Apple Watch Game

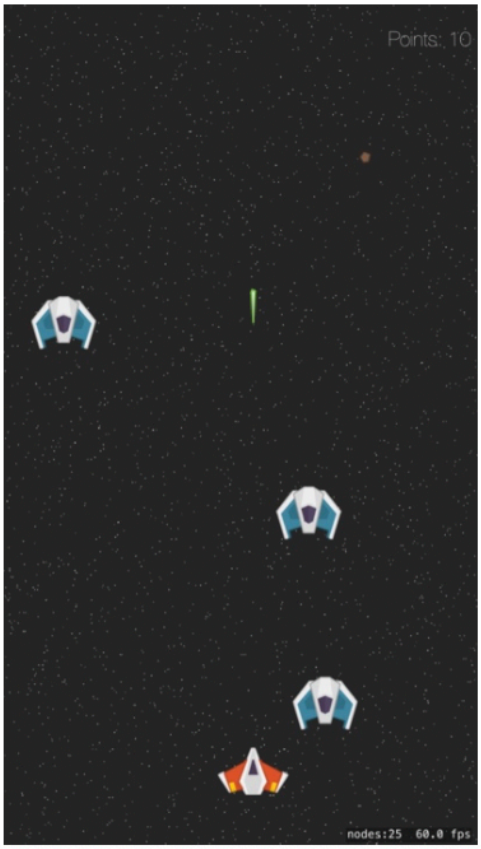
Purpose of the application

The purpose of this project was to create an innovative game which utilized the sensors on an Apple Watch to control player motion on the screen of a paired iPhone. In order to do this an iOS app running on an Apple iPhone, iPad or iPod Touch and a WatchOS app running on a paired Apple Watch was needed. To play the game, the user rotates their arm that is wearing the watch and this movement controls the direction of the spaceship and shooting the enemy ships to gain points. The objective of the game is to accumulate as many points as possible whilst avoiding the enemies.

After much research and consideration, we came to this idea as we wanted both a challenge and to create something that had previously not been done before. Our chosen project was going to be a challenge for a couple of reasons. One we had not previously done any native development on either iPhone or Apple Watch, and on top of this the language we would be writing in was going to be Swift, which we did not have any experience in. The second reason that this would be difficult is that during our initial research there were many aspects to creating this kind of application that were either unclear if they were possible or there was a very limited amount of information that we could draw from online.

We are also using this project as an experimentation process in the gesture-based U.I. field to try out different types of hardware that can detect various gestures and movements. The first hardware device we were working with was the Myo Armband that picked up five different gestures that could be programmed to carry out a specific function. This induced a thinking process revolving around which gesture would trigger which function and how this changed with respect to the task required. The decision-making process was one of trial and error mainly, where we'd try various gesture binds to find the most free-flowing, practical use of the gestures available.

Gameplay Screenshots:



Gestures identified as appropriate for this application

The assignment given to us in our Gesture-Based UI module was to develop an application with a natural-user interface. This gave us a broad range of options to implement the app including voice-control, hand gestures or device-motion control. There were various choices to make in regards to what one we were going to implement, but we felt the device-motion controlled application was the best option for us as it is currently the most robust and prolific form of gesture-based.

The next decision for the project was the actual implementation of the gesture based UI and we considered a couple of options. These consisted mainly of a Unity Games where we incorporated the gestures of the apple watch to control the character to complete levels in 2-D platformer scenarios. This proved difficult for us due to the limited compatibility and documentation of the Apple Watch and the Unity Software. Other ideas included the classic 2D game PACMAN, Temple-Run and Tetris. In the end we chose a 2D scroller space-shooter game and we decided to develop it in Swift as there is more support and references developing in Swift using the Apple Watch.

There are three main gestures available for the Apple Watch which are pitch, yaw and roll. Pitch reports rotations around the device's x axis, roll reports rotations around the y axis, and yaw reports rotations around the z axis.

As designers and developers, it's important to build smooth user flows to create frictionless interactions for users. The main thing we had to keep in mind was the User-Friendliness of the gestures and being able to transition between different states of the game with ease.

In terms of the rationale for our decisions as to which gestures trigger which actions in the game, we felt the Pitch gesture for the movement of the spaceship object in the game was best fitted and the most physically natural choice for the user. The other gestures tended to be quite jerky in motion and resulted in inconsistent movement from left to right on screen.

We spent a considerable amount of time and effort on this stage of the project as we valued it highly in relation to the project as a whole. The decision rationale of the appropriate gesture can easily be a defining factor in the usability and success of gesture-based applications.

Hardware used in creating the application

In terms of hardware options, GMIT has various devices available to students including the Myo Armband, LEAP motion controllers, Kinect V2, HoloLens and Durovis Dive. Shane also owns an Apple Watch which gave us a another option. After some research and experimentation, we both decided to use the Apple Watch as we were both familiar with Apple's mobile operating system iOS. We were also interested in learning the Apple programming language Swift -which is developed by Apple Inc. for iOS and macOS and working in the XCode development environment.

We also found that the Apple Watch is somewhat similar to the Myo Armband so the research already compiled could be of some use going forward with the project.

In comparison with the Myo Armband, we found the Apple Watch much more responsive and accurate. Although the gestures available programmed in the Myo were more sophisticated and detailed, we found the accuracy lacking for a reasonably fast-paced shooter game. We also discovered that two gestures in quick succession with the Myo often yielded different results therefore we couldn't predict a control function to be carried out in the game.

The Apple Watch device and it's implementations can be utilised innovatively in various scenarios in the real world. The main hardware features of the device we investigated were the accelerometer and the gyroscope. The motion data extracted from these can alert the system to specific movements by the user and track the type of the user movement as well.

An accelerometer measures changes in velocity along one axis. All iOS devices have a three-axis accelerometer, which delivers acceleration values in each of the three axes x, y and z. The values reported by the accelerometers are measured in increments of the gravitational acceleration, with the value 1.0 representing an acceleration of 9.8 meters per second (per second) in the given direction. Acceleration values may be positive or negative depending on the direction of the acceleration.

A gyroscope measures the rate at which a device rotates around a spatial axis. Many iOS devices have a three-axis gyroscope, which delivers rotation values in each of the three axes. Rotation values are measured in radians per second around the given axis. As with the accelerometer the rotation values may be positive or negative depending on the direction of rotation.

The most common use of this hardware is in fitness apps, where the gyroscope and accelerometer values are extracted and relayed in a useful manner to the end user.

Architecture for the solution

As stated above the game requires both an iPhone and Apple Watch to be played. This is because the game runs as an app on the phone whilst the companion watch app records and sends live data to the phone which is processed and used to control the onscreen avatar.

iPhone

The app running on the phone is the 'game' part of this project. It creates the screen in which the gameplay occurs. The player spawns at the bottom of the screen whilst enemies spawwn randomly at the top and fall down. The player moves left and right depending on the inputs given. The players spaceship continuously fires a laser and when they scored a hit the enemy explodes and the score is increased.

Apple Watch

Meanwhile on the watch app, the wearers arm rotation is recorded as attitude data (pitch) every tenth of a second and this data is sent to the gamed app on the paired iPhone.

Code

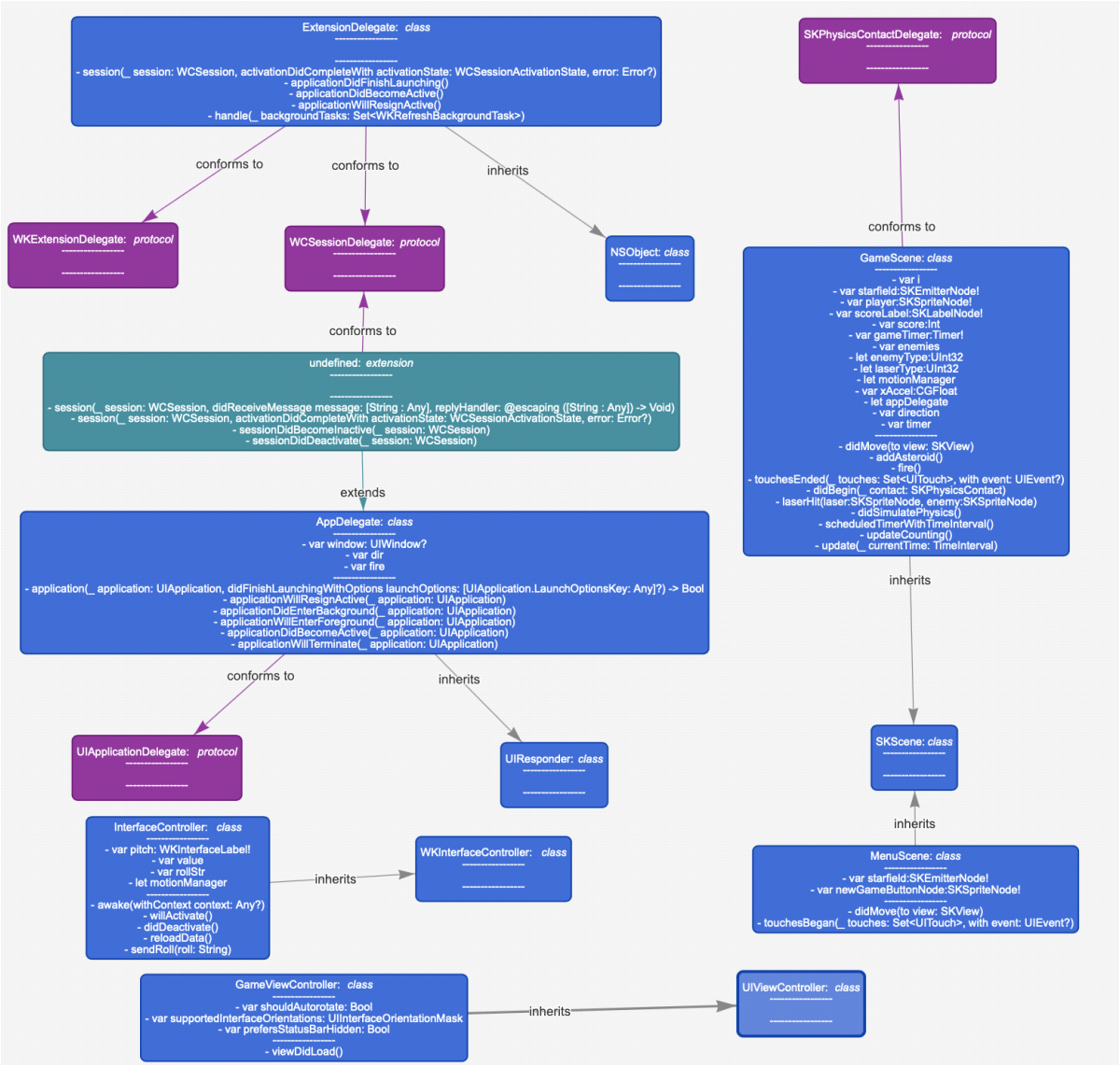
The main game loop is created in the GameScene.swift file. This is where the scene is set up, the background starfield emitter, player, enemy spawner, motion manager etc. are added and the timer for auto-fire is initialised. Separate functions handle what happens when there is player movement, spawning enemies, firing, contact and the timing. The names of the enemies are kept in an array which is selected from randomly when the addEnemy function is called. This would enable us to easily add more enemies and enemy types in the future. The fire function is called at a constant interval using the timer. It gets the image from Assets.xassets sets its position to a point slightly above the player, sets the physics body for collisions and gives the laser a set velocity.

In the AppDelegate amongst other things the WCSession is set up. This is what allows the data transfer between the paired watch and phone and also where a global variable is updated so that the movement data can be accessed in the GameScene. Here it is checked to see if the

session between the iPhone and Apple Watch are active and if so receives the dictionary sent in the message and using the received value sets the global variable.

On the watch app, the Interface.storyboard is set up with a basic label to tell the user how to play. The InterfaceController is where the CMMotionManager is set up and begins recording environmental data at a set interval of 0.11 seconds. This number we found was the optimal time as anything quicker would take too much data to send and would clog up the data stream. The data being sent is the pitch of the watch. This pitch data is sent as a dictionary over the WCSession every interval of data.

Class Diagram



Libraries

- WatchKit:

The WatchKit framework contains the classes used to create a WatchOS app. A WatchOS app may contain elements such as tables, buttons, sliders etc. and these visual components are manipulated by WatchKit to respond to user interactions.

- WatchConnectivity:

This framework is used to transfer data between the paired Apple Watch and iPhone. For this project we utilized WatchConnectivity to send live data from the Watch to the phone.

- CoreMotion:

CoreMotion is used to report environmental data taken from an iOS devices onboard sensors such as the gyroscope, barometer, manometer and G-meter.

- GameplayKit:

Used for building iOS games, the GameplayKit is an Object-Oriented framework that includes tools for designing games with reusability and functionality in mind. This framework is used for the player movement, physics, collisions, emitters and spawning to name a few.

- UIKit:

This is the framework that provides the basic infrastructure for iOS apps. The main app loop window and view architecture and event handling for input are all features controlled by UIKit.

Conclusions & Recommendations

Throughout the process of making this project there were many things that we liked, disliked, learned and things we would have done differently. Firstly, one of the things that we both enjoyed was using new technologies and a language that we had not used before. As we were creating iOS applications we had to use the Swift programming language. For the most part we found it both intuitive and easy to use. Furthermore, we had never before developed on any wearable tech before and we found this concept interesting. From our online research we

found a vast number of cases where gestures could be used to perform real world tasks, interface with UI and generally innovative applications of the technology. We enjoyed the fact that this project gave us the opportunity to learn about futuristic hardware and software and allowed us to play around with these concepts and ideas.

The objectives given to us inspired our research into Gesture-Based Technology and it brought the ever-growing use of this technology in everyday life to our attention. The first piece of hardware we worked with was the Myo Armband and we learned that it wasn't 100% accurate in the context of consecutive gestures. We created basic applications in MS Visual Studio to test the different gestures by changing the colour of the game objects with the various physical gestures pre-programmed in the Myo technology.

One thing in particular that we found to be difficult was the fact that before we embarked on this project there was no example of a similar game either online or on the App Store. This meant that we had a lot of trial and error involved in getting the connection working. Another series of issues we encountered were hardware issues related to the Apple Watch. These included problems such as the display going to sleep after a set time or when the Watch thinks it is not in use. The issue costing the most amount of time in the development was the with the transfer of data between hardware. We had to play around with the rate of data recorded by the watch so that the phone did not get bombarded with data and force the spaceship to stutter onscreen. We had to find a middle ground so that we could reduce the latency between the watch movement and the spaceship movement, whilst not overwhelming the connection.

Overall we thoroughly enjoyed making this project and we are glad that we chose this idea. Considering that it was not guaranteed that this implementation would be possible or even be a viable control mechanism in the end, we are extremely happy that we were able to create a link between the iPhone and paired watch that would allow live controls based on simple gestures. We feel that by doing this project we have gained invaluable experience and proved that these technologies are applicable to use in both gaming and gesture control.

Links / References

Gameplay:

Sprites: https://assetstore.unity.com/packages/templates/tutorials/into-the-space-2d-space-shooter-project-20749

Animations: https://www.brianadvent.com/spritekit-space-game-explosions/

Watch connectivity:

https://www.raywenderlich.com/3358-advanced-watchos/lessons/5

https://www.natashatherobot.com/watchkit-open-ios-app-from-watch/

https://developer.apple.com/documentation/watchconnectivity

Sending Data:

https://stackoverflow.com/questions/33200630/wcsession-sendmessagereplyhandler-error-code-7014-wcerrorcodedeliveryfailed

https://forums.developer.apple.com/thread/107831

https://stackoverflow.com/questions/32092243/global-variable-in-appdelegate-in-swift/32092335

https://stackoverflow.com/questions/42522499/access-variables-from-app-delegate-in-view-controller

Class Diagram:

https://github.com/yoshimkd/swift-auto-diagram

Apple Docs:

https://developer.apple.com/documentation