Gesture Based UI Development Documentation

WINDOWS AND ANDROID GESTURE DRIVEN APPLICATION USING UNITY

Stephen Kelly | G00361283 | 4TH year
https://github.com/Stephen313k/KrazyKart

Purpose of the application

The purpose of the application is to implement and carry out different gesture types compatible with windows PC and android devices. The application can be used entirely with touch gestures on the android build. The windows version has the capability to control the application with Myo-Armband and voice commands. The main goals for development of the application are:

- Implement an application compatible with the myo-armband gesture device, voice command recognition and touch screen gestures.
- Design an intuitive and responsive application for a fluid user experience.
- Develop an application that can be fully controlled through gestures and voice recognition.
- Provide accurate gesture recognition with practical functionality.

The gesture-based application is a single player 2D game called KrazyKart. The control types for each platform are:

1. Windows application
   - Player movement and actions controlled with Myo-armband.
   - Menu navigation and user interface controlled with voice commands.
   - Keyboard and mouse controls are also implemented.
2. Android application
   - Player movement controlled by tilting the device, and touch inputs.
   - Menu navigation and user interface controlled with swipe gestures.

The Game

The gameplay consists of moving a car across the x-axis of the screen to dodge oncoming cars that are randomly spawned. The player can shoot a bullet towards the top of the screen and will kill the enemy car upon collision. Enemies have a 75% chance of dropping ammunition packs. This powerup adds ammo to the players current amount (increased by one, the player starts with ten counts of ammo). If the enemy car collides with the player, then the game is over. There is a score counter that is multiplied as long as the player stays alive.

Start Screen

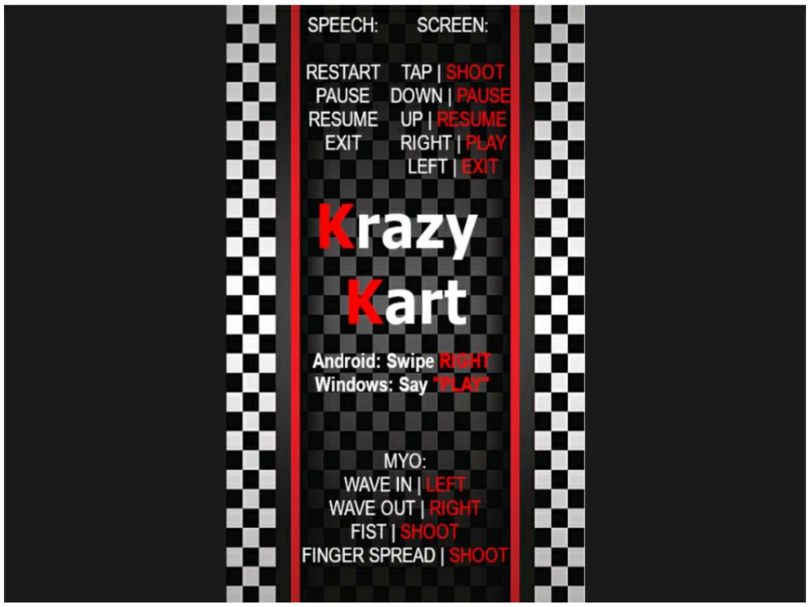

Fig 1. Start Screen

The start screen includes all the gesture types and commands that can be carried out by the user.

In Game

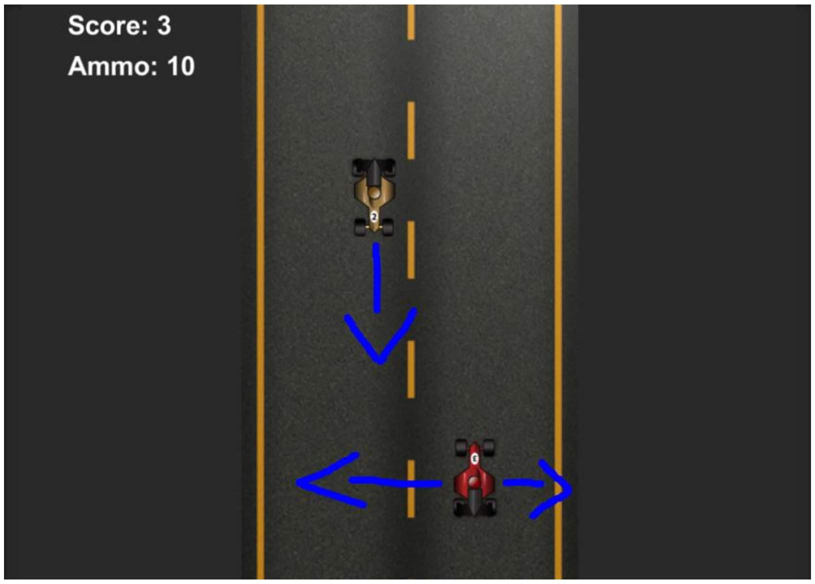

Fig 2. In game

In game gameplay is shown above, and the direction of travel.

Pause Menu

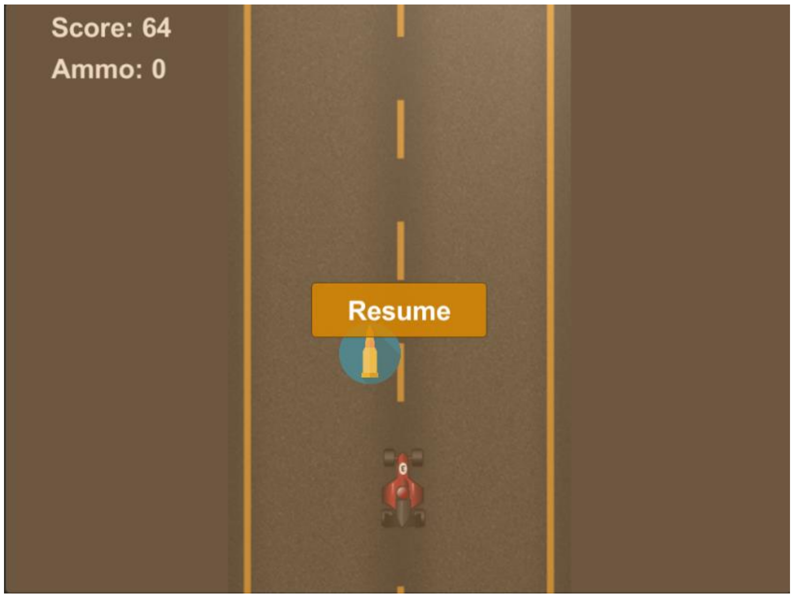

Fig 3. Pause menu

Player power up and pause menu is shown above.

Death Screen

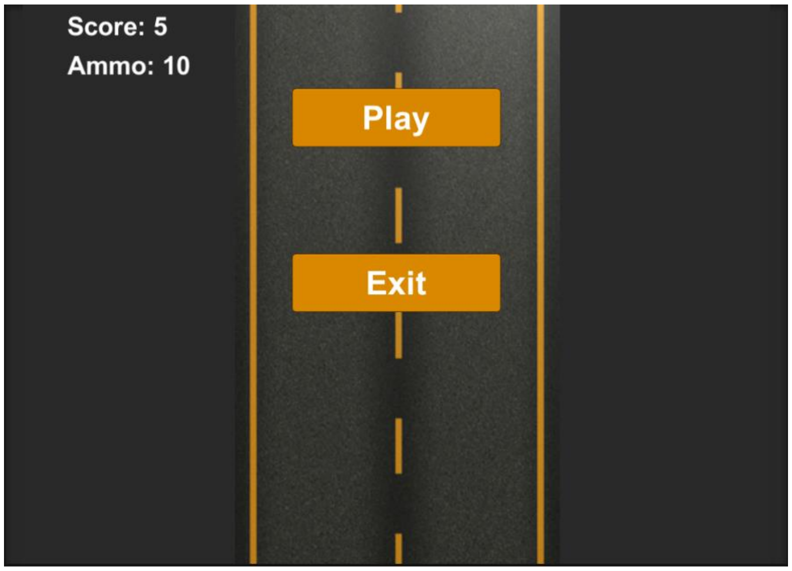

Fig 4. Death Screen

Above is the death screen, the player is removed and the option to play again or exit is present.

Gestures Identified

Myo-armband

The Myo-armband detects muscles activity to control and recognise gestures. The armband has predefined gestures that can be easily recognized after product installation, this includes wave in, wave out, double tap, fist and spread fingers. The armband contains an accelerometer and gyroscope for extra functionality and spatial control. The gestures implemented in my application are as below.

Rest

The rest gesture is simply the default position the user's hand should be. The rest gesture does not control the player however from this hand position other gestures may be carried out from for high accuracy. For example, when a user goes from a rest gesture to a wave gesture, the wave gesture is easily identified as it came from the base position.

Wave In

When the user waves in their hand as seen below, the in-game car will move left, the same direction the user's hand must travel. The car will continue to move left until the user changes the gesture.

Wave Out

When the user waves out their hand as seen below, the in-game car will move right. The car will continue to move right until the user changes the gesture or rests their hand.

Fist

When the player makes a fist with their hand a bullet is shot from the players location upward towards enemies.

Double Tap

Originally double tap paused or resumed the application, this was removed for voice commands.

Spread Fingers

Originally spread fingers restarted the game, this was removed for voice commands.

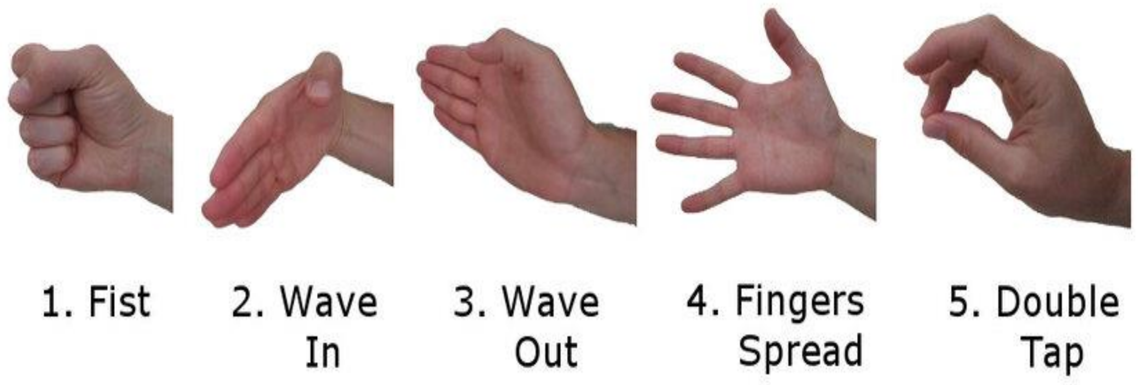

Fig 5. Myo Gestures

Voice Commands

Voice commands are used for menu navigation: pausing, resuming, restarting/respawning, and to exit the application. The windows speech library is used on unity with keyword recognizers to detect commands called by the user. This application originally had not planned for voice commands, initially I set the respawn/restart button to the myo-armband finger spread gesture, and the pause/resume function with the double tap gesture.

The issue with pausing and restarting the game with the myo-armband gesture is it could pick up a false positive gesture, randomly pausing or restarting the game while playing. The random restarts and respawns were not ideal for a functional application. While the user can use mouse and keyboard, I wanted to allow the player to control the game without having to interact with buttons. Voice commands became clear to me that is it the best solution for menu and UI navigation.

Start

To execute the start command, the user can say “play” or “restart”. This will essentially start the game from the main menu or respawn the player after their death.

Exit

To execute the exit command, the user can say “quit” or “exit”. This will completely terminate the application.

Pause

To execute the pause command, the user can say “pause”. This can be carried out while in mid-game and the application will pause, along with all player and enemy movement.

Resume

If the game is paused the user can resume the application, to execute this command the user can say “Resume”, the application will go to its default playing state, movement can be controlled, and the enemies are resumed also.

Android Gestures

The Myo-armband is not compatible with android, and the windows speech library used for speech recognition is also not compatible with the android version. The use of swipe gestures is used for control of the application, navigation, and UI. To control player movement the accelerometer is used and can be controlled by tiling the screen, a tap gesture is also used for player interaction.

Swipe Right

Upon opening the application, the user can swipe their finger towards the right side of the screen. This will either go from the main menu to start a new game. If the player is upon the death screen it will respawn the player and a new game begins.

Swipe Left

In contrast to the swipe right gesture, the swipe left action will exit and close the application completely. This is carried out when the user drags their finger across the screen towards the left side.

Swipe Down

To pause the game the user can drag their finger down towards the bottom of the screen, the game will be paused.

Swipe Up

If the game is paused the user can resume the game by dragging their finger up towards the top of the screen, the pause menu will close the game is then resumed.

Tap

If the user taps on any area of the screen while in-game the player will fire a bullet directly above the car’s location.

Tilt

To control the player along the X-axis of the screen the user may tilt their phone left to move the player left, correspondingly if the phone is tilted right then the player will move towards the right side.

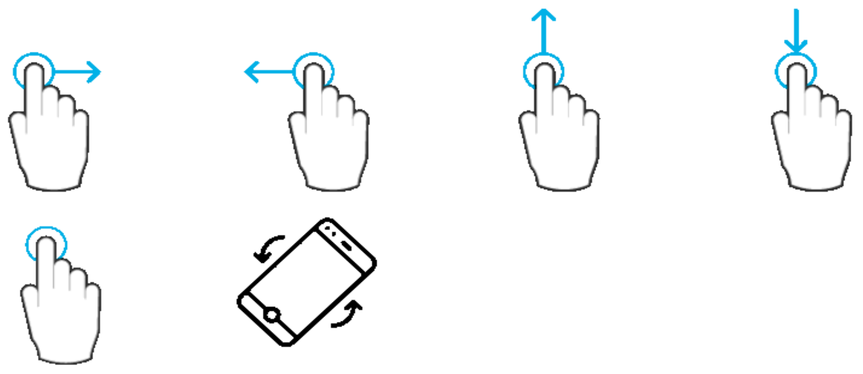

Fig 6. Android gestures and tilt

Summarize Gestures

While there are two version of the application, and three different gesture types I will briefly explain all twelve gestures implemented (eight movement gestures and four voice commands).

Windows

Myo-armband:

1. **Wave in** – Moves player left
2. **Wave out** - Moves player right
3. **Fist** – Fires a bullet from players location

Voice commands:

1. **Start** – Starts the game/respawns
2. **Exit** – Exits application completely
3. **Pause** – Pauses in-game
4. **Resume** – Resumes if the game is paused

Android

Swipe/touch gestures:

1. **Swipe Right** – Starts the game/respawns
2. **Swipe Left** - Exits application completely
3. **Swipe Down** – Pauses in-game
4. **Swipe Up** – Resumes if the game is paused

5. **Tap** – Fires a bullet from players location

Hardware Considered

While planning the project and defining the scope there were two hardware technologies considered using for this project, Myo-armband, and Microsoft's Kinect. While developing the application I explored the option of using a mobile device for an additional platform.

Microsoft Kinect V2

The Kinect V2 is a sensor produced by Microsoft, it's composed of a RGB camera, infrared camera, infrared emitter, and a microphone. The Kinect is capable of facial and body recognition, it's able to calculate the distance moved of each point/joint on the user's body by using the infrared light and measuring the time travelled as it bounces off its surroundings. The Kinect comes with four different microphones that are used to differentiate between the user's voice and background noise. The Kinect is used primarily for games that require full body tracking.

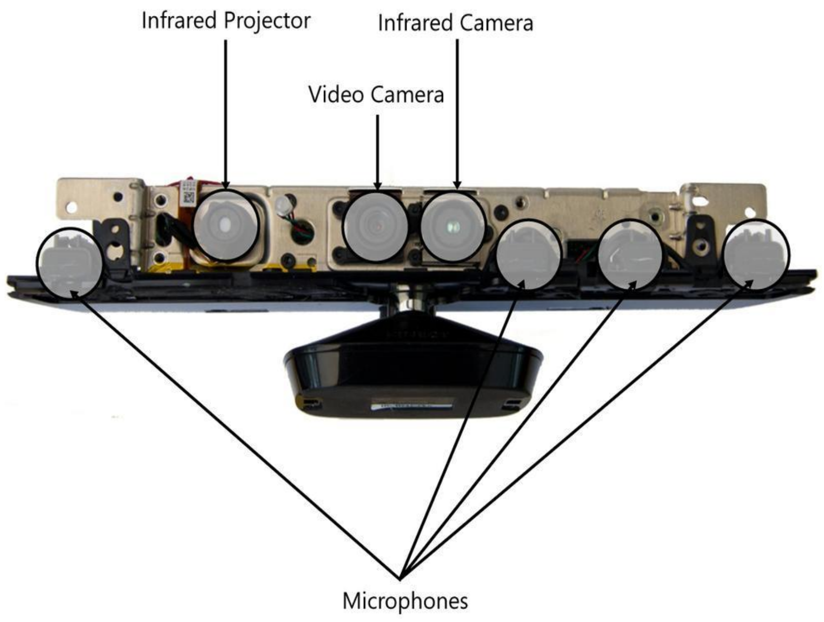

Fig 7. Inside the Kinect

Myo-armband

The Myo-armband is a wireless device that the user wears on their forearm. Unlike body tracking with the Kinect, the main purpose of Myo-armband is to detect when gestures have been carried out (Only on the arm that the armband is being worn on). The armband works by using eight electromyographic (EMG) sensors and inertial measurement unit (IMU) containing three-axis gyroscope, three-axis accelerometer, and three-axis magnetometer. This is used for spatial recognition in an application, an example is a 3D game Race The Sun.

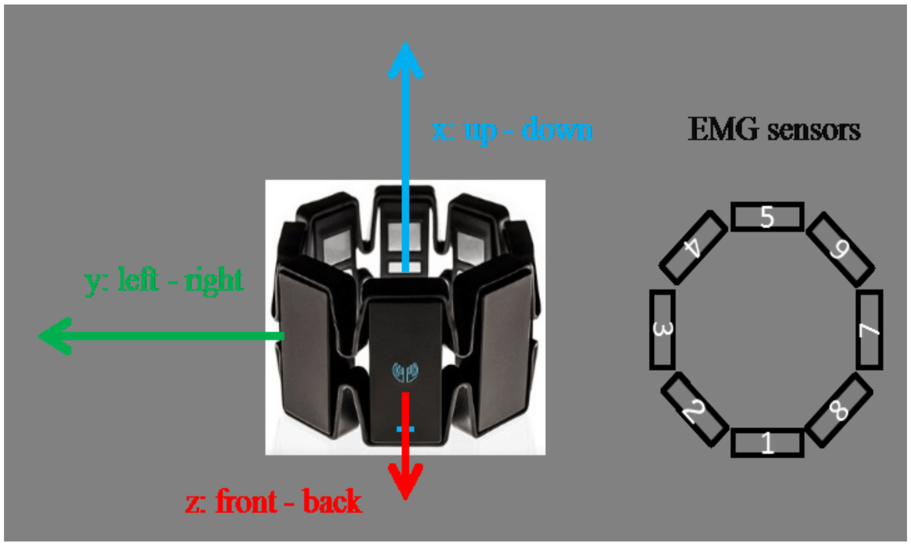

Fig 8. Myo armband

The armband can detect the intensity of the gestures carried out by the user. This is another thing that differentiates itself from the Kinect. The armband can be used for many things such as controlling a slideshow presentation, video playback, gaming and more. The five pre-installed gestures (wave in, wave out, fist, double tap, finger spread) on the armband are easy to remember and is quick with gesture detection. Installation for the Myo-armband is simple as you slip it on your forearm and sync the five gestures after carrying out each gesture in the menu panel.

Mobile device

Smartphones have many sensors such as an accelerometer, a gyroscope and obviously a sensor to detect touches on the screen. The accelerometer is a sensor that detects the different motions of a device such as tilting or rotating. The accelerometer simply detects movement on the X, Y, and Z axis. The gyroscope is a more advanced feature that can be used in 3D applications or virtual reality. The gyroscope sensor is responsible for sensing angular rotational velocity and acceleration.

Most used on mobile devices is the touchscreen feature for gestures or input. Screens on a smartphone device can detect specific gestures such as tap, swipe, or pinch. This is usually very accurate and fast in detection. Another feature that can be exploited is the camera on a mobile device, programs such as Vuforia for augmented reality can be used to recognize objects, images, or interact with your real-world environment.

Hardware Selected

The hardware that was selected for the final version of the application are Myo-armband and Android mobile devices. The Kinect was originally explored and researched for the primary hardware. However, the main purpose of the Kinect is for applications that require a lot of body movement. As my game controls are relatively simple (moving across the X-axis and shooting) this technology seemed to be excessive for my game. Having compared the scope of my project to the hardware available to me, it became clear that Myo-armband would best suit my needs for the project. The option of having five gestures to control my application made sense as I planned, I would have been able to use all five gestures for complete control of the application.

By having an Android mobile device and an interest in building mobile applications, it made sense to try and add an android version of the game on top of the Windows version using Myo-armband and speech recognition. Having explored that Android devices come with a built-in accelerometer it was clear it would be the perfect way to control the player in-game. To take things a step further I realised that mobile devices are capable of many swipe gestures, by combing this with the accelerometer I knew it was possible to build an application that can be fully controlled with gestures.

Solution Architecture

Architecture Design

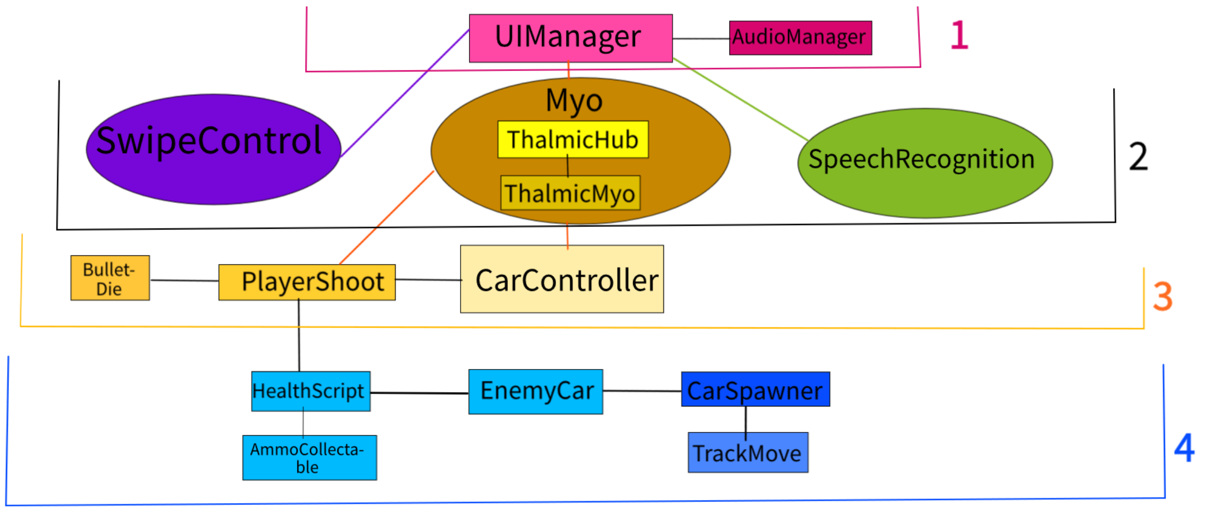

Fig 1. Solution architecture

There are essentially four layers that interact with another in my system design. The UIManager and Audio Manager can be considered as the first layer for application navigation and settings. The second layer is all the gesture scripts I have implemented, SwipeControl, Myo-scrips and SpeechRecognition communicate with the layer previously mentioned and the third layer for player controller. The player control layer consists of a CarController, PlayerShoot and BulletDie, this layer is used for player actions and movement. Finally the fourth later is everything enemy related including EnemyCar, CarSpawner, HealthScript, AmmoCollectable and TrackMove.

Core Technologies

- Unity – version 2019.4.38f 1
- Myo Connect – version 1.0.1
- Android Mobile Device

Core Libraries and Development Tools

- Myo - ThalmicHub, ThalmicMyo pre-installed scripts

- Unity Speech Recognition API
- Android SDK
- Scripts written in C#

Scripts

Layer 1

UIManager

This script is responsible for starting the game, exiting the application, pausing, and resuming while mid-game. All scripts in the 2nd layer are coupled with the UIManager for full gesture controls related to the navigation in the application.

AudioManager

The AudioManager is a simple script responsible for the in-game sound.

Layer 2

Myo Scripts

The two scripts to implement and recognise Myo gestures were pre-installed and connected to the rest of my scripts that I made independently.

ThamicHub

The ThalmicHub script essentially implements the connection from the myo-device to the application.

ThamicMyo

The ThalmicMyo script is responsible for recognising when the user carries out any of the five default gestures.

SpeechRecognition

For Windows speech recognition, this script is linked only to the UI manager to control navigation of the application through voice commands. This includes starting the game, exiting the application, pausing, and resuming while mid-game.

SwipeControl

The swipe control script is responsible for the Android version gestures related to navigation of the application. This includes starting the game, exiting the application, pausing, and resuming while mid-game.

Layer 3

CarController

This script is responsible for the movement of the car across the x-axis. This script contains the Android accelerometer movement method and Myo-armband gesture detection to move the player. The car controller script is responsible for detecting collision against an enemy.

PlayerShoot

As the name implies this method is to fire a bullet from the player towards the top of the screen. The android method TouchShoot is implemented and triggered when the user taps anywhere on the screen. In the update method the application is listening for Myo gestures and when a fist is recognised the shoot method is called.

BulletDie

This is a simple script to destroy the players bullets after a set amount of time, to keep the application running smoothly.

Layer 4

EnemyCar

This is a simple script to control the pace at which the enemies move towards the player.

CarSpawner

This script is responsible for spawning enemies on the road, there is a position boundary randomised for unpredictable spawns, a spawn timer is added to consistently spawn cars at a certain rate.

HealthScript

The health script is responsible for removing the enemy car when a player bullet hits it. This contains the rate at which ammo packs are dropped from enemies.

AmmoColletable

Linked with the HealthScript when an enemy dies, they may drop an ammo pack. When the player collides with the free-falling ammo pack, they receive additional ammo and the ammo count is updated.

Conclusions

What I Have Learned

I have learned how to implement hardware technology into an application (Myo-armband). I have also never created an android application prior to development of this project. Not only have I created an android application, but I have researched and learned how to implement two new gesture types, swipe/touch gestures and gestures carried out using third party hardware linked to software. After extensive research on gesture types and recognition it became clear to me that gestures must serve a practical purpose for users to create a fluid interactive application. If gestures are confusing for users or are not easily recognised then the application as a whole will suffer and provide poor user experience.

Future Development

Recommendations for future development consist of adding more in-game features. A high score list of previous games to track your high score would be ideal so users have the goal to beat their previous record. Multiple levels could be added and different enemy types that drop more powerups (invincibility, double points, and lives). A life system would be a feature to improve gameplay and would reward the player attempting to pick up powerups. Unit testing to test features of the application should have been implemented to ensure functional and re-usable code. Future development could use different interactive hardware rather than the Myo-armband as that product is discontinued.

Conclusion

Overall, I am very happy with the application I have created. I was able to utilize my skills from previous years of development to create a gesture driven application. Initially the project seemed daunting having no prior experience with gesture controls and deciding the scope of the project, and project research took a lot of time and delayed starting the development of the project. Having better time management, I would have implemented testing to my application to ensure quality coding standards, however I was still able to loosely couple methods and scripts with another. As I was building my application, I would often test out the gestures and features, this made me realise that voice commands were a solid option for the UI and application navigation, as the Myo-armband can pickup gestures that the user did not imply (pausing and restarting randomly). The game is very simplistic and does not have many features, however all my focus went into implementing practical and different gesture types. If I had another partner for the development of the application the game would have seen more bells and whistles. To summarize I was able to create an application that can be fully controlled with gestures on Android, and on Windows. This is a big accomplishment for me.

References

Android Gestures: https://www.androidauthority.com/android-10-gestures-1025113/

Myo armband review: https://time.com/4173507/myo-armband-review/

Unity Windows speech recognition: https://docs.unity3d.com/ScriptReference/Windows.Speech.PhraseRecognitionSystem.html

Kinect: https://en.wikipedia.org/wiki/Kinect

Unity 2D games: https://unity.com/solutions/2d

What is an accelerometer: https://www.fierceelectronics.com/sensors/what-accelerometer

Gyroscope: https://en.wikipedia.org/wiki/Gyroscope

Race The Sun: https://store.steampowered.com/app/253030/Race_The_Sun/