Final Report 2014
Paloxis

Sean O'Brien
20053411
Bsc(Hons) Entertainment Systems

Supervisor: Dr Kieran Murphy

13th March 2015

Table of Contents

1. Introduction

1.1 Summary

This research and development project was undertaken in an attempt to both further my own knowledge on Apple's new programming language Swift. While also taking Swift to the Apple's other game based libraries such as SpriteKit and SceneKit This was done to see what apple have implemented and the possibilities of this language for new developers.

1.2 Aims

- Create an example project in Swift to show its progress as a mobile language.
- Enhance my learning, by learning SceneKit, SpriteKit and Swift.
- Become informed about Swift and SceneKit, and how they work together in their current state.
- Explore the possibilities for control on mobile devices

1.3 Key Features

- New language – Use of Apples latest language.
- Motion Controls – Gyroscope and Accelerometer
- Practical learning – First use of SceneKit and SpriteKit

2. Revised Game Design Document

2.1 Game Design

This section covers the design of the project from a games design point of view, through the use of Baldwin's Game Design Document to outline the main aspects of game design.

2.1.1 Game Concept

A Flight simulator based on the World War II Spitfire airplane. Players will be brought into a sky bound arena, with the use of motion controls they will chase down and clear out the enemy pilots. Through the power of new technology their control is a key point to this game. Three dimensional movement allows for the freedom a player desires, in an effort to give the feel and look of flying without restrictions.

2.1.2 Feature Set

Internal

1. SceneKit - Use of Apple's latest 3D Shader library, built for modern devices and operating systems.
2. Swift iOS8 - Use of Apple's new Swift programming language giving you the latest and greatest experience.

External

1. Motion control - Allows the users of the game to feel the movement of their plane flying giving a better experience than standard controllers.
2. 3D Graphics - Allows for a more real experience and shows the user how it feels to sit behind the stick of one of these flying machines.

2.1.3 Genre

The Genre of this game is aircraft combat flight simulation; the aim of the project is to get the aircrafts moving as realistic as possible. Using iDevices built in gyroscopes and accelerometers the game aims to simulate the motion though movement making it a better simulation Due to the player interaction and control via body movements its possible to create the feeling of flight.

2.1.4 Target Audience

Mobile iOS devices, capable of running iOS 8.

2.1.5 Game Flow

The flow in this game is derived form the ability to fly in three dimensional space, this allows the game to be difficult for learning players due to the control system taking practice, while also allowing difficulty for the more experienced players in the form of advanced maneuvers due to the freedom of controls offered.

2.1.6 Look And Feel

Due to its nature, the look and feel will need to be as realistic as possible in the development time allowed. This creates a dilemma on which API to be used. The feel of the game should always be freedom of movement with the added element of fear due to the multi-directional approach of NPC's and other players. Because of this the API needs to keep up with the motion and be compatible with Swift and CoreMotion.

2.1.7 Project Scope

The World

This project hopes to create one world in which the game is played. This also leaves scope for more worlds to be added at later times.

The Player

A singularly controlled unit that is able to move around in three-dimensional space.

The Controls

Fully functional three-dimensional controls using the devices gyro and accelerometer to keep the player in the sky via their own movement. This movement style was chosen as it adds complexity to the controls and also another element of fun; as players have to physically fly their plane.

The Menu System

The menu system is made up of simple menus allowing for the set up of single player or multi-player games. The simpler the menus the easier the navigation, as good navigation relies on easy clear choices.

2.2 Game Play

2.2.1 Game Progression

The game will progress with the player's skill level as their control over the game will determine how hard or easy it is. Use of the motion controller will give players freedom of movement and this will allow for their skill at the controls to build most of the games difficulty, on top of this AI will have multiple levels of intelligence to keep the player on their toes.

2.2.2 Challenge Structure

The challenge structure will be based on risk and reward. Dangerous flying will yield easier kills with the risk of crashing but safe flying while keeping you from hitting the ground, will make you an easy target for the other players. The motions control system should add to make the challenge adequate to keep the casual game player happy.

The decision to add AI to the game was made for two reasons:

The addition of non-player characters means you can play solo to practice you skills in the game and come up with your own style of play without fear of ridicule.

The secondary reason for the AI is that it adds content to the game. Although simple AI can be predictable and easy to beat, Learning to beat them creates more of a challenge for the players to overcome.

2.2.3 Objectives

The objective is simple; for the main combat mode is to be the last person in the air and take out the most other planes. This allows for the need to engage in combat to win. Because just being in the air, might be a moral victory for the player and contributes towards their final score. If they don't engage in combat their score may still be too low to claim them victory.

This objective requires three things. Firstly a scoring system, this gives worth to player's achievements. Winning is the main goal but knowing how much you win by adds to the overall experience. The second is unfortunately games that are last man standing can take a long time so a way to increase the chances of play concluding is necessary. This can be done through the use of a game timer or by making the area of play smaller as time progresses. Lastly to achieve these objectives, players are needed and therefore the game requires a multiplayer element.

2.2.4 Play Flow

The play flow is all about adaptability; players are expected to adjust to situations that may arise during play. Due to the nature of the way in which the game flows, this can be hard to achieve, getting out of the realm of boring but keeping interest without making it too difficult is a major task (Baron, 2012). I believe this is achieved by the freedom in the controls; players will be able to make it hard both for them and others through the multiplayer elements. Although the game may always be in the same area the added human elements make for a different game every time thus creating game flow though the multiplayer elements.

2.3 Mechanics

2.3.1 Physics

The Physics will be made to look as realistic as possible in the time allotted. Due to the nature of the game physics this means that although made to look real, the mathematics may not be the exact calculation for that specific scenario. As a result real world mimicking physics will be the physics in this project.
Key Points:

- Plane crashing into terrain.
- Plane colliding with another plane.
- Bullets crippling flight ability (for example how missing part of the plane will affect your flying)

2.3.2 Movement

The Movement is going to involve using the accelerometer and gyroscope that all iDevices have to act as a motion controller for the movement of the spitfires. The control over three dimensions will increase the difficulty as well as the fun factor due to having to get up and move around.

2.3.3 Types of movement

Motion Controls

Through use of:

Gyroscope- Uses earths gravity to determine the rotation of a device in relation to a fixed axis.(Goodrich, 2013)

Accelerometer- Detects sudden impulses of movement and this the gives you acceleration and deceleration.(Goodrich, 2013)

These two hardware devices allow the player to control the game by rotating the phone in different directions, allowing for three dimensional movement via physical interaction.

Heads Up Display (HUD)

This is another type of control that relies heavily on screen space, as controls are mounted as an overlay over the game. On a phone for example you have touch screen buttons and sliders to control rotation speed and general movement. I aim to save on screen space by using motion controls instead of many on screen buttons and switches. However I still plan on using some on screen controls for firing weaponry and controlling speed.

Techniques required for motion controls to work correctly

Dampening- A technique used to degrade movement over time rather then instantly snapping to a direction. This will allow for realistic flying, as the plane will move smoothly through the air.

Calibration- A technique for detecting the devices sensitivity, Simply lie the phone flat and then use this as the phones base or zero value so that the movement is not overly sensitive. This makes the device much easier to control as the highly sensitive equipment is set to only change at positions once over a certain threshold.

2.3.4 Objects

The objects all come from a DAE file. This is a scene file that scene kit uses to incorporate 3D models into the environment. The Following models will be included:

- Spitfire Airplane.
- Farm House
- Trees

The scenes terrain will be made of flat ground for the first prototype. This leaves room to expand for the finished product but cuts down on development time for the prototype. While I will be putting the DAE file together myself, the models used will be sourced online and detailed later in the report.

2.3.5 Actions

The player will have the ability to speed up and slow down the plane as well as control its flight patterns. They will also have control over two forward mounted turrets. The actions have been limited to focus on functionality and more actions can be added at a later date.

2.3.6 UI

The UI will be kept as simple as possible. Refer to in the game art section for a sample screen, both for in game and in the menu system. Use of Sprite kit as an overlay will be an effort to both keeps with the use of Apple's own libraries and to keep the game free of clutter and non-necessary code.

2.3.7 Combat

The Combat is that of a world war dogfight; players are given a limited area to fight over and they must kill the other. The front mounted turrets on each plane create need to get behind the other or risk flying straight at them.

2.4 Screen Flow

Screen Flow Chart

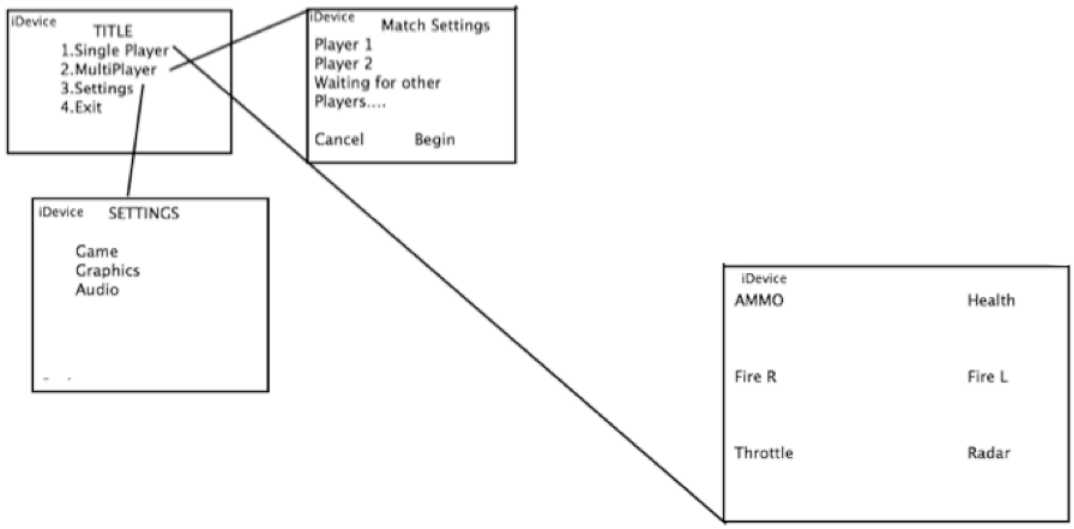

Figure 1.1 Proposed screen flow diagram

2.4.1 Screen Descriptions

The first screen contains the title and main menu for game options. The next screen is a lobby screen; this is for the intent of waiting for other players to connect. The third screen is the settings screen and is styled like the main menu with a back button to get back to the main menu. The fourth screen is the game; this is a simple overlay with some on screen controls.

2.4.2 Game Options

- Single Player
- Multiplayer
- Settings (also available in game)
  - Audio
  - Display
- Quit

2.5 Narrative And Setting

2.5.1 Game World

Country setting in France during the Second World War, the battle takes place in the air above the quiet fields of France. Setting is dusk and light is starting to fade so visibility is low. The ground is that of grass and trees.

2.6 Interface

2.6.1 The HUD

The HUD, or rather Heads Up Display, is going to be kept very simple. Two onscreen buttons for left and right guns, two onscreen dynamic bars to show health and ammunition and finally one slider for the planes throttle.

2.6.2 Rendering System

There are three options for rendering each has its advantages and disadvantages.

Apple's Metal - Apple's new rendering language, built solely for their hardware. It sits as a smaller layer between the GPU and the running game but only on the new A7 chipset. (Apple, 2014)

OpenGL ES - This is not available in Swift yet and there are issues with enum types for the OpenGL swift to Objective-C Bridge. This is mainly INT32 non-convertible to ENUM<Type> that causes issues when trying to use a lot of functions in OpenGL ES. OpenGL seems to have been replaced by Metal or rather is due to be replaced. (Eyles, 2014)

Scene Kit - Sitting above OpenGL ES Scene Kit allows the use of OpenGL with Swift giving you all the tools for meshes cameras basic physics and scenes. This means it is perfect for using Swift with older devices that do not support metal.(Apple, 2014)

Reasons for choosing Scene Kit

There were three factors, which finally lead to the decision to use SceneKit.
As detailed in report one, OpenGL ES is still not available for Swift and time wise the project needed to start.
Metal Apple's new shader language is only available on newer machine running the A7 this is also an issue as I want to target older devices.
And Finally Scene kit is also a new enough development only released in iOS 7 therefore keeping up to date with modern technology and software fits in with the project.

2.6.3 Cameras

One camera per player, one in first person mode that can move into third person mode. The first person will be just a cross hairs on the screen while the third person will be a full render of the plane and will follow the player with an over the shoulder effect.

2.6.4 Audio

Audio will be ambient country sounds; plane engine noises and gunfire all of which will be free open source sound pack found online.

2.7 Artificial Intelligence

2.7.1 Enemy AI

I plan to have three types of AI:

1. Simple AI - this AI will fly in circles until the player is close then go directly for the player. Idle NPC with Player detection.
2. Medium AI - this AI will fly in paths and will try to avoid the player unless they are behind in which case they will engage. Way pointing NPC with player detection and evasion.
3. Difficult AI - These will chase the player always staying behind the player unless they are being chased in which case they will fly low to the ground in an effort to shake the player. Player Detection and stalking.

2.8 Technology Requirements

2.8.1 Target Hardware

The Target Hardware for this project is all iDevices capable of using iOS8.
The List of iDevices I will be focusing on:

- iPod 5th generation
- iPhone 6/ iPhone 6+
- iPad 3rd generation

2.8.2 Development hardware and software

Development will be done on a MacBook pro early 2011 using xCode 6. The Management of the project will be kept track of using the free online service Trello and the UML graphs for the project will be done with Umlet editor. This is a free software. Models will be generated using Blenders DAE export function, which of these will be decided by ease of use and integration with Rendering library.

For test devices I have an iPod 5th, this Device and Software were chosen for their simplicity and mostly free open source software.

2.8.3 Development procedure and standards

The project will be sticking with the Apple Swift standards for code and for procedures I will be using agile development through scrum, this is breaking down tasks into there simplest forms and rating them with points of how difficult they will be in relation to time, e.g. one point is equal to roughly ten minutes.

2.9 Game Art

2.9.1 Models

The decision was made to use open source 3D model files and to modify them to suit the needs of the project to cut down on development time. Creating a DAE in Blender to act as my scene, my models are as follows.

Fig 2.9.1 Spitfire DAE model (Jasonowen, 2014)

Fig 2.9.3 Rock Low-Poly (Tyrosmith, 2013)

Fig 2.9.4 Wooden House (Kaos2d, 2015)

2.10 What Already Exists

RC Plane 2 - This is a flight simulator with on screen controls. The gear stick for the throttle is a piece of control I would like to mimic in this project. (Brozolo, 2014)

My Paper Plane 3 - This has the motion controls the project would need to have. The only thing missing from this app is the rotation controls as it only moves in one direction whereas the project needs to move in 3D not down a liner path. (Wavecade, 2014)

Historical Landings - This application has the look and feel this project aims to achieve and has controls almost perfect to what the project is aimed towards. Particular attention to be paid to the use of calibration before to set the axis based on the users preferred sitting position. (RORTOS SRL, 2014)

2.11 Potential issues

2.11.1 Software Issues

Swift is Apple's newest language and has some issues. Most of the issues seem to be around integration with Objective-C and its many libraries.

Some examples:
Swift Usability - While Apple claims it's easy to use for new developers; this is in fact not the case. For simple stuff, it's easy to write and read but it is still riddled with Apple's own way of doing things. For an Objective-C developer this is a common every day thing, so Swift in turn is simple and easy to use as you are already in the mind frame of apple's ways. For a new developer this becomes a problem much like it was with Objective-C. (Napolitano, 2014)

OpenGL ES - According to Apple if you set up a Swift OpenGL ES Project it is still in development.

```
/*
NOTE: The OpenGL ES option is still in progress for Swift.

import GLKit
import OpenGLES

/*
#define BUFFER_OFFSET(i) ((char *)NULL + (i))

// Uniform index.
enum
{
UNIFORM_MODELVIEWPROJECTION_MATRIX,
UNIFORM_NORMAL_MATRIX,
NUM_UNIFORMS
};
GLint uniforms[NUM_UNIFORMS];

// Attribute index.
enum
{
ATTRIB_VERTEX,
ATTRIB_NORMAL,
NUM_ATTRIBUTES
};
*/
```

New Project-> Swift->OpenGL->GameViewController.

Figure 1.5 Screen shot of a basic apple project swift with OpenGL ES (Apple, 2014)

The issues around this project have grown since starting.
Scene kit - Uses a command based system this is a new way of coding for me so I am having trouble with this as well as the Swift being a new programming language. Learning how to use both swift and the SceneKit library have been a major part in the development of this project, the render loop in SceneKit for example requires you to delegate an object as the scenes render delegate but cannot be the Game View Controller.

2.11.2 *Hardware Issues*

The hardware issues with this project is the requirements on iOS8 cut out all usability on older devices such as the iPhone 4 which enable to use iOS8 but not only this the newer iPhone 4s can run iOS but it has trouble running to heavy a load, while on this operation system.

3. Technologies Used

3.1 Swift

3.1.1 Apple Documentation

According to Apple. (Apple, 2015)
"Swift is an innovative new programming language for Cocoa and Cocoa Touch"

Apple's new programming language; they claim that its new way of coding is Objective-C without the C, integrating into Cocoa and Cocoa Touch so its perfect for game applications and boasts fast and powerful performance allowing fast games. (Apple, 2015)

3.1.2 Developer Reviews

On Release

When Apple "Wow'ed " the crowd at last year's development conference with the release of iOS 8. Apple have been using Objective-C since the 1980's so the language is pretty old in computing terms, keeping with the times adding new libraries to the huge suite of libraries. This is why Apple's release of Swift really took the spotlight at the conference. Apple stating that its like Objective-C without the C is the main point the new language is taking a step in the right direction. Swift looks and feels simple sticking to more conventional standards without loosing the Apple feel. (Metz, R 2014)

Syntax

Swift emphasizes on easy to read code is in high demand among today's developers, working towards fighting off bad programming habits through type safe interfaces and the removal of semicolons. (Chester, B 2014) But the biggest advantage to Swift is the use of Objective-C and the Frameworks integration of its predecessor.
(Siracusa, 2014)

Type Safe

When construction variables a programmer will set its type int x = 1; this makes sure that this data is always an Integer or String or what ever it has been set to with swift you can set var x = 1 and it will except this as an Integer, however later in the code you may what to change its data type for example x = "N/A" this now has accepted a string. For this method to work you must never have a null or nil object so the compiler returning an error caters for this.
Not to say you are forced to use the Type Safe style you can set a flag to declare you are aware this object can be null or nil and also set a data type for certain variables for example var x : int! this give the developer the option to set the type of a variable so I will only accept that type and also applies a compiler flag so the variable can be nil or null. (Chester, B 2014)

Removal of Semicolons

This is the simplest area of swifts modern style, simply put you are not required to put semicolons at the end of every line but the compiler will ignore them if they are there so as not to introduce errors. (Chester, B 2014)

Frameworks integration

Simply put this just means Objective-C and Swift work together in the same file a great selling point for a new language, as it will be able to back track through the use of its predecessors libraries and make sure there are no blocks in the code as you can revert for a class or method that is missing from Swift. (Siracusa, 2014)

Swift 1.1

The release of Swift apple claims 2.6x faster then objective-c and 8.4x faster than Python 2.7, but on a closer look by Tyrone an iOS developer in the states. He discovered just how slow swift was, while Apple's claim was its speed Tyrone did a test using the same code just in different styles of Swift, Objective-C, Swift Like Objective-C and Ruby Motion to find Swift with all its claims on speed was in fact the slowest of the four.

Style	Optimization	Time
Objective-C	-Os	0.09
Objective-C-like Swift	-Onone	0.29
Swift	-O	1.42
Ruby Motion	N/A	0.21

The Above chart was the result of Tyrone's tests, Swift clocking in at 1.42 seconds on a performance test on the iPod 5th generation. Being the iPod is the slowest device for testing on iOS 8 that is at least functional.
(Tyrone, 2014)

Swift 1.2

Apple's Response

"… produce binaries that run considerably faster, and new optimizations deliver even better Release build performance."

	Swift 1.1	Swift 1.2
ObjC	0.09	0.06
Swift	1.42	0.35
ObjC-like Swift	0.29	0.13
RubyMotion	0.21	0.21

Performance on the same piece of code shows a great improvement in speed. (Tyrone, 2015)

Swift Vs. C

While C is known for is speed due to its low level when it comes to sorting Swift seems to have it beat, in a test done by Jesse Squires an iOS developer also from the states, Swift out preforms C in the standard library Sort() method for a basic array of a random amount of integers.
Using this array as a benchmark she tested Swift against C resulting in the following chart.

T = 20 *N* = 100,000 Release	Std lib sort	Quick sort O(n log n)	Heap sort O(n log n)	Insertion sort O(n²)	Selection sort O(n²)
C -O3	0.020853 s (± 0.000191)	0.011243 s (± 0.000042)	0.014909 s (± 0.000360)	7.575798 s (± 0.016685)	6.981794 s (± 0.004399)
Swift -O	0.014114 s (± 0.000413)	0.016745 s (± 0.000202)	0.042325 s (± 0.003881)	33.819305 s (± 0.120471)	25.255743 s (± 0.084152)
Difference	1.48x	-1.48x	-2.84x	-4.46x	-3.62x

(Squires, 2014)

Personal Observation

Show swift is marginally faster than C in a standard sort, although falling behind in the other algorithms its still shows promise as a language in regards to speed after halving the time on Tyrone's performance test on the recent update Swift is a language to watch for the future.

3.2 SceneKit

3.2.1 Apple Documentation

"Scene Kit is a 3D-rendering Objective-C framework that combines a high-performance rendering engine with a high-level, descriptive API. Scene Kit supports the import, manipulation, and rendering of 3D assets without requiring the exact steps to render a scene the way OpenGL does." (Apple, 2014)

Apple's SceneKit sitting under the Cocos library and above OpenGL gives you more access to features at its lower level, while keeping some of the high level aspects of Cocos. Integrating with many other technologies such as core animation and GL Kit to give you more options. (Apple, 2014)

Loading Assets

Supporting the import of DAE file, which are creatable by many modeling tools such as Maya, 3ds max and Blender. Once in the scene all the models properties are assessable such as geographic location and shaders. (Apple, 2014)

Manipulation and Rendering

Giving you access to the Render delegate allows for use of your own Open GL code, while not used in this project it also gives you access to the render loop which the project makes use of to control state machines and ongoing game logic such as health, ammo and score. (Apple, 2014)

The three levels of SceneKit's rendering:
SCNView – Module in the interface builder allowing for easy integration.
SCNLayer – Holds the 3D scene and controls the Layer Tree.
SCNRender – Access to rendered for Open GL code and Renderer loop
(Apple, 2014)

Scene Graph

The graph depicts the engine it self and shows in detail the architecture of SceneKit.

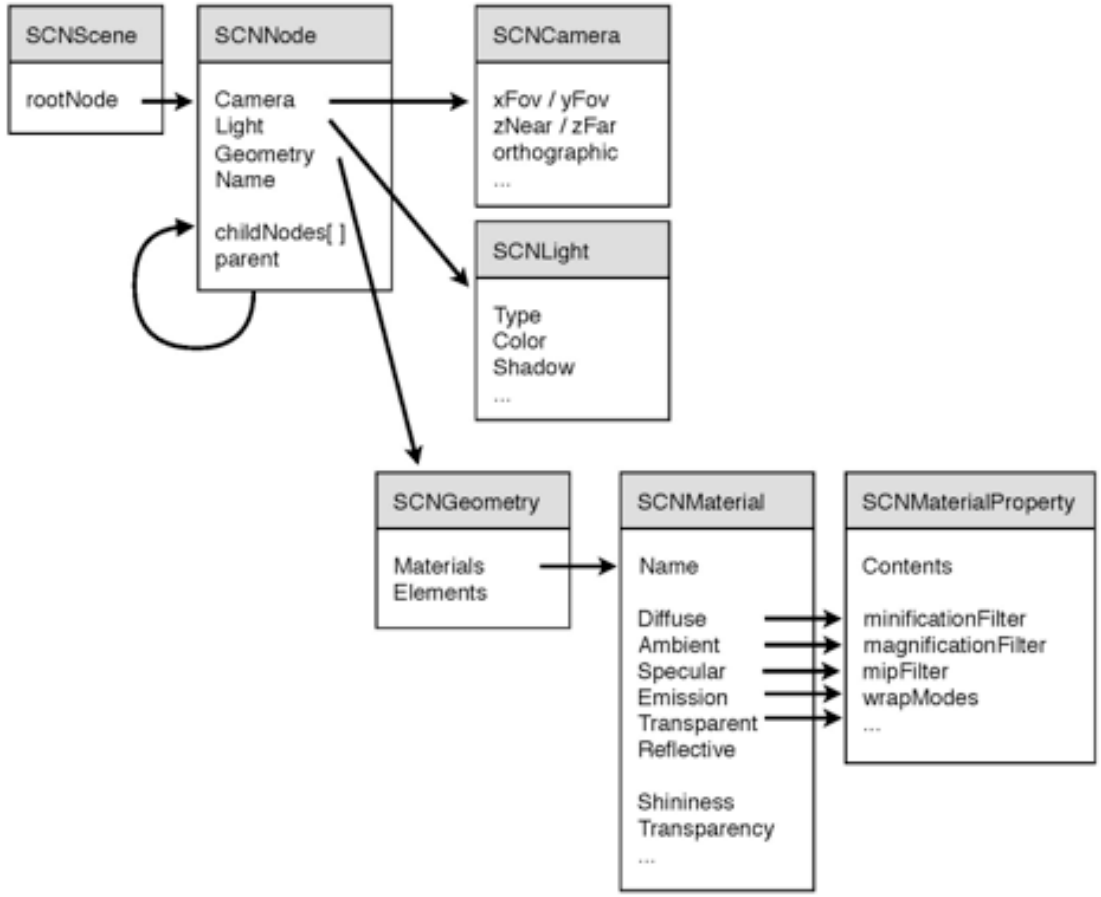

Fig. 3.2.1.1 Apple Scene Graph For SceneKit (Apple, 2014)

3.2.2 Developer Reviews

On Release

Scene kit makes it easier to produce 3D games for iOS through its integration of many advanced technologies such as:

- Built in physics engine
- Built in particle generator
- Built in Action Script

One of the biggest draws for SceneKit is its simplicity compared to libraries such as OpenGL, opening doors for casual game developers who do not want to use big clucky middle ware as the base of there game. (Cohen, 2014)

According to Blockcom. (Blocksom, 2014)
"SceneKit creates a scene graph, a more abstract way of working with 3D shapes and rendering than OpenGL, sort of like dealing with web pages instead of strings."

They explain SceneKit rather well, as it differs from Open GL in a big way. Creating scenes and controlling nodes within a scene to create games rather the doing everything from base level and writing large shader scripts so things draw correctly.

The Problem With SceneKit

SceneKit does a lot for you, COLLADA files look great and are easy to get into the scene and SceneKit optimizes the graphics pipeline a lot making time to focus on other areas. However Hartmut explains in his blog the control issues with SceneKit some of which this project will encounter. The main issue he found is:

"A huge problem with SceneKit is that it has no user accessible state. Everything is asynchronous and there is no access to the traversal stages and to the transformation stack. This also means one cannot get frames rendered in sync with other things that might happen." (Hartmut, 2015)

This is something I will need to keep an eye on for the integration with CoreMotion, as controls need to be synchronized to avoid problems with motion delay.

3.3 CoreMotion

CoreMotion simply provides data from the gyroscope and accelerometer that all iDevices have in the hardware. Allow this data in a clean simple library gives developers options for control, that are under used in favor of simple swipes and drags. Accelerometer and Gyroscope data provides the 3D movement required for this project. (Cook, 2014)

3.4 Possible Technical Issues During Development

During my research I have outline three areas of concern:

3.4.1 SceneKit Library and Documentation

Being that I have never used SceneKit before I have concerns about learning the library. This is due to a lack of information on the web from doing my research the tutorials for SceneKit are minimal. Everything needed is in the developers documentation provided by Apple. But this shows what each class and method does with very little detail on implementation of each class and or method.

3.4.2 CoreMotion Data

Different phones have different sensitivity in they're gyroscopes and accelerometer, this brings in the issue of experience for different devices, being it may be harder or easier to control.

3.4.3 Swift Performance

Based on research done in section 3.1 Swift, I have outlined there may be a slight difference in the speed of Swift and Objective-C. This being quite small in the latest release.

4. Development

4.1 Iterations

4.1.1 Iteration One Setup

On starting the project I had little to no knowledge of how SceneKit worked and how development work is carried out on the native Apple 3D Library. So for the first iteration I took time to familiarize myself with the Library and how it handles models, physics and input.

Starting with a basic scene with a single model of a Starship, I first looked into getting in a new model this required getting familiar with Blender, a 3D modeling tool used for building COLLADA scene files which is where SceneKit pulls models from. Once in the correct folder the files are automatically picked up in the "art.scnassets" collection folder. Once in here you can add pointers to textures for the various parts in the scene.

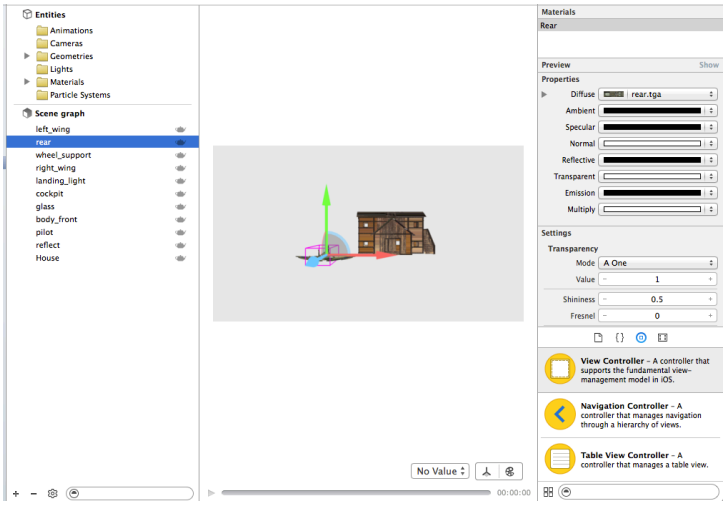

Fig 4.1.1.1 xCodes COLLADA file xCode version 6.1 (6A1052d)

Simple access from the code then using.

```
scene.rootNode.childNodeWithName("pilot", recursively: true)!
```

This pulls in the model with the name "pilot" into the code for use adding it to the scenes root node. This provides the project with easy use of models and is one of the main benefits of SceneKit.

4.1.2 Iteration Two Control

"Throughout the history of video games, one of the most important aspects of console video games is the controller. This piece of hardware is the one with which the players interact the most and is by far the most memorable component." (Lu, 2003)

Throughout the course of this project the above quote describes the most important part of the control aspect in games such as the motion control in this project; therefore I thought the first aspect of the game required for the functionality was the controls.

On the controls I went with the option of motion control. In iOS this required the use of CoreMotion, this allows access to the gyroscope and accelerometer; this gives access to the rotational data through the gyroscope and acceleration in given XYZ directions.

With access to the rotational data I began working on the rotation for the Spitfire Plain I had imported, using the `startGyroUpdatesToQueue` to produce the rotation quadrants with use of `SCNAction`, rotate by X is the `SCNAction` used to rotate based on the gyroscopes rotational data returned.

```
self.ship playerShip.runAction(SCNAction rotateByX(0.0,y:0.0,
z: accelX, duration: 1))
```

Based on this line above, we are assigning an action to the playerShip SCNNode causes the node to rotate based on the action because of how the `startGyroUpdatesToQueue` is set up. A queue is formed for the gyroscope and they are preformed in a linier manner causing realistic movement based on player movements.

Once I had the rotation set up the movement in the facing direction was the next issue, this I decided would be easier to deal with if I set up a controlNode.

Creating a hierarchy in the Ship class to deal with the different types of movement separately.

Ship
- ➔ Control
  - playerShip
    - Directional Node
    - Ship Part One
    - Ship Part Two
    - Ship Part Three
    - Etc.

Using this hierarchy I can control certain parts of the physics without interfering with the others, basing the playerShip for the rotations gives the Directional Node the correct position just in front of the ships nose. Using this Node I can get a directional vector between the ship.control.position and the ship.control.playerShip.directionalNode.position giving me the distance between the two in the direction towards the DirectionalNode which I can then base the ships velocity on a combination of the facing direction, the distance between the two and the effect of gravity.

4.1.3 Iteration Three UI

For this iteration I focus on the on screen controls, such as:

- Thrust
- Bullet Fire Button
- Bullet Overheat Bar
- Health Bar

Using the SpriteKit overlay feature I can create a transparent SpriteKit scene to put my UI in over the main game.

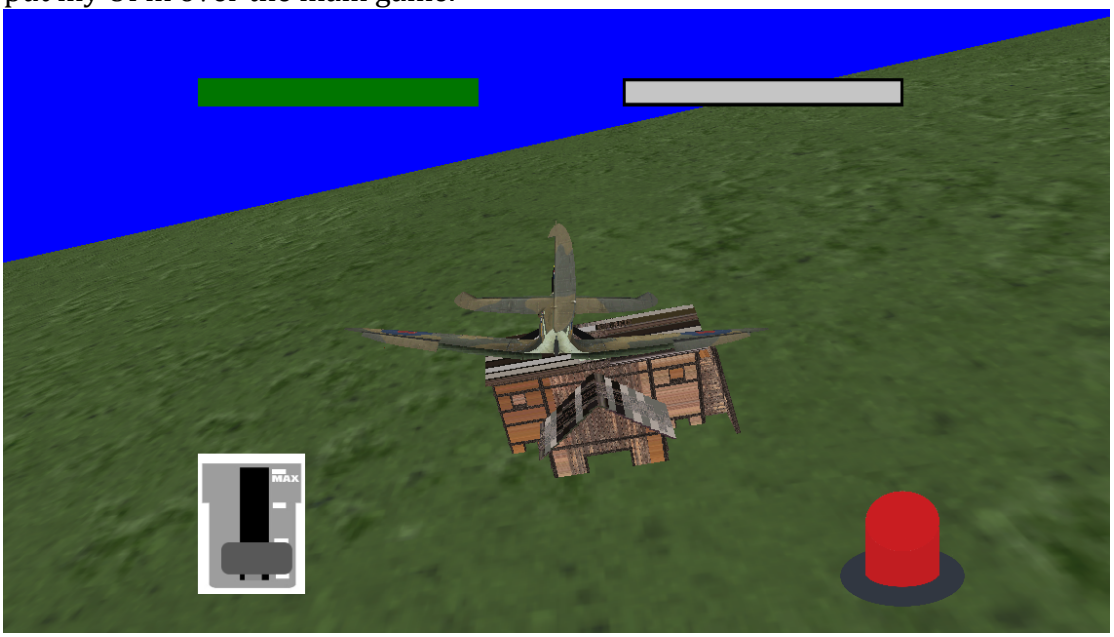

Fig 4.1.3.1 In-game UI.

Health Bar

Consists of two sprites layered on top of one another with a scaling float 0.0f-1.0f for the width of the green bar. Adding the method "func decreaseHealth(amount: CGFloat)" to decrease the health by varying amounts in the main game file.

Ammo Bar

Consists of two sprites layered on top of one another with a scaling float 0.0f-1.0f for the width of the yellow bar, updating the width depending on whether the red button has been or is being pushed. Once it hits max it stops firing bullets.

Thrust Control

Through use of the touchesBegan, touchesMoved and touchesEnded, use of methods built into SpriteKit I was able to detect when the ShiftStick sprite has been clicked. Once selected moving the ShiftStick with the constraints of only moving it in the Y direction and in the bounds of the GearBox sprite is all done in the touchesMoved function.

Red Button

The red button simply sets a Boolean, stating that the scene is capable of firing bullets the functionality changes the sprite in touchesBegan and changes it back in touchesEnded. Bullet firing and object handling is done in the main game class.

4.1.4 Iteration Four Scenery

For the scenery I build up a Blender file using models I obtained from the net detailed in the Game Design Document.

Once the models were in the blender file I build the scene in xCode's COLLADA interface builder as part of the art.scnassets folder.

4.1.5 Iteration Five Collisions

Simple collisions as all objects are simple treated as simple spheres and planes. This allows for point-to-point collisions or in the case of the terrain, simple Y value checks of the plane and bullets.

4.1.6 Iteration Six Game Play

Game play required a few features to be implemented; these include a simple Victory and Game Over screen to help navigate back to the main page after game completion or loss.

Score count for the amount of targets left and the health bar dictate how the game ends. While the bullet overload system keeps from to many bullets being created.

4.2 The Code

4.2.1 Source Control

For source control I used a Bitbucket private repo and Source Tree as a GUI client for access. As I was working on my own source control seems less important however I found it was even more important as I was the only one committing to the project I found myself needing to work with Source Tree more and more to keep separate features from interfering with one another. For example I had six branches and a master branch.

Master

This contained developed working code only; so on the completion of iteration, I would merge that branch into master to move the project forward. This gave the option to work on multiple iterations at once.

Iteration branches

These were simply branches to isolate the work for individual iteration. Merging in master as master was updated to keep the code in sync to ensure that each branch could be merged into master at any point without issue.

Through using Git in this way I was able to hop between iteration pretty easily and revert them when needed. Git improved development time and organization of the project through control and ease of use.

4.2.2 Main Functionality

Control

Control has been talked about a lot in this project so this section will provide the code to accompany the discussion in 4.1.2 Iteration Two.

```
func controlLoad() {
    motionManager = CMMotionManager()
    motionManager.accelerometerUpdateInterval = 0.3
    motionManager.gyroUpdateInterval = 0.3
    motionManager.startGyroUpdatesToQueue(NSOperationQueue.currentQueue()) {
        (gyroData, error) in
        let acceleration = gyroData.rotationRate
        let accelX = CGFloat(acceleration.x)
        let accelY = CGFloat(acceleration.y)

        if(true){
            self.ship.playerShip.runAction(SCNAction.rotateByX(0.0, y: 0.0, z: accelX, duration: 1))
            var velocity : SCNVector3 = self.ship.control.convertPosition(self.ship.directionalNode.position, fromNode:
                self.ship.playerShip)
            //NSLog("%f",velocity.y)
            velocity = SCNVector3(x: velocity.x * 4 * Float(self.overlay.gearSpeed), y: velocity.y * 4 * Float(self.overlay.gearSpeed), z:
                velocity.z * 4 * Float(self.overlay.gearSpeed))
            self.ship.control.physicsBody?.velocity = velocity
        }
        if(true){
            self.ship.playerShip.runAction(SCNAction.rotateByX(accelY, y: 0.0, z: 0.0, duration: 1))
        }
    }
}
```

Fig 4.2.2.1 Control Code

Physics

I have added rudimentary physics to the scene both using the Physics Engine provided by SceneKit and through my own calculations.

```
func physicsLoad() {
    //scene.physicsWorld.gravity = SCNVector3Make(0.0, -9.18, 0.0)
    ship.control.physicsBody = SCNPhysicsBody(type: SCNPhysicsBodyType.Dynamic, shape: ship.body_front.physicsBody?.physicsShape )
    ship.playerShip.physicsBody?.restitution = 0.9


}
```

Fig 4.2.2.2 Example SceneKit Physics

```
if self.ship.control.presentationNode().position.y < 0.0{
    overlay.decreaseHealth(1.0)
    gameOver()
}
for b in bulletPool {
    if b.presentationNode().position.y < 0.0 {
        NSLog("Bellow")
        b.removeFromParentNode()
        bulletPool.removeObject(b)
    }
}
```

Fig 4.2.2.3 Example My Own Simple Physics Code

4.2.3 Use Of Technologies

SceneKit

The most complex part to SceneKit was deciding how to use it, it has two ways of processing in-game features such as animation, physics and models.

SCNAction

This is the first way, this is done by running an action on a SCNNode at a preset condition for example onTap runs an SCNAction for the camera allowing for camera control if turned on.

Renderer

This is the other way to provide game logic on a frame-by-frame basis, being a little harder to setup but providing the project with a lot more control. It works by setting up a class as a delegate of the SceneView this class needs to be set as an NSObject and a SCNSceneRendererDelegate this allows the class to be used in the GameViewController as a variable and can then be assigned as a delegate to the scene view.

```
func renderer(aRenderer: SCNSceneRenderer!, updateAtTime time:
NSTimeInterval)
```

The Above method can then be used in the class as an addition to the in build renderer rather then an override allowing the project to add logic to the scene on a frame by frame basis without interfering with the SceneViews render process.

SpriteKit

This is used as an overlay in the `SCNScene` creating a transparent layer of SpriteKit to sit on top of the Scene. After setting up the Render Delegate for the Scene this was the same or rather a similar process setting an NSObject to a Class so as to delegate it as an SKView for the overly. After which it acts just like a normal SpriteKit View.

CoreMotion

CoreMotion returns data; it was quite easy to work with as its function for rotation and management of movement are well documented. Setting up the motion manager in the main game file allowed for simple action commands for the movement.

4.3 Methodology

4.3.1 Agile

Agile was chosen due to the uncertainty of the technology being used, being new to Swift, SpriteKit and SceneKit. I was unsure as to the problems the project might bring up meaning the waterfall methodology was not really an option. Agile allows for unpredictability and allows for the project to adjust based on the current state of the software and any known issues that have occurred.

Managing Agile is done using a board style containing cards, a card is a task and that task moves from board to board depending on its state. Cards can be changed deleted or altered at any time during the project but once they're on the completed task board they're done.

Trello

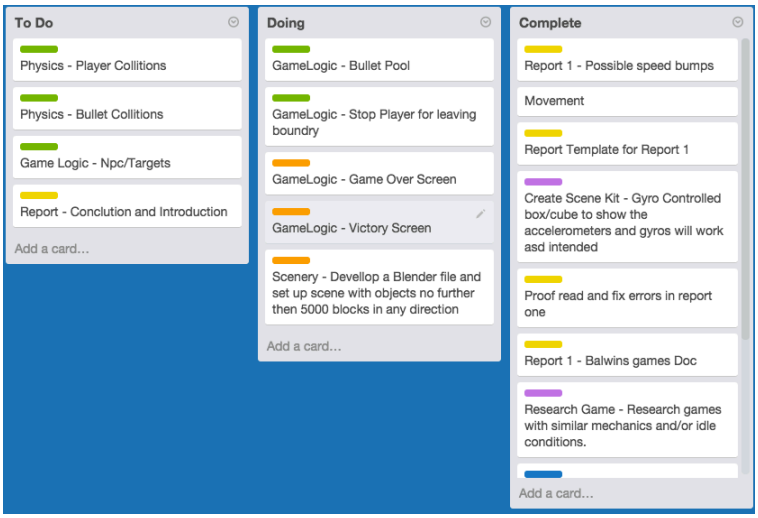

Fig 4.3.2.1 Trello Board In-Progress

Use of Trello as a project management tool links into the agile development allowing for breaking down of the tasks into smaller parts using cards and boards.

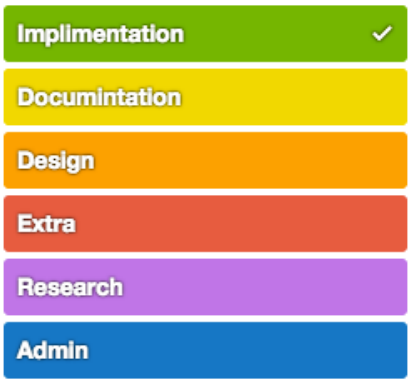

Fig 4.3.2.2 Trello Board Labels

Also keeping inline with agile Trello allows for labeling of tasks, which divides up the work evenly among the different areas of the project.

5. Conclusions

5.1 Project Completion

The end of this project provides a simple game. This game uses new technology to explore Apple's native libraries and how they work with on another. Creating a template with many different possibilities.

5.2 Were To Go From Here

There are multiple directions this project could take which include.

Enhance Physics

The physics in the project was kept very simple, because of this there is room to enhance this giving a better experience for the user.

Multiplayer

Modern day gaming is mostly on the web, so a direction this project could take is to implement a multiplayer connection so as to connect many players in one Scene.

The Hidden Console

Part of the original premise was the use of the Apple TV this caused issues due to time constraints. The extension of the game onto a secondary monitor would very much enhance the game play.

5.3 Final Conclusions

Working on this project was a great learning experience for me and being a chance to look at three areas of programming I was looking to explore.

Swift is still in its early stages I found it extremely easy to use and it certainly sped up my development time, but due to it being a new language the amount of research that when into small areas of the language slowed the whole process down.

SceneKit is an extremely useful tool but I don't see it being used in the near future for and high end applications as I found it hard to grasp and struggled to get some of the features workin. Although it did a lot of the work for you in the rendering area I found getting it to work with game mechanics an up hill struggle.

SpriteKit as I only used a small portion of the library as an overlay for SceneKit but for the most part it is very straightforward and similarly style to SceneKit in many areas.

Motion controls in games are extremely interesting and I would like to certainly look more into this are and develop better controls for this game.

6. References

Apple (2014) Metal For Developers Available at: https://developer.apple.com/metal/ [Accessed 4th November 2014].

Apple (2014) Introduction to Scene Kit Available at: https://developer.apple.com/library/mac/documentation/3DDrawing/Conceptual/SceneKit_PG/Introduction/Introduction.html [Accessed 4th November 2014].

Apple (2014) Core Motion Available at : https://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/Art/gyro_rotation_2x.png [Accessed 4th December 2014].
Apple (2015) Swift Avalable at: https://developer.apple.com/swift/ [Accessed 4th December 2014].

Baldwin, J(2005) Game Design Document Outline Available at: https://www.kth.se/social/upload/527a0f6df276544fc10d63a6/BaldwinGameDesignDocumentTemplate.doc [Accessed 5th November 2014].

Barron, S. (2012) Cognitive Flow: The Psychology of Great Game Design Available at: http://www.gamasutra.com/view/feature/166972/cognitive_flow_the_psychology_of_.php [Accessed 22 October 2014].

Blocksom, J(2014) SceneKit in Mountain Lion Available at: https://www.bignerdranch.com/blog/scenekit-in-mountain-lion/ [Accessed 11th April 2015].

Brozolo, O(2014)RC Planes 2 Available at: https://itunes.apple.com/ie/app/rc-plane-2/id442082328?mt=8 [Accessed 5th November 2014].

Chester, B (2014) Apple's Swift and What it Means for Developers and Users Available at: http://www.anandtech.com/show/8140/apples-swift-and-what-it-means-for-developers-and-users[Accessed 26th March 2015]

Cohen, P (2014) SceneKit in iOS 8: Explained Available at: http://www.imore.com/ios-8-scenekit-explained [Accessed 11th April 2015]

Cook, N(2014) CMDeviceMotion Available at: http://nshipster.com/cmdevicemotion/ [Accessed 11th April 2015]

Engel, A (1940) Another bunker at La Berlière Available at:http://users.skynet.be/albertengel/France/bunkers.htm [Accessed 3rd November 2014].

Eyles, F (2014) Will Apple's Metal Framework Fully Replace OpenGL for iOS Development? Available at: http://blog.clearbridgemobile.com/post/88479379567/will-apples-metal-framework-fully-replace-opengl-for [Accessed 4th November 2014].

Goodrich, R. (2013) Accelerometer vs. Gyroscope: What's the Difference? Available at: http://www.livescience.com/40103-accelerometer-vs-gyroscope.html [Accessed 29th October 2014].

Hartmut (2015) SceneKit - Good looking and useless. Availible at: http://www.technotecture.com/blog/scenekit-is-not-a-scene-graph [Accessed 11th April 2015]

Jasonowen (2014)UK WW 2 Spitfire MK 1 Available at: http://tf3dm.com/3d-model/uk-ww-2-spitfire-mk-1-64128.html[Accessed 5th November]

Kaos3d (2014) Wooden House Available at: http://tf3dm.com/3d-model/woodhouse-32132.html [Accessed 10th March 2015]

Lu, W (2003) Evolution of Video Game Controllers Available at: http://web.stanford.edu/group/htgg/sts145papers/wlu_2003_1.pdf [Accessed 14th April 2014].

Metz, R (2014)Apple Seeks a Swift Way to Lure More Developers About Available at: http://www.technologyreview.com/news/527821/apple-seeks-a-swift-way-to-lure-more-developers/ [Accessed 22nd March 2015].

Napolitano, S (2014) The Problem With Swift No One is Talking About Available at: http://www.huffingtonpost.com/sam-napolitano/the-problem-with-swift-no_b_5459902.html [Accessed 3rd November 2014].

RORTOS SRL (2014) Historical Landings Available at: https://itunes.apple.com/ie/app/historical-landings/id638728858?mt=8 [Accessed 5th November 2014].

Siracusa, J (2014) INFINITE LOOP / THE APPLE ECOSYSTEM Available at: http://arstechnica.com/apple/2014/10/os-x-10-10/23/ [Accessed 26th March 2015].

Squires, J (2014) Apples to Apples Part III Available at: http://www.jessesquires.com/apples-to-apples-part-three/ [Accessed: 31st March 2015]

Tyrone (2014) Swift Performance: Too Slow for Production Available at: http://blog.sudeium.com/2014/12/10/swift-performance-too-slow-for-production/ [Accessed 27th March 2015].

Tyrone (2015) Swift 1.2 Performance: Pretty Much Fine Available at: http://blog.sudeium.com/2015/02/10/swift-1-dot-2-performance-pretty-much-fine/ [Accessed 27th March 2015].

Tyrosmith (2013) Low-Poly Rock Available at: http://tf3dm.com/3d-model/low-poly-rock-4631.html [Accessed 27th December 2014]

Tyrosmith (2014) Realistic Tree Pack Available at: http://tf3dm.com/3d-model/realistic-tree-pack-3-trees-95419.html [Accessed 27th December 2014]

Wavecade (2014) My Paper Plane 3 Available at: https://itunes.apple.com/ie/app/my-paper-plane-3/id478643935?mt=8 [Accessed 5th November 2014].