Gesture Based UI Development Project

Ross, Heaney
G00345608

GitHub
https://github.com/ross39/Gesture-Based-UI-development-

March 2020

1 Purpose of the application

The purpose of the application is to allow the user to play the classical game of space invaders[3] using the myo armband [4] to interact with the game. The player, using the myo armband, will be able to move left, move right and shoot in accordance with the rules of space invaders. A score will be kept on a local stored locally using player preferences. I also did this project solo.

2 Gestures identified as appropriate for this application

The gestures used in the game are designed to be as true to the original game as possible. Therefore I decided to use the gestures as follows

- Wave left - player spaceship moves left
- Wave right - player spaceship moves right
- Make a fist - player spaceship shoots projectile

I decided on these three gestures as they are intuitive and easy to remember. The idea was that even a small child could play the game. The actual implementation was as follows. Download the myo armband sdk and import it into unity. I then simply imported it into unity and from there I could program the interaction between the user and the player. Overall it was a pretty painless process. I found the fist gesture to be much better than the clicking the finger and thumb gesture. This was just my experience. I also included some vibration feedback for the user.

3 Hardware used in the creation of the application

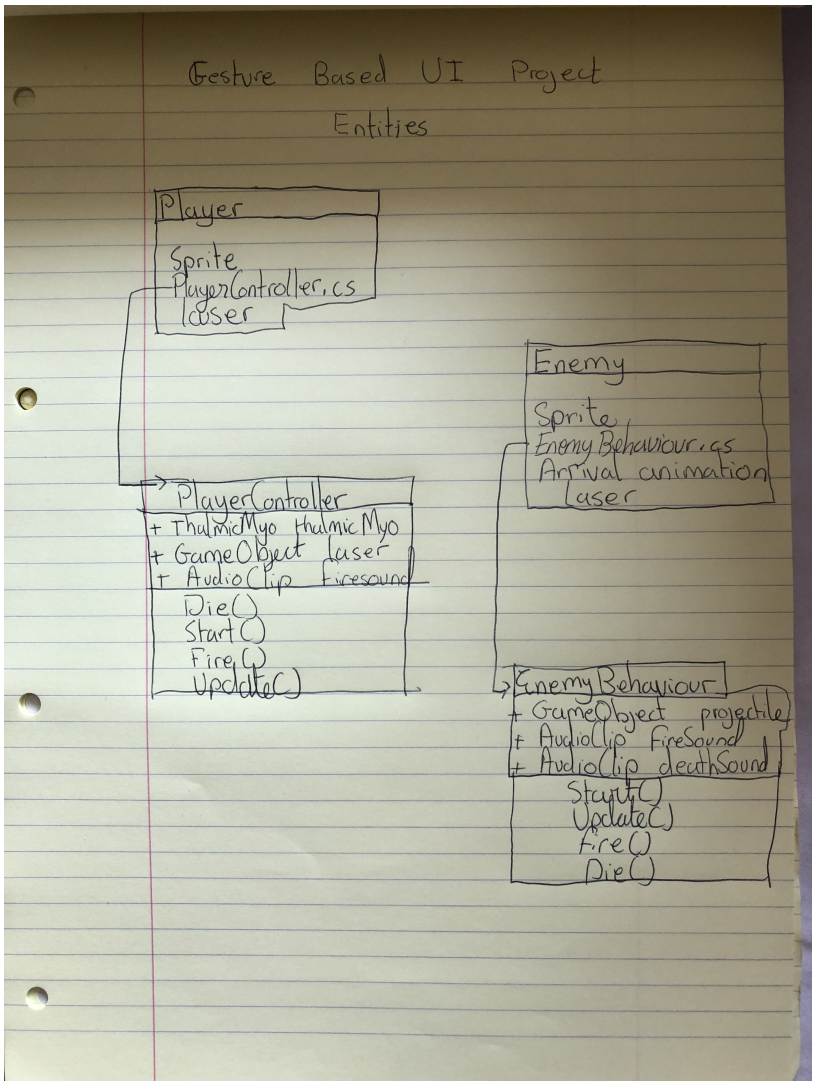
The Thalmic labs myo armband was used in the creation of this game.The Myo armband is a gesture recognition device worn on the forearm and manufactured by Thalmic Labs. The Myo enables the user to control technology wirelessly using various hand motions. It uses a set of electromyographic (EMG) sensors that sense electrical activity in the forearm muscles, combined with a gyroscope, accelerometer and magnetometer to recognize gestures. [3] The Myo can be used to control video games, presentations, music and visual entertainment. I decided to just stick with the myo armband that way the controls remained intuitive and there was no need for the added complexity of somethinng like speech recognition. In essence I wanted this to be a plug and play game with little to no overhead and memorisation of the controls.

4 Architecture for the solution

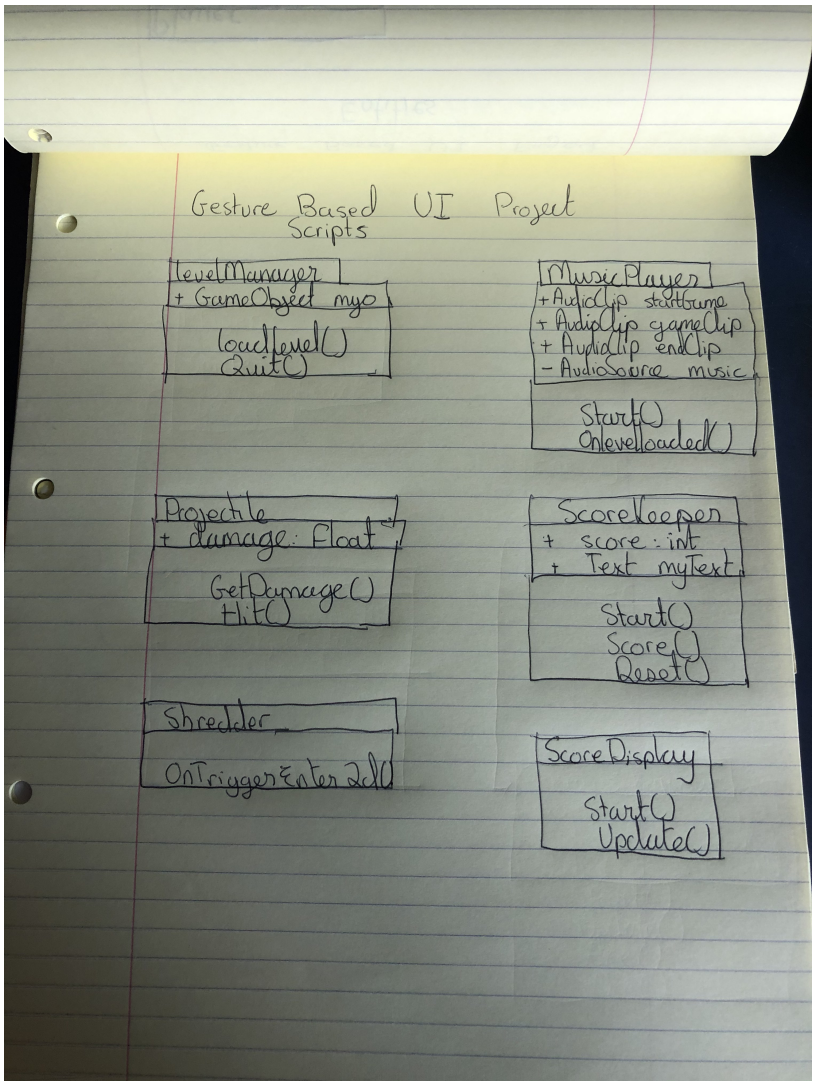
I have included a hand drawn architecture overview of the main clases used in the project. I have two reasons for this

- I had these already drawn out before commencing the project
- It was painfully annoying to try and generate the class diagrams on mac. Visual studio only supports this on windows.

4.1 Hand drawn class diagram of entities used in game



4.2 Hand dawn diagram of main scripts used in the game



4.2.1 The first screenshot

The first screenshot is an overview of the main entities using in the project. The player entity is controlled using the playerController script. It has it's own sprite and laser associated with it. Likewise the enemy entity is controlled using the enemyBehaviour script. The enemies have an arrival animation associated with them. This way when the player starts they will witness the arrival of the enemies on the screen and then the game commences. I found this to be more natural than the enemies just sitting there and then suddenly attacking

the player. The screenshot also shows the main breakdown of the playerController class. A ThamlicMyo object must be instantiated in order for the game to be compatible with the Myo armband. In terms of the code used to control the flow of gestures I used if statements. I realise now that a better approach would have been to use a while loop such as while(myo.pose!= lastpose). I could have achieved a much smoother flow of control which would have translated to smoother gameplay for the user because at the minute there is a slight lag between the user's movements and the character movements. The enemyBehaviour class is also broken down to it's main parts. In this we see the projectile and audio initialised. These refer to the lasers the enemies will shoot and the sounds they will make. The enemey formation code is how the enemies appear on the map and make the attacking structural formation.

4.2.2 The second screenshot

The second screenshot shows the main overview of the classes contained in the scripts folder. These scripts control the main game. I feel that they are fairly self explanatory. The shredder class controls the destruction of the enemies. The player score is stored using player preferences. I debated using SQLite for this but it was easier to store the score's using playerprefs. If the game was distributed or multiple players could play on the one device the SQLite would be a better choice but seeing as we're just storing a simple score I felt there was no need to use SQLite.

5 Conculsions and Recommendations

I learned a lot during the course of this project. I learned the following

- Hardware doesn't always work as nicely as software.
- There are many external factors that can effect how hardware works.
- User experience is arguably even more important when it comes to them interacting with hardware.
- While using multiple sources of interaction such as speech recognition and gestures can be nice, they can add a lot of unnecessary complexity.

Overall I really enjoyed this project. I did have one or two pain points such as figuring out the best way to give a smooth user experience and generating the class diagrams in the end. I used code from the unity developer course[5] and learnonline notes and labs[1]. I can see why the kinect wasn't a huge success despite the actual technology being a leap forward. Gesture interaction hardware can be clunky and the interaction just isn't as smooth and defined as a controller or keyboard. That combined with the fact that most people prefer to sit down and not be active when playing games or interacting with their machines, It's easy to see why gesture based interaction still has a good

bit to go. I would probably get a windows virtual machine if I was going to do the project again as I found the visual studio support on mac to be subpar to that of windows. One example is that I had a lot of trouble generating class diagrams for the C# code on mac. In visual studio for windows you can auto generate class diagrams but no such support exists on mac. I tried a bunch of different programs but after a few hours of trying to bend them to my will I threw in the towel and drew them as best I could by hand.

5.1 Bugs and where things went wrong

I had one or two bugs that I couldn't fix in time. The main one Is that when the player restarts the game unity says that there are too many myo armband objects. I suspect what is happening is that when the player dies in the game the objects are not being destroyed properly so when the game restarts there is more than one myo gameobject and unity doesn't like that. I have found the bug to happen on and off. As it's close to the due date for this project I decided to just leave it as I have been trying for a while to fix it. Finally the screenrecording[2] software I used had a max limit of 100 seconds so that's why it's short. The reason I used it because it required no login and it was free on the mac app store.

References

[1] *Gesture Based UI Development Learnonline GMIT.* `https://learnonline.gmit.ie/course/view.php?id=1833`. Accessed: 2020-04-06.

[2] *Screen record software.* `ScreenRecordHD-ScreenLite`. Accessed: 2020-04-06.

[3] *Space Invaders game.* `https://en.wikipedia.org/wiki/Space_Invaders`. Accessed: 2020-04-06.

[4] *Thalmic myo website.* `https://support.getmyo.com/hc/en-us/articles/203398347-Getting-started-with-your-Myo-armband`. Accessed: 2020-04-06.

[5] *Unity Developer Course Laser Defender.* `https://github.com/CompleteUnityDeveloper/06-Laser-Defender`. Accessed: 2020-04-06.