ROOTIN' TOOTIN' COWBOY SHOOTIN'

Purpose of the application

When assigned this project, we thought about creating something in Unity [1] as we had a little experience in it from the previous semester. With this thought, we figured the MYO [2] armband would work best for us as it had its own unity assets that would get us off the ground. With this decided upon we now had to come up with an idea on what we were going to develop. We are fans of gaming so we thought a game would be fun to develop with the use of the MYO armband, and Unity being a game development environment. We had many ideas, which included a golf game where the user would swing their arms to hit a ball into a hole/area. Another idea was a Monkey Ball style [3] game where the user would use the gyroscope on the MYO to control a platform where a ball would roam around and try reach the end goal. We agreed upon a first-person perspective shooting type game as we're both fans of the genre and that the MYO might emulate the feel of a shooting type game.

Gestures identified as appropriate for the application

The application we developed is a First-Person Shooter game where the user would use the MYO armband to target bottles in a shooting range type scene. We thought the MYO's gyroscope would best suit the aiming element of the shooter as it seemed to make the most sense for a project like this one. The user moves their arms and the crosshairs on the screen move with it. We contemplated with what gesture to use for the shooting itself, narrowed down to two options. The "Open Spread" gesture and the "Fist" gesture. Both viable options as the both replicate a shooting motion in their own way. The "Open Spread" gesture almost emulates a release or a throwing motion that could be used within this game, but in the end, we opted for the "Fist" gesture. We felt it suited better as it almost replicated a *trigger pulling* motion.

These gestures are the most important for the game as it covers all bases for the core gameplay, the aiming and the shooting. With these decided upon we had to figure out how we wanted to go forward with the other gestures. During development of these gestures we noticed that mid-gameplay that the gyroscope would read some incorrect values, which would result in such bugs as the camera constantly turning to one direction. This was too big of an issue not to ignore so we decided to implement a gesture that would recalibrate these values mid-game if the issue were to ever arise. For this we decided to use the “Double Tap” gesture as it was the most unique gesture available to use. We wanted a unique gesture as it was a bug that would only occur occasionally.

Since we had some more gestures available to use, we decided to use these gestures for menu navigation throughout the application. With the use of scene transitions, we were able to access a new pool of gestures in each scene, allowing us to reuse previous gestures such as the “Fist” from the gameplay scenes. This was needed as there was a bug during scene transition that would register a gesture from the previous scene and execute it in the next. For example, we had a “Fist” gesture to exit the Controls scene from the pause menu, but then the user performed this gesture it would carry into the pause menu and execute the gesture on that scene, which was to exit the game. So it would take the user from the Controls menu, to the game, and then exit the game and navigate to the main menu. To prevent this, we had to use unique gestures in each scene so that they would carry over from the previous scene.

Scenes / Gestures

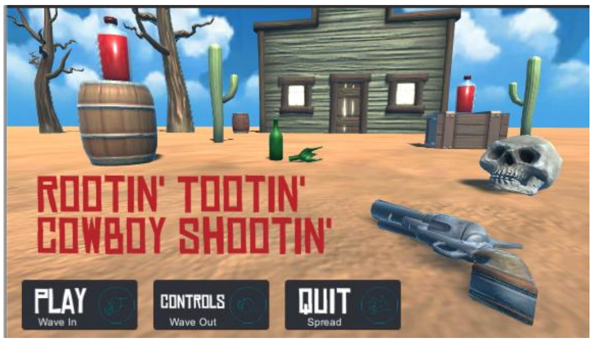
$$\begin{tikzcd}
 & \boxed{\text{Controls}} & \boxed{\text{Level 1}} \arrow[d] \\
\boxed{\text{Splash Screen}} \arrow[r] & \boxed{\text{Main Menu}} \arrow[u] \arrow[ur] & \boxed{\text{Level 2}} \arrow[d] \\
 & \boxed{\text{Game Complete}} \arrow[u] & \boxed{\text{Level 3}} \arrow[l]
\end{tikzcd}$$

Main Menu

The Main Menu consists of 3 buttons, each controlled by a unique gesture to that scene.
The Play button is triggered by the "Wave In" gesture. This navigates the user to the first level of the game.

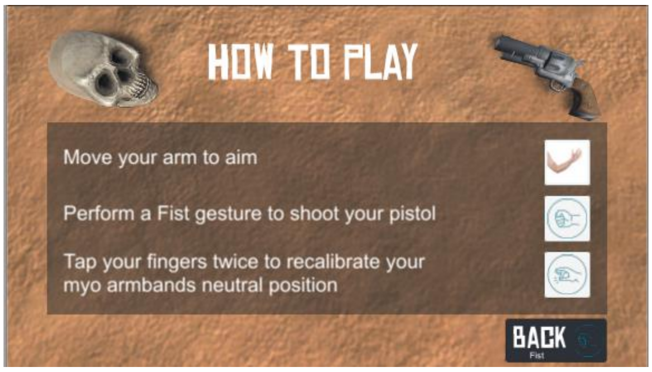
The Controls button is triggered by the "Wave Out" gesture. This navigates the user to the Controls scene.

The Quit button is triggered by the "Spread Hand" gesture. This exits the application.



Controls

The controls menu displays the game controls and each gesture used in the game scene to use these controls. It also contains one gesture-controlled button to navigate away from the scene.
The Back button is triggered by the "Wave Out" gesture. This exits the controls scene and navigates to either the main menu or level pause, depending on how the user navigated to the options menu.


HOW TO PLAY

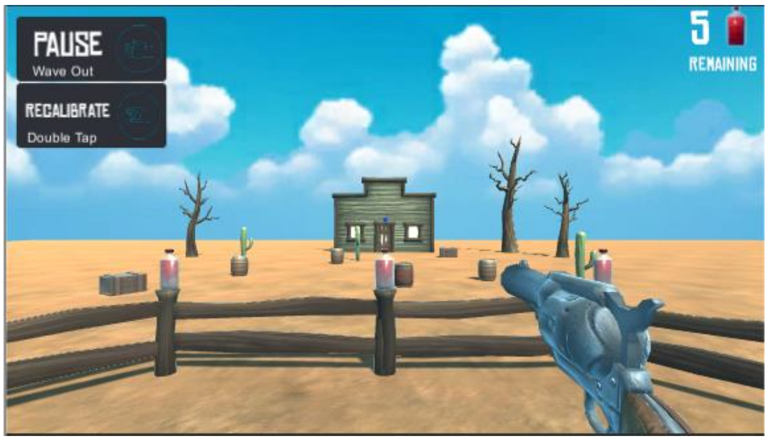
Move your arm to aim

Perform a Fist gesture to shoot your pistol

Tap your fingers twice to recalibrate your myo armbands neutral position

Levels 1-3

The level scenes are where the game is played. Camera is in first person perspective and is controlled by the gyroscope on the MYO armband.
To shoot the gun the user must use the "Fist" gesture. This sends out a Raycast to whatever object the crosshairs are pointing at. The objective is to shoot the bottles so when you shoot the bottles, the Raycast is sent out and they are destroyed.

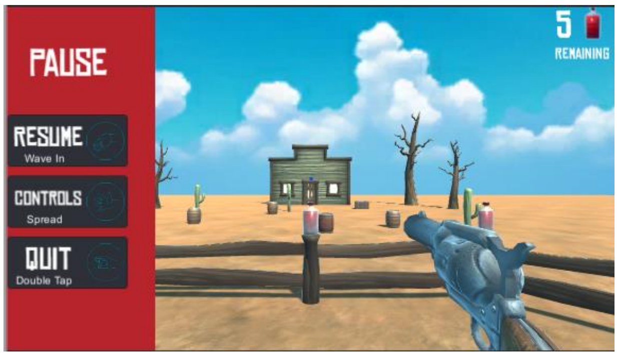


To Pause the game, the user uses the "Wave Out" gesture. This pauses the current scene and opens more options for the user to navigate to.

Each level has a set number of bottles. Three in the first, Four in the second and Five in the third. Once all the bottles in the scene are destroyed, the user is navigated to the next. Once the final level is complete, the user is brought to the win scene and the game is completed.

Pause Menu

The Pause menu itself is not a new scene, but an overlay in each level. When the pause menu is triggered the gameplay is paused and a new pool of gestures are available.
To resume the game, the user uses the "Wave In" gesture. This hides the pause menu and resumes the game from where it left off.



The Controls button is triggered by the "Spread Hand" gesture. This navigates the user to the control’s menu.

The Quit button is triggered by the "Fist" gesture. This navigates the user back to the main menu.

Hardware used in creating the application

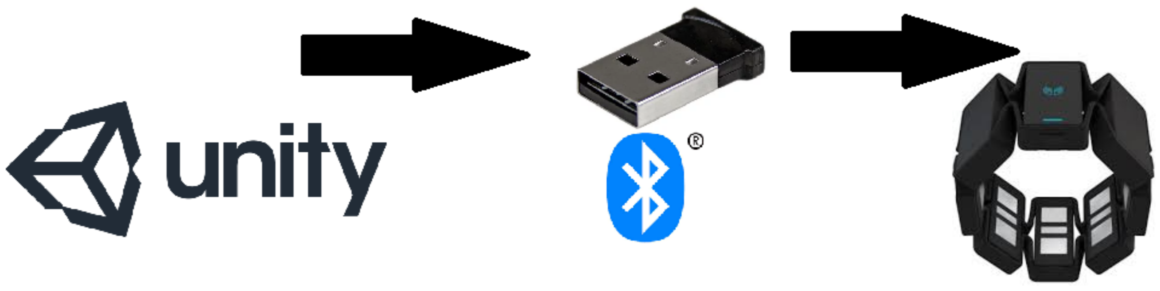
Since we decided on making a game in unity, it limited the options on hardware available to use. For example, the likes of the Raspberry Pi, the Arduino and the Lego Mindstorm wouldn't be needed with what we were doing. We had decided on the MYO as it's the first one that caught our eye in the labs, but once we had decided on the type of application we were developing, we looked into the other devices available.

The leap motion [4] controllers focus on hand tracking and movements, which could have been used in our shooting range, but we decided against it as it focused on detailed hand movement such as bending fingers and wrist motion. It would be difficult to emulate this in a gaming environment unless we were to use an oculus rift but even then, it would be difficult to aim with a weapon in that environment in comparison to the MYO armband.

We also looked into using the Kinect for this project, which uses a camera to view the user and enables users to control and interact with their console/computer without the need for a controller, through a natural user interface using gestures and spoken commands. The Kinect's main trait is that it focuses on full body input from the user, which wasn't really needed in what we were developing. It also takes up a lot of space physically for the user to get accurate results in tracking. After looking at these issues we decided we didn't need what the Kinect offered when directly compared to the MYO.

Deciding on the MYO was easy in retrospect, the gestures it provided was exactly what we needed. The fact that the user can use the MYO from almost any position, whether they're sitting in front of the computer screen or standing in front of a television.

Architecture for the solution



ThalmicLab provides two C# scripts used to connect the myo armband to a unity project [6]. Data from the input can then be taken in and manipulated as required.

In PoseCheck.cs, we import relevant libraries from Thalmic.Myo. It is not recommended to import the entire Thalmic.Myo library as it contains some types (such as Vector3) which are also used in Unity and would lead to errors and overall confusion.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using LockingPolicy = Thalmic.Myo.LockingPolicy;
using Pose = Thalmic.Myo.Pose;
using UnlockType = Thalmic.Myo.UnlockType;
using VibrationType = Thalmic.Myo.VibrationType;
using UnityEngine.SceneManagement;
```

The LockingPolicy is used in order for the instance of the ThalmicHub to be a singleton, thus meaning only one instance of the hub can exist at any one time.

Pose is used to determine which type of pose is currently being detected from the armband, with the Rest Pose being the armbands neutral state.

UnlockType and VibrationType are used to connect to the armband and be sure the armband is synced correctly. From the Armband Manager software, a Ping can be sent to the armband to vibrate it and show the armband is connected. We have not used the vibration in the game.

On each scene, a different set of actions are mapped to each pose input, meaning that performing a wave in gesture will cause a different action on the game scene than the main menu scene.

```
switch(myo.pose)
        {
            case Pose.FingersSpread:
                if(PlayerPrefs.GetInt("isPauseLoaded") == 1)
                    Options();
                break;
            case Pose.Fist:
                Shoot();
                break;
            case Pose.WaveIn:
                if(PlayerPrefs.GetInt("isPauseLoaded") == 1)
                    Resume();
                break;
            case Pose.WaveOut:
                if(PlayerPrefs.GetInt("isPauseLoaded") == 0)
                    Pause();
                break;
            case Pose.DoubleTap:
                HideRecalibrateScreen();
                if (PlayerPrefs.GetInt("isPauseLoaded") == 1){
                    Quit();
                }
                break;
            case Pose.Rest:

                break;
            case Pose.Unknown:

                break;
        }
```

This code extract is from the main game scene, and thus a PlayerPref int must be checked to see if the pause menu is currently active. If it is then DoubleTap will quit the game, rather than simply re-calibrating the armbands neutral input like it normally does here in CameraLook.cs

```
void Calibrate()
    {
        //Get myo XY axis angles at calibration time
        myoXAtCalibration = myo.transform.localRotation.eulerAngles.x;
        myoYAtCalibration = myo.transform.localRotation.eulerAngles.y;

        //Set dead zone centers
        deadZoneXCenter = myoXAtCalibration;
        deadZoneYCenter = myoYAtCalibration;

        if(deadZoneXCenter > 270 && deadZoneXCenter <= 360)
        {
            deadZoneXCenter -= 360;
        }
    }
```

Conclusions & Recommendations

Throughout this project we have had the opportunity to research multiple different gesture input devices and techniques. This has led us to viewing the way we interact with software we design in a completely new sense. The use of physical gestures was never a thought for either of us before being presented with this project but incorporating such input techniques has been an interesting change from more traditional mouse and keyboard or even touch controls. This field has really progressed rapidly in the past few years, with Virtual Reality being the next big step in gesture-based IO experiences, and learning how to incorporate these new ideas into a project has been challenging and rewarding, with both of us believing the future of gaming and computing could be headed towards a more gesture-based future, as long as hardware continues to evolve and become more and more accurate and consistent.

As for recommendations, having hardware on hand to properly develop from the beginning would have been a great help. Also, although the MYO armband was fun and very interesting to develop with, it's problems with accuracy and misread inputs can sometimes be very frustrating, if we were to do the project again, we might try and find different hardware to try and develop with. Saying that, from what we researched, the MYO armband was still the best option for us at the time.

Overall the project was an enjoyable experience, especially playing with the hardware and testing the different features and gestures available to us. We have gained a new understanding of the thought that can go into user input and how important a user's perceived idea about how a gesture should work actually is.

Link to GitHub - https://github.com/BrianD147/Gesture-Based-UI-Development-Project

References:

1. Unity - https://unity3d.com/learn
2. Myo – https://support.getmyo.com/hc/en-us/articles/203398347-Getting-started-with-your-Myo-armband
3. Super Monkey Ball - https://en.wikipedia.org/wiki/Super_Monkey_Ball_(video_game)
4. Leap Motion - https://www.leapmotion.com/
5. Kinect - https://en.wikipedia.org/wiki/Kinect
6. Thalmic Labs C# code - https://github.com/thalmiclabs/myo-unity/tree/master/project/Assets/Myo/Scripts