Gesture Based UI Development: Myo Armband + Unity3D

Neil Kyne

Contents

[GitHub Link - https://github.com/NeilK-94/Gesture-Based-Development-Project 2](#_Toc36312108)

[Purpose of the Application 2](#_Toc36312109)

[Gestures Identified as Appropriate for this Application 3](#_Toc36312110)

[**Myo Armband** 3](#_Toc36312111)

[**Microsoft Speech Engine** 3](#_Toc36312112)

[Gestures Used 4](#_Toc36312113)

[**Fist** 4](#_Toc36312114)

[**Finger Spread** 4](#_Toc36312115)

[**Wave Out** 4](#_Toc36312116)

[**Start, Quit, Pause, Restart** 5](#_Toc36312117)

[Custom Gestures. 6](#_Toc36312118)

[Hardware Used in Creating the Application 6](#_Toc36312119)

[**Speech Recognition** 6](#_Toc36312120)

[**Kinect** 7](#_Toc36312121)

[**Myo** 7](#_Toc36312122)

[Development 8](#_Toc36312123)

[Architecture for the Solution 8](#_Toc36312124)

[Conclusions & Recommendations 10](#_Toc36312125)

[References 10](#_Toc36312126)

# GitHub Link - <https://github.com/NeilK-94/Gesture-Based-Development-Project>

# Purpose of the Application

The aim of this project, from the project brief is to;

*Develop an application with a Natural User Interface using gestures to interact with it. For example, a voice controlled application fits the parameters of gesture based control. You can reproduce a classic game or system using a gesture-based interface.*

With this in mind, I decided to create a gesture-based game in Unity using the Myo armband as the primary tool to control on screen actions. I chose to use the Myo armband for my application for several reasons which we will go into detail on later. I also chose to add voice controls to the game later into project development for managing the game’s state as I felt it made transitioning between the game restarting more seamless.

The game itself is hardly a new concept. The player controls a spaceship which is endlessly moving along an axis. Asteroids and enemy ships are spawned on the top of the screen and the player can shoot them to earn points.

The game has largely been adapted from an existing but deprecated Unity 3D tutorial. I chose to do this because I thought it could save a lot of time gathering suitable assets as they are provided in abundance in this file. I also felt I would still have enough of a challenge in implementing the Myo armband controls and the voice recognition controls. It’s important to note I coded the game myself and implemented the Myo controls and speech recognition myself.



# Gestures Identified as Appropriate for this Application

### **Myo Armband**

The Myo armband has 5 built in gestures. The armband reads electrical activity in the muscles in your forearm to identify which gesture you are making with your hand. I believe all the gestures Myo has to offer could fit in well for this game, my biggest challenge was trying to find a use for them.



The Myo device also has a 9-axis Inertial Measurement Unit (IMU) that includes a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis magnetometer. The direction and movement of the wearer’s arm are measured by these units and determined by analysing the spatial data provided. Orientation represents the positions of the band in terms of roll, pitch, and yaw. The angular velocity of Myo device is provided in vector form and the accelerometer represents the acceleration corresponding of Myo device [1].

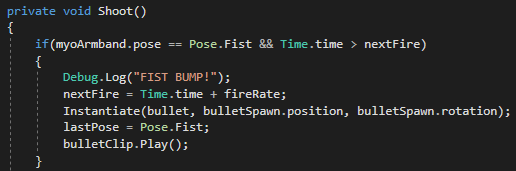
### **Microsoft Speech Engine**

I also identified the Microsoft Speech Engine as a suitable addition to this project. This is installed in Unity through the Universal Windows Platform SDK. I felt looking at my game practically, the ability to not need to touch the laptop would be quite important as it would be played standing up and away from the screen. The combination of voice controls and the Myo armband would make navigating not only the game but the menu’s, quite smooth.

## Gestures Used

### **Fist**

I think the one that almost picked itself was the fist gesture. It tied in well with the aggressive notion of firing a weapon on the spaceship. Having done some research on developer’s opinions on the Myo armband and the accuracy and consistency of its ability to pick up the gestures it seemed one of the more reliable gestures was in fact the fist. It was picked up more often then the likes of the double tap or finger spread. An example of how I got the fist to fire the ship’s gun can be seen below.



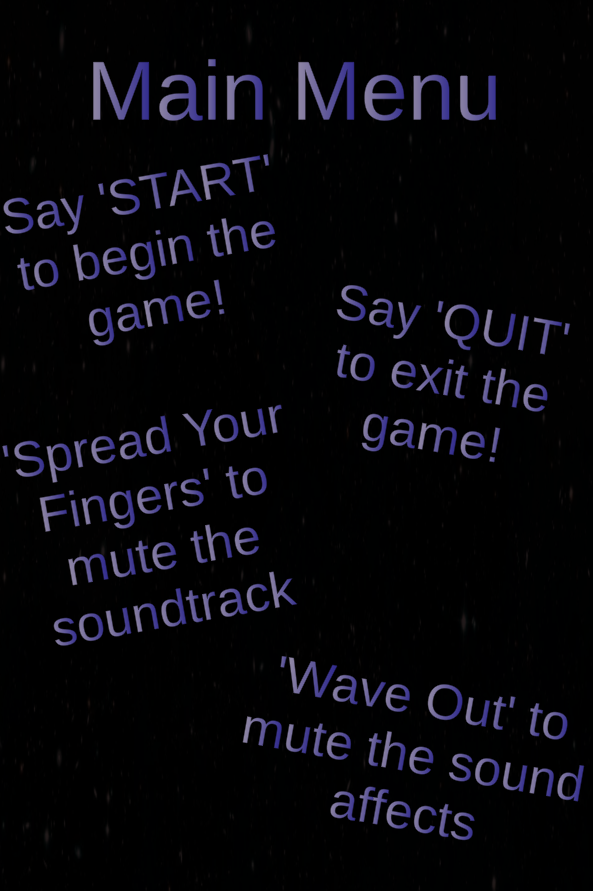
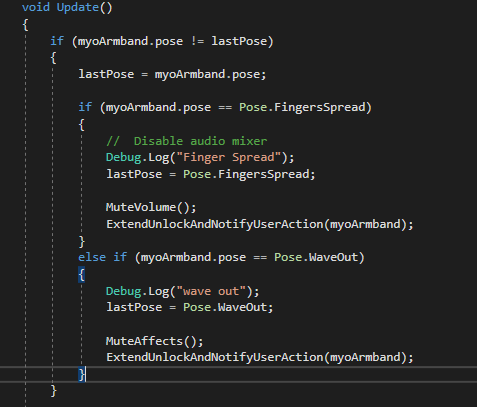
We check if our *myoArmband’s* current pose is equal to *Pose.Fist*. If so, the statement runs. We then set the last pose stored by our myoArmband equal to the fist pose. I added a short vibration to this to further immerse the wearer into the notion of shooting.

### **Finger Spread**

Another gesture I chose to utilise was the finger spread gesture. I used this on the main menu scene to mute the soundtrack. In most arcade games, the first thing most people do is mute the soundtrack, I felt it was important to use the Myo armband for this ahead of the speech recognition as if the volume is quite high on the user’s laptop, the speech recognition might struggle to hear the user’s command. I added a vibration to this to let the user know they’re gesture was read.

### **Wave Out**

The other Myo gesture I employed on the menu scene was the Wave out, this controlled the sound effects in the game. Like the finger spread gesture, I didn’t want to use speech recognition to control the sounds on the menu. I added a vibration to this also to let the user know that their gesture was read.



Above is the if statement that checks for the Myo gestures. If triggered, they simply call their corresponding methods that manipulate their respective audio mixers.

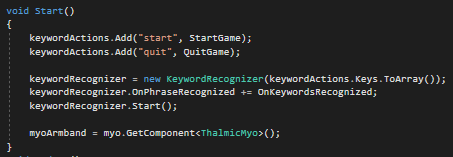
### **Start, Quit, Pause, Restart**

The voice control is used in two areas, the main menu and the actual game. In the main menu the user can start the game by saying ‘start’ and can quit the game by saying ‘quit’. The user is prompted to do either of these with text boxes on the screen. Below is an example off setting up the *KeywordRecognizer*.

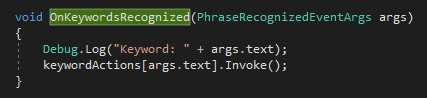


First we set up a Dictionary which will store a key, value pairs called *keyActions*. In our application, an example of a key value pair is “start” and the ‘*StartGame’* method.

We also create an instance of a *KeywordRecognizer*. The Keyword Recognizer listens to speech input and attempts to match uttered phrases to a list of registered keywords [2].



Above we populate our dictionary of keywords and their corresponding methods. We also instantiate our *keywordRecognizer*. *keywordRecognizer.Start()* Makesthe phrase recognizer start listening to phrases. *OnPhraseRecognized* is an event that gets fired when the phrase recognizer recognizes a phrase.



*OnKeywordsRecognized* is our own method that takes in the phrase recognised, prints it to the console and invokes it.

In the game the user can say ‘pause’ to stop the game and bring up the pause screen. From here the user can say either ‘resume’, ‘quit’ or ‘menu’ to do each respective action.

Largely the same logic and coding methods were applied to the pause menu voice controls, so I won’t go over how I implemented this again.

## Custom Gestures.

Unknown to many, the Myo armband can actually have custom made gestures outside of the pre-set 5. This involves

Developers can combine these pre-set gestures with arm motions (data from the Inertial Measurement Unit) to create new gestures. Meaning you would move your arm in different ways while performing these gestures in tandem this gives the potential for hundreds of different gestures [3].

I considered implementing custom gestures into my application but felt in the end there was no requirement for them. I didn’t even use all the 5 pre-set gestures so trying to implement custom gestures would’ve just taken up time and energy that could’ve been better spent elsewhere.

# Hardware Used in Creating the Application

For this assignment, we were given the choice of incorporating any type of hardware we wanted. I will go over some of the hardware I considered using and finally the ones I chose giving my reasons for each.

* Speech Recognition
* Kinect V2
* Leap Motion Controller
* Myo armband

### **Speech Recognition**

I added speech recognition a little later into development of my project. I was on the fence at first as I had had problems getting it working in the labs. However I thought the amount of work done on my project with just the armband was a little on the light side and I thought it could actually add to the game and not just be thrown in for the sake of it.

If a user is playing the game they’re likely standing back a bit from the screen, yes I could’ve added Myo gestures for the functionality the speech recognizer deals with, however, I feel the armband works better when some of the 5 pre-set gestures are left unused. Often it picks up gestures that aren’t being made. If every gesture had a function in a game, you’d often find things happening that weren’t supposed to. This could for example, lead to the game being paused when the user tried to shoot! That would be quite annoying. Much of my implementation was based off what was done in the labs.

### **Kinect**

The Kinect was a very close second in my choice of hardware to use. I think perhaps the only reason I decided against it was that the Myo seemed a bit more unique as far as gesture controls (With it not using a camera) and also the fact that I know a lot of my class members were using it and there might have been a high demand for them. I don’t own one myself meaning I would’ve possibly gotten limited time with the equipment.

The main benefits of using the Kinect v2 would’ve been that it comes with an SDK and it’s also apparently very straightforward to connect with unity [4].

The negatives would be that it is a little more cumbersome, requires an adapter and more actual space around the user when using it. This would’ve also been irksome while developing and testing with it.

### **Myo**

A big reason I chose to use the Myo armband was because I had never heard of it prior to this module. What struck me about it was that it read muscle movement using in the user’s arm and used that as data on top of the IMU data for spacial mapping. Unlike most gesture hardware this does not use a camera or lense to capture the user’s movements. It relies purely on reading the electrical activity in your forearm, to do this it uses 8 EMG sensors on the inside of the armband [5].



I also chose to use it because I felt I had gotten a reasonable hang of it from the labs, it was the first piece of hardware we covered and we received our project brief during the time we were still covering it. It made sense for me to choose something readily available that I was familiar with.

Lastly, unlike some other gesture-based equipment, it’s very easy to setup and jump right in with. All you need is the armband, the Myo SDK which includes the Unity package and a Bluetooth adapter. The battery life lasts for well over a week on full charge and it was easy to quickly test features of my game while wearing it.



## Development

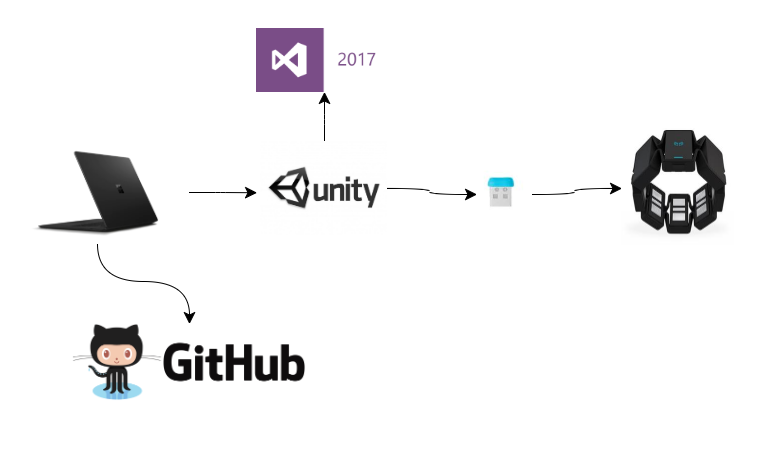
To develop the application, I used Unity 2019 and Visual Studio 2017 as my text editor. Unity allows for development in either C# or Javascript. I opted for C# as I feel more comfortable with game development in it. When making a game I always choose to develop it in Unity as I feel there is ample documentation available online and the editor itself is very easy to use and understand.

I used Git throughout development to manage my application as I do with all projects.

# Architecture for the Solution

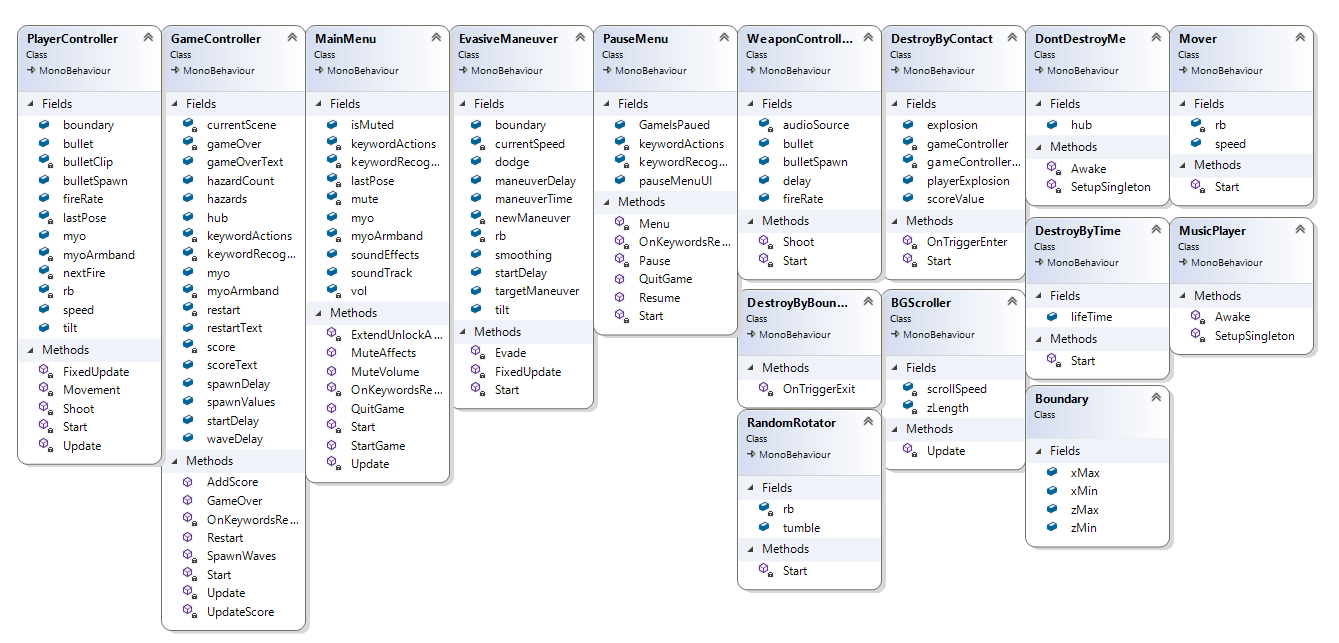
The technologies used were:

* Myo Armband
* Unity 2019
* Visual Studio 2017
* Git

****

The above diagram shows the architecture for my application. The Myo armband connects to the Bluetooth USB adapter connected to the laptop.

All the scripting was done in visual studio and unity is then able to handle all the import files from the Myo SDK that are used when running the application. All project files are then backed up on Github.



Above is the class diagram. As a Unity project, all the classes are inheriting from MonoBehavior. The class diagram above is not all that useful as they are all just stand-alone classes. However, I added it as it is requested in the brief and it shows the variables and methods used in each script.

Some relevant libraries used in developing this application include:

* TextMeshPro
  + This was used for creating text boxes to give the user prompts on how to navigate through the game with voice commands and gestures.
* Thalmic.Myo
  + This includes all classes needed for developing in Unity using the Myo armband.
* UnityEngine.Windows.Speech
  + This is needed for speech recognition in Unity. It’s installed through the UWP add on for Unity.

# Conclusions & Recommendations

In conclusion, I feel like I have learned a lot from this project. It was my first time developing something for an external piece of hardware. It brought with it a lot of challenges at first as there isn’t a great deal of information out there on developing with Myo in Unity. However, I feel I managed well and can say I know a good deal about it’s capabilities and could develop applications for it in the future much easier.

If I were to do it again, I would probably have investigated creating custom gestures. They would require a greater amount of time in development, but I think they would definitely show that the developer has a good understanding of the technology.

Another thing I would do differently is add more levels to the game and also add more functionality such as boosters, boss levels, different enemies etc. I didn’t add them in to this application due to time constraints and because the main aim was to demonstrate gestures.

The speech recognition worked very well and consistently. It wasn’t difficult to add to the project in Unity and I feel it was worth implementing.

All in all, I would cautiously recommend the Myo armband for development. At first, I found it very finicky and difficult to consistently register the desired gesture. However, I feel this is the opinion of many who haven’t given it a fair chance, once the armband is given significant time to warm up, and it’s tight enough on the wearer’s arm, the results are quite consistent.

# References

1. [www.ncbi.nlm.nih.gov/pmc/articles/PMC5796387/](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5796387/)
2. <https://docs.unity3d.com/ScriptReference/Windows.Speech.KeywordRecognizer.html>
3. <https://support.getmyo.com/hc/en-us/articles/210428723-Can-I-create-custom-gestures->
4. <https://en.wikipedia.org/wiki/Kinect#Kinect_2_for_Windows_(2014)>
5. <https://medium.com/thalmic/what-can-the-myo-armband-actually-do-93f431877240>