a good frame of reference to understand what specification is required to run this game.
The game makes use of very few game objects therefore the application should not cause any performance issues at even lower spec devices.

5.4 Difficulties

5.4.1 Myo armbands

As explained previously I was unable to attach 2 Myo armbands to an android device, the issue was due to the Myo SDK currently not supporting such a system, and attempting to do it by myself lead to a lot of issues, causing Bluetooth interference between the 2 Myo armbands and the android device, so in the end this was not able to be resolved. Future Myo SDK development has been proposed to fix this issue, and so splitting the player arm models (into right and left arms) has been done to easily implement the controls for the left arm (when the Myo SDK is updated).

5.4.2 VR compatibility issue

As briefly mentioned before, the GoogleVR requires the android software version 19 or 24 (depending on whether using Cardboard or Daydream), when I changed the version of the android device in the build settings, I was unable to synchronise the Myo armbands to the android device, this caused a lot of compatibility problems, which took a while to understand and overcome, but essentially when using the GoogleVR SDK, 2 Android manifests are added to the plugin folder of the project directory, and each have a different Android version added to them, 19 for Cardboard and 24 for Daydream and the main manifest contains the original version 18 which was done by me, to fix this issue I needed to specify the minimum android level to be 24 in the manifest and the target needed to be set to version 22, after which the application ran smoothly.

5.4.3 Linear Accelerometer

The linear accelerometer problem has been mentioned several times already, where the main issue here is the accuracy of the accelerometer when the user is wearing the VR headset and trying to move the character in difference directions, initially when I proposed the idea of making such a method for moving a player, I did not consider that the device's accelerometer properties to be so limited. The fact that this inaccuracy occurs makes the gameplay very unpleasant for the user, I tried to minimise this issue by playing around with the z and y axis input values, lowering them allowed for easier read, meaning that less force is needed to move the player in certain directions. However overall the best approach would have been to use a hybrid system using both GPS and accelerometer readings.

Chapter 6. Conclusion

6.1 Summary

The initial goal of this project was to create a simple game that demonstrates the ability of muscle recognition hardware combined with VR, as a proof of concept. Overall this has been a successful project I have been able to show how using both the input from the accelerometer and Myo armbands, it is possible to create an immersive game. Though the initial idea for my project was to create an FPS game however having listened to Peter Blanchfield, we concluded that a squash game was more effective to demonstrate the power of the Myo armband technology.

Ultimately, I have enjoyed working with these different tools and developing such a game. A great lesson learnt from working on such a large project was understanding my ability and knowing when to ask for help when stuck, I prided myself to much in doing everything by myself, but I realise that in the real world where many programmers work together to solve issues at hand, individual skill is not such a big asset to have as I initially had thought. Nevertheless, I am proud of the work I have done, the research done before hand on this technology was very enjoyable, seeing different ways developers use this technology to create new ways of doing things.
Working on the combinations of these tools gave me hope that these tools will be used together in the future, perhaps we will see more games like Pokémon Go where the Myo armbands and VR is used. My biggest regret is not using a GPS based approach for the movement of the player, the combination of using an accelerometer and GPS reading will give for very accurate results.

6.2 Potential Expansions

As with most games, they are never truly finished. Especially with this project, there are a lot of expansions that can be added, such as a cleverer AI agent, using a hybrid source of input for the movement (GPS and accelerometer), and implementing the narrator design which I never got a chance to do because of external pressure from outside of my academia. The actual graphical of the game could also do with some polishing, there are definitely more implementations which I would like to add to this project, as it was very enjoyable for me to work with.

Chapter 7. Bibliography

[QU3] Quake arena bot AI. Retrieved from quake wiki: http://quake.wikia.com/wiki/Quake_3_Bots

[Rul1] *Rules for squash*. (2016). Retrieved from England Squash: https://www.englandsquash.com/get-involved/play/rules-of-squash

[Acc1] Sam Naghshineh, G. A.-E. (n.d.). *Human Motion capture using Tri-Axial accelerometers.* Retrieved from http://edge.rit.edu/edge/P10010/public/PDF/HME.pdf

[Myo1] Myo Armbands (2016, 10 19). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Myo_armband

[Acc2] Graham, B. B. (2000, 05 11). Measure Human Hand Motion. Retrieved from mtl: http://www-mtl.mit.edu/researchgroups/MEngTP/Graham_Thesis.pdf

[Hv1] Vive. (n.d.). *Vive user guide*. Retrieved from HTC.com: http://www.htc.com/managed-assets/shared/desktop/vive/Vive_PRE_User_Guide.pdf

[Kla1] Miseckas, K. (2016). *Arm model* . Retrieved from bitbucket: https://bitbucket.org/AngryBurritoCoder/myo-armband-star-wars-inspired-force-powers/src/4630101b9d58c8c727d4a894b98a56a1f30cf87b/Myo_Powers/?at=master

[Micro] Wikipedia. (n.d.). Micropsia . Retrieved from https://en.wikipedia.org/wiki/Micropsia

[POK1] Pokémon GO. Retrieved from Pokémon Go website www.pokemongo.com

[AIG1] Allingeek, Myo data sheet, using MIT license. Retrieved from https://github.com/allingeek/myo-dataset

[GVR0] Google VR. Retrieved from Google VR: https://vr.google.com

[MMED1] Mahmoud Abduo and Matthias Galster. Myo Gesture Control Armband for Medical Applications. Retrieved from https://www.cosc.canterbury.ac.nz/research/reports/HonsReps/2015/hons_1502.pdf

[ACC0] Ryan Z. Amick, Jermey A. Patterson, Michael J. Jorgensen. Sensitivity of Tri-Axial Accelerometers within Mobile Consumer Electronic Devices: A Pilot Study. Retrieved from http://www.ijastnet.com/journals/Vol_3_No_2_February_2013/14.pdf

[ACC1] Adam J. Bittel, Ashraf Elazzazi, Daniel C. Bittel. Accuracy and precision of an accelerometer-based smartphone app designed to monitor and record angular movement over time. Retrieved from http://digitalcommons.wustl.edu/cgi/viewcontent.cgi?article=6000&context=open_access_pubs

[Mod1] Free models for Blender. Retrieved from TurboSquid https://www.turbosquid.com/Search/3D-Models/free/blend

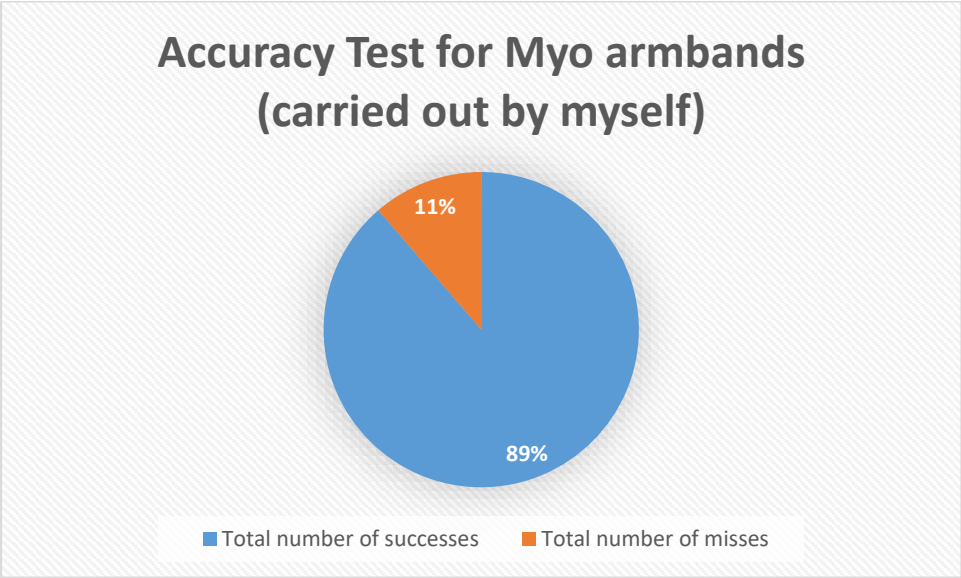
[ARM1] Arm model made by eafg however amended by me, License used (http://creativecommons.org/licenses/by/3.0/) Retrieved from http://www.blendswap.com/blends/view/42830¨

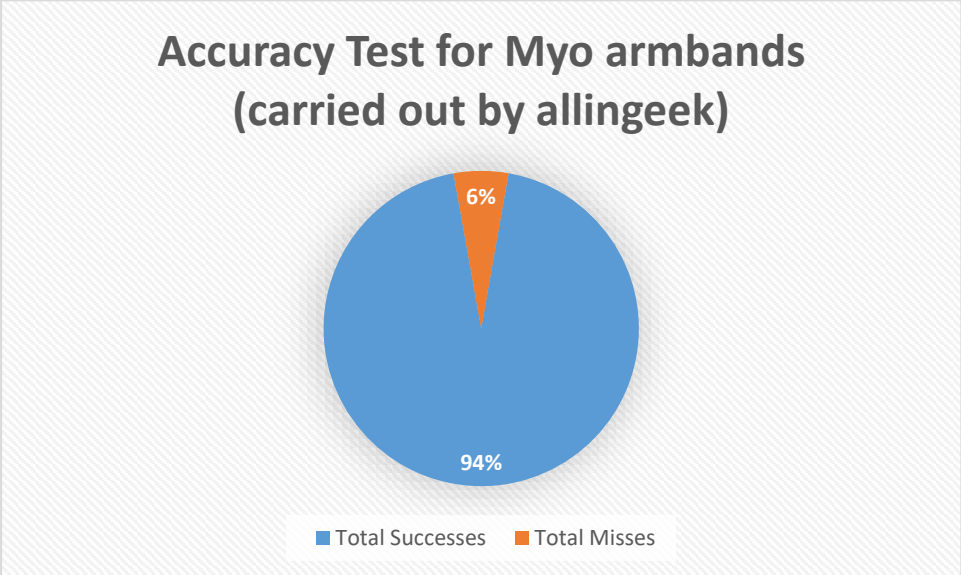
[SUP1] SuperSensor Library Asset. Retrieved from https://docs.google.com/document/d/1c6y_1sqnR0q_zdzQQRTAgv5jvtuf4PCzQg7LWiiBuzo/edit
&
https://www.assetstore.unity3d.com/#!/content/58729

[THA2] Myo SDK by Thalmic Labs. Retrieved from https://developer.thalmic.com/docs/api_reference/android/index.html

Chapter 8. Appendices

Appendix A: Testing





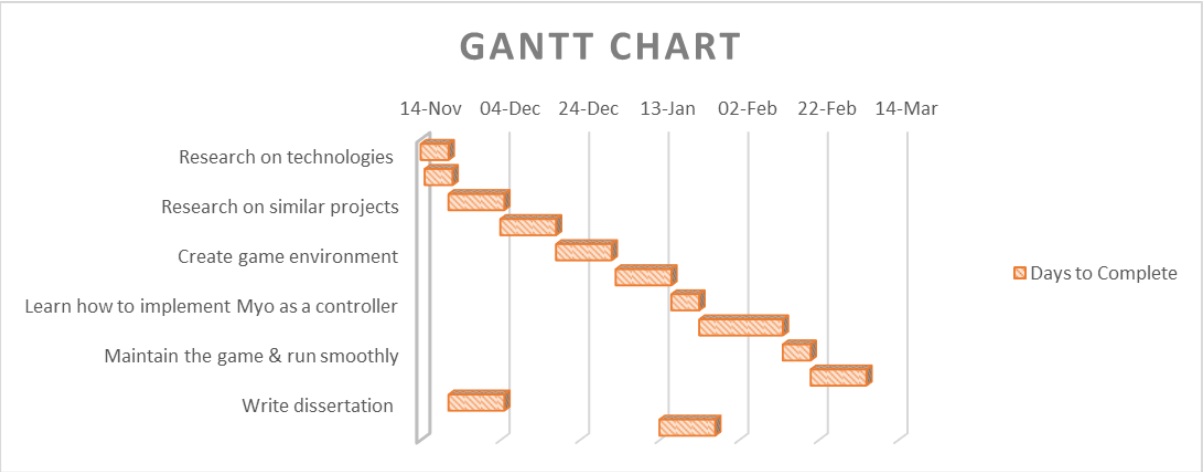
Test Case No.	Description	Case Steps	Expected Outcome	Pass/Fail	Actual Outcome	Solution (if failed)
1	Synchronise Myo Armband to device	1) Launch game on android device 2) Tap Myo Armband against Device	Myo armband to light up blue and character moving arms based on the rotation of the Myo armband	Pass	As expected	
2	Check if orientation of the character arm models is the same as the user's (using Myo armband)	1) carry out the steps in test case 1 2) Move arm and check if the orientation of the character arm model is the same as the user's arms	Character arm to follow the user's arms based on the Myo device rotations.	Pass	As expected, however if Myo armband is low on battery the orientation is not entirely accurate	Make sure Myo armband is always charged before playing the game. Also, inform the user of the battery percentage for the Myo device
3	Carry out fist pose to see if ball object gets grabbed by the character.	1) carry out the steps in test case 1 2) Perform fist pose gesture 3) make sure within range of ball	Ball object to move inside the left hand of the character arm model. Fist animation to be played and character remains in that animation.	Pass	As expected	
4	Carry out finger spread pose to see if the grip released and the ball falls out of the character's hand.	1) carry out the steps in test case 1 2) perform finger spread pose gesture	Ball object to fall out of the character's hand and bounce to the wall. Character's hands should carry out the spread finger animation and remain in that animation.	Pass	As expected	

5	Carry out wave in pose to see if the swing action is made (from the character model)	1) carry out the steps in test case 1 2) perform wave in pose gesture	Character right arm model should perform the swing 1 action (playing swing 1 animation).	Pass	As expected however, the accuracy on the wave in pose is lower than expected	Re-calibrate the Myo armband.
6	Carry out the wave out pose to see if the swing 2 action Is made (from the character model)	1) carry out the steps in test case 1 2) perform wave out pose gesture	Character right arm model should perform the swing 2 action (playing swing 2 animation)	Pass	As expected	
7	Move character forward	Push the android mobile device forward in the z-axis	Character model should move forward by 1 unit (1 unit = 0.8 move speed)	Pass	As expected	
8	Move character backwards	Push the android mobile device back in the z-axis (-ve value)	Character model should move backward by 1 unit	Pass	As expected	
9	Move character left	Push the android mobile device to the left direction x-axis	Character model should move to the left by 1 unit	Pass	As expected	
10	Move character right	Push the android mobile device to the right direction x-axis (-ve value)	Character model should move to the right by 1 unit	Pass	As expected	
11	Keep character at an idle position (no movement)	Perform no action	Character model should not be moving at all	Fail	Character sometimes moves to the left location	Increase the y-axis value (+ve) to prevent random movement occurring.

12	AI opponent to serve the ball diagonally	Start the game	AI agent should serve the ball diagonally (near player side)	Fail	AI hits the ball at any direction	Add if statement to check if it's first turn and then serve the ball at x-axis close to the player's location.
13	AI opponent to continue returning the ball to the user	Start game and play past first serve and user should return the ball to the AI	AI agent should receive the ball from the user and return it by hitting it against the wall	Pass	As expected	
14	AI opponent to lose some points by not returning the ball	Carry out test case 13	AI agent should not always win and miss some ball shots	Fail	AI always returns the ball to the player	Reduce the AI's box collider and move at a slower pace to where the ball is expected to land
15	The user to return the ball	Start game and play past first serve	The player should be able to return the ball when bat detects that it is touching the ball	Pass	As expected	
16	The user to serve ball when AI loses point	Start game and wait till AI loses a point (by not being able to return the ball to the user)	The user should serve next	Fail	Because of the problem in Test case 14 the user never had the chance to serve	Fix step 14
17	The game physics to work correctly when user performs swing action	1) Start game and play 2) perform swing action	The ball should have more force when the ball is hit against the bat which performed swing action	Pass	As expected	
18	The AI should be able to return the ball at different force	1) Start game and play	Once AI performs swing action the ball should be	Fail	The ball is always returning at the same force	No solutions found, possibly that swing action

		2) wait until the AI performs swing action	returned to the user at a higher velocity			is not read properly due to it being used also by the user
19	Count score	1) Play game and win a point	If the AI returns a ball to the user and the user unable to return the ball, then AI should gain 1 point on the score table	Pass	As expected however, when players turn unable to see this work as the AI did not lose any points	Fix step 14
20	UI interface should work correctly in VR	1) connect device to a VR platform, in this case the Google Daydream was used 2) launch game and play	All the UI interface should be shown in VR mode	Fail	UI interface was not shown (i.e. the score UI)	GoogleVR Bug (expected to be fixed by next patch update)

Appendix B: Gantt Chart



Appendix C: User Manual

Requirements

- An android device – currently only supported for android devices
- An Myo armband
- A VR headset, preferably Daydream, however, Cardboard will work too.

How to play

Running the game: Unzip the folder and install the .apk file (this can be done easily using your android device), other ways to install the apk file to your phone is by uploading it on a file sharing site.

While installing, you should put on your Myo armband, this will go on either arm but it must be placed on the forearm just before your elbow (as shown in figure 6.0).

Figure 6.0: How to wear the Myo armband. Image taken from: https://support.getmyo.com/hc/en-us/articles/201169525-How-to-wear-the-Myo-armband

It is recommended that you allow the Myo armband to be warn for at least 5 minutes for it to warm up and adjust to your muscles.

Once installation is complete, run the app, wait until the main menu is shown, after which tap your Myo armband against your android device (repeat this until a blue light appears on your Myo device). This means your Myo device is synchronised and now you are ready to play!
You should notice that the arm models inside the games are moving based on your arm rotations.

Controls:

Wave in: Swing action 1
Wave out: Swing action 2
Fist: Grab ball
Finger spread: Release ball.
Double-tap: Start Game.

To move forward simply push your head slightly forward, the input is the phone's linear accelerometer.
To move backwards simply push your head backwards.
To move left just push your head towards the left side.
To move right just push your head towards the right side.

Gameplay:
Simply hit the ball back every time it turns cyan colour (this indicates that it is your turn) when the ball turns red it is the AI's turn.

Enjoy!