

Virtual Reality Driving Tutor Simulation

Cian Gardiner

*School of Computing
Dublin Institute of Technology
Kevin St. Dublin 8, Ireland.*

Project Supervisor: Dr Bryan Duggan

Table of Contents

DECLARATION.....	6
ABSTRACT.....	7
ACKNOWLEDGEMENTS	8
1 INTRODUCTION.....	9
1.1 INTRODUCTION	9
1.2 PROJECT BACKGROUND	9
1.3 PROJECT OBJECTIVES.....	10
1.4 PROJECT CHALLENGES.....	10
1.5 STRUCTURE OF THESIS	11
2 BACKGROUND RESEARCH.....	13
2.1 INTRODUCTION	13
2.2 INTENDED RESEARCH	13
2.3 RESEARCH	13
2.3.1 <i>Platform Research</i>	13
2.3.2 <i>Development Software</i>	13
2.3.3 <i>Existing Virtual Reality Driving Simulators</i>	15
2.3.4 <i>Most Common Driving Problems</i>	17
2.3.5 <i>Rules of the Road and How to Drive a Car</i>	17
2.4 CONCLUSION.....	17
3 DESIGN AND SOFTWARE METHODOLOGY.....	19
3.1 INTRODUCTION	19
3.2 SOFTWARE METHODOLOGY	19
3.2.1 <i>DSDM Principles:</i>	19
3.2.2 <i>DSDM Lifecycle</i>	20
3.2.3 <i>DSDM Timeboxing</i>	21
3.3 APPLICATION DESIGN	22
3.4 HORIZONTAL PROTOTYPE – SCREENS	23
3.5 SCENE LAYOUT.....	25
3.6 APPLICATION SCREEN SHOTS	25
4 ARCHITECTURE AND DEVELOPMENT.....	29
4.1 INTRODUCTION	29
4.2 SYSTEM ARCHITECTURE	29
4.3 DEVELOPMENT.....	31
4.3.1 <i>Introduction</i>	31
4.3.2 <i>Design Choices:</i>	31
4.3.3 <i>Equipment and Environment</i>	31
4.3.4 <i>Development Tools and Third Party Simulations</i>	32
4.3.5 <i>Development Components</i>	32
4.3.6 <i>Development Challenges</i>	43
5 SYSTEM VALIDATION	44
5.1 INTRODUCTION	44
5.2 UNIT TESTING	44

5.3	MODULE TESTING.....	44
5.4	BLACK BOX TESTING.....	45
5.5	USABILITY TESTING.....	45
5.5.1	<i>Usability Testing Conclusions</i>	46
5.6	EXPERT EVALUATION	47
6	PROJECT PLAN ANALYSIS.....	48
6.1	THE ORIGINAL PLAN.....	48
6.2	THE ACTUAL PLAN	50
6.3	REASONS FOR CHANGES	50
6.4	FUTURE PREVENTION OF PROBLEMS	51
7	EVALUATION, CONCLUSION AND FUTURE WORK	52
7.1	EVALUATION	52
7.2	CONCLUSION.....	54
7.3	FUTURE WORK.....	54
	APPENDIX A	56
	BIBLIOGRAPHY	60

Table of Figures

<i>FIGURE 1: SUPPORTING DOCUMENTATION FOR OCULUS RIFT INTEGRATION [10]</i>	14
<i>FIGURE 2: EUROTRUCK SIMULATOR 2 WITH THE OCULUS RIFT [11]</i>	15
<i>FIGURE 3: DRIVEON'S PHYSICAL ARCHITECTURE [12]</i>	16
<i>FIGURE 4: VIRTUAL REALITY DRIVING SIMULATION WITH THREE SCREENS [13]</i>	16
<i>FIGURE 5: iRACING, A RACING GAME WITH OCULUS RIFT INTEGRATION [14] [15]</i>	17
<i>FIGURE 6: SIMPLIFIED DSDM LIFECYCLE (ADAPTED FROM DSDM CONSORTIUM, 2004)</i> [16].	20
<i>FIGURE 7: APPLICATION USE CASE DIAGRAM</i>	22
<i>FIGURE 8: HORIZONTAL PROTOTYPE SCREENS</i>	24
<i>FIGURE 9: SCENE LAYOUT</i>	25
<i>FIGURE 10: MAIN MENU SCENE</i>	26
<i>FIGURE 11: REVERSE AROUND A CORNER SCENE</i>	26
<i>FIGURE 12: PARALLEL PARKING SCENE</i>	27
<i>FIGURE 13: FREE DRIVING SCENE</i>	27
<i>FIGURE 14: HOW TO PLAY SCEN</i>	28
<i>FIGURE 15: SYSTEM ARCHITECTURE</i>	30
<i>FIGURE 16: APPLICATION SCENES</i>	32
<i>FIGURE 17: EMPTY GAME OBJECT WITH WORLDLOADER SCRIPT ATTACHED</i>	33
<i>FIGURE 18: CODE EXTRACT FROM FREEDRIVE.TXT</i>	33
<i>FIGURE 19: LOAD() CODE SNIPPET</i>	34
<i>FIGURE 20: RAY CAST BEAM FROM RIGHT EYE</i>	35
<i>FIGURE 21: REVERSE AROUND A CORNER SCREENSHOT</i>	36
<i>FIGURE 22: PARALLEL PARKING SCREEN SHOT</i>	37
<i>FIGURE 23: FREE DRIVE SCENE SCREENSHOT</i>	37
<i>FIGURE 24: BUILDRIFTCAMERA() CODE SNIPPET</i>	38
<i>FIGURE 25: PLAYER CAR SCREENSHOT</i>	39
<i>FIGURE 26: PLAYER CAR - WHEELCOLLIDER EXAMPLE</i>	40
<i>FIGURE 27: SEEK STEERING BEHAVIOUR - AUTOMOTIVECAR</i>	41
<i>FIGURE 28: HANDBRAKE SCRIPT</i>	41
<i>FIGURE 29: UPDATE TRAFFIC LIGHT CODE SNIPPET</i>	42
<i>FIGURE 30: TRAFFIC LIGHT BOX COLLIDER SCREENSHOT</i>	43

Table of Tables

<i>TABLE 1: COMPARISON OF GAME ENGINE SYSTEM REQUIREMENTS [6]</i>	14
<i>TABLE 2: SURVEY QUESTIONS AND ANSWERS</i>	46

Declaration

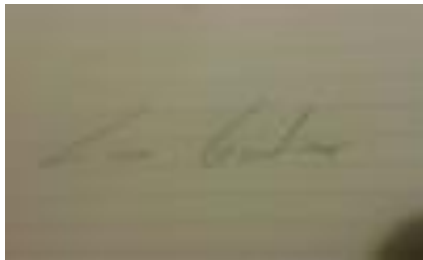
I certify that this thesis which I now submit for assessment for the award of Bachelor of Science (Hons) in Computer Science, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This thesis was prepared according to the regulations for postgraduate study by research of the Dublin Institute of Technology and has not been submitted in whole or in part for an award in any other Institute or University.

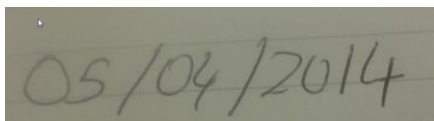
The work reported on in this thesis conforms to the principles and requirements of the Institute's guidelines for ethics in research.

The Dublin Institute of Technology has permission to keep, to lend or to copy this thesis in whole or in part, on condition that any such use of the material of this thesis is duly acknowledged.

Signed:

A photograph of a handwritten signature in dark ink on a light-colored background. The signature is cursive and appears to read 'L. Butler'. Below the photograph is a horizontal line.

Date:

A photograph of a handwritten date '05/04/2014' in dark ink on a light-colored background. Below the photograph is a horizontal line.

Abstract

The scientific community has been working in the field of virtual reality for decades, having recognised it as a very powerful human-computer interface [1]. Virtual reality is currently trending on the technological scene with the invention of an upcoming virtual reality head-mounted display, the Oculus Rift. What makes the Rift so desirable compared to other existing virtual reality devices is its immersive 3D stereoscopic rendering, low latency head tracking, a wide field of view and the fact that the average consumer can afford one [2].

The aim of this project is to create an application which teaches the basics of driving through a virtually simulated world. This application comes in the form of a Unity executable, playable on PC and uses the Oculus Rift as a head mounted virtual display and a force feedback steering wheel as the controller. The application offers three different driving scenarios. The user chooses an aspect of driving to practice or learn, are placed in a virtual car in a generated virtual city and can practice driving at their leisure.

The application is aimed at new drivers who have little or no experience driving a car but can be used by experienced drivers who wish to improve their driving.

The application will be written through C# scripts inside of Unity.

This report covers all work completed on this project and is comprised of background research, discussion of technologies investigated as well as justification for those chosen, design specifications including methodology used, development, testing, a project plan analysis, evaluation and possible future work.

Acknowledgements

To all the people around the world whose help, advice, articles, books, tutorials, and publications which I have listened to, read and used as a source of reference. I **sincerely** thank you for your effort and your time.

To Dr Bryan Duggan, my project supervisor, for always offering me advice whenever I asked for it.

To Jean, for showing me what a thesis should look like, for helping me code better and for teaching me basic grammar.

To my parents, Gerry and Jacqui, for supporting me in every way throughout my college career. I love you dearly.

Finally to Leonard, for reasons that don't even need mentioning.

1 Introduction

1.1 Introduction

Virtual reality is currently trending on the technological scene with the invention of an upcoming virtual reality head-mounted display, the Oculus Rift. OculusVR, the makers of the Rift, are at the stage of shipping out their second developer kit world-wide.

The aim of this project is to create an application which teaches basic driving in a safe virtual environment. Aspects of driving which many people find difficult to perform can be learned inside the application, such as parallel parking and reversing around a corner. The application is predominantly aimed at inexperienced drivers but can be utilised by drivers of all experience levels.

1.2 Project Background

Statistics show that inexperienced drivers are more likely to be involved in a car accident on Irish roads in comparison to more experienced drivers [4]. If a person knows how to perform a driving manoeuvre before experiencing it in real life, e.g., driving on black ice, it will reduce the risk of an accident occurring. The individual will better know how to react in the situation. This is the main reason why this project was chosen.

Irish drivers issued their first B category learner permit (for a car) on or after the fourth of April 2011 must complete twelve mandatory Essential Driver Training (EDT) lessons administered by an Approved Driving Instructor (ADI) before applying for a full driving licence [3]. These drivers must also have a fully qualified driver in the passenger seat, at all times, when driving on Irish Roads [4]. Practice is necessary for improvement and an individual does not always have access to a fully qualified driver. The completed application will allow an individual to practice driving in these instances.

The idea of creating a virtual reality driving tutor simulation using the Oculus Rift and a force feedback steering wheel comes from Dr Bryan Duggan (a lecturer in the School

of Computing, Dublin Institute of Technology). This type of application does not currently exist and it presents an opportunity to extend the scope of the Rift's virtual reality capabilities by creating a serious gaming application. Serious games are a branch of video games which are designed for a purpose other than pure entertainment [5]. The project focuses on creating a serious game where the goal is to teach an individual how to drive.

1.3 Project Objectives

The goal of this project is to create a virtual driving simulation which enables the user to learn how to drive from the safety of their own home. To accomplish this, a list of project objectives was created.

The objectives for the project are as follows:

- Research which game engines support the Oculus Rift.
- Decide which engine is best for this project.
- Research what driving is like and the rules of the road.
- Learn how to develop and create applications on the chosen engine.
- Create a virtual car which the user can control via a steering wheel and pedals.
- Create the driving environment.
- Create game logic.
- Create different driving scenarios.
- Integrate the Oculus Rift into the application.
- Create a user interface for the application.
- Test the completed application.

1.4 Project Challenges

This project will have multiple challenges throughout its lifecycle:

- Integrating the Oculus Rift into the application. The Rift is in its infancy and there is a lack of supporting documentation.
- Choosing the most suitable engine to build the application on. Why should one engine be chosen over the other?
- Completing all project objectives within the allotted time.

- Ensuring that testing is accurate and complete.

1.5 Structure of Thesis

The structure of the dissertation is as follows:

Section 1: Introduction

This chapter gives a general introduction to the project, detailing the project background, project objectives, project challenges and the dissertations structure.

Section 2: Background Research

This chapter describes background research carried out on the project, detailing books, papers and technologies researched. This chapter also details information about the problem being solved, other comparative work done so far and the criteria for selecting technologies.

Section 3: Design and Software Methodology

This chapter details the software methodology chosen for this project and why it was chosen. It also shows a high level design of the system shown through a use case diagram and explanation.

Section 4: Architecture and Development

In this chapter a system architecture diagram of the application is displayed and explained. Application design, such as the programming language chosen and the tools used are described in this chapter. Development work carried out is detailed in this chapter also. Development details include which equipment and environments were set up, which development tools were used and what components were made.

Section 5: System Validation

This chapter describes all testing carried out throughout and after the project's lifecycle, which includes unit testing, system testing, usability testing and an expert evaluation.

Section 6: Project Plan Analysis

This section gives an overview of the original plan and the plan that was actually implemented.

Section 7: Evaluation, Conclusion and Future Work

This final chapter evaluates the project as a whole, gives a finishing conclusion and planned future work is discussed.

2 Background Research

2.1 Introduction

This chapter details background research carried out during this project.

2.2 Intended Research

For the purpose of this project the following topics were researched and explored:

1. Potential development platforms.
2. Different kinds of development software.
3. Existing virtual reality driving simulators.
4. The most common driving problems.
5. Rules of the road and how to drive a car.

2.3 Research

The following sub-sections detail all research carried out on the topics listed above (see **Intended Research**).

2.3.1 Platform Research

It was decided early on that the application needed to be built for a mid to high-end desktop computer. This is due to the Rift needing to be connected to a computer to work. Also, sufficient computing power will be needed to run the application, which the average laptop would not have.

2.3.2 Development Software

In order to create a virtual driving tutor simulation, the most appropriate development tool or tools needed to be chosen. After reading Schroders research on the best way to implement virtual reality it was decided that a game engine needed to be used as the development tool. “In the research by Bell and Folger (2003), Lepouras and Vassilakis (2004), and Smith and Trenholme (2008) into the best way to implement virtual reality with a mainstream audience and affordable cost, all came to the same conclusion, that to use video game engines was the most practical approach.” [6].

2.3.2.1 Which Game Engine to use?

Three of the most popular game engines which have Rift support are Unreal Development Kit (UDK), Unity and CryENGINE 3. Each were researched as potential candidates for creating the application. All three engines have well

documented support as well as a large user bases. Unity was chosen as the most suitable game engine to use in the end.

It was noted that Unity requires the least amount of computing power to run after looking at each game engine's respective websites [7] [8] [9].

	CryEngine 3	Unity	UDK
Operating System	Windows 7 64-bit	Windows XP SP2	Windows 7 64-bit
CPU	Intel i7 3 GHz	N/A	Multi-core at 2 GHz
Memory (RAM)	3 GB	N/A	3 GB
Video (RAM)	1.8 GB	N/A	512 MB
DirectX Support	DirectX 9 and 11	DirectX 9	DirectX 9 and 11

Table 1: Comparison of Game Engine System Requirements [6]

It was also noted that the OculusVR forum offer more support and documentation for integrating the Rift into the Unity game engine than any other game engine (see **Figure 1**). It was also noticed that the vast majority of Rift applications made to date were made using Unity. It is for these reasons that Unity was chosen as the development software to be used for this project.













OCULUS INTEGRATIONS		TOPICS	POSTS	LAST POST	
	Unity 4 Integration Discuss the Oculus - Unity 4 integration.		252	1903	by UberLax  Sun Dec 08, 2013 12:08 am
	Unreal Development Kit (UDK) Integration A place to discuss the Oculus-ready version of the UDK.		87	790	by BlackFang  Thu Dec 05, 2013 8:31 am
	Unreal Engine 3 Integration Discuss the Oculus - Unreal Engine 3 integration.		6	35	by cybereality  Fri Dec 06, 2013 11:07 pm
	Other Engine Integrations A forum to discuss integrating the Oculus SDK with other engines.		45	394	by Neith  Fri Dec 06, 2013 8:29 pm

Figure 1: Supporting documentation for Oculus Rift Integration [10]

2.3.3 Existing Virtual Reality Driving Simulators

Existing virtual reality driving simulators were researched to gain a better insight into how to create the most immersive driving experience.

Eurotruck Simulator 2 is a game available on steam, which has Oculus Rift support. The aim of the game is to transport cargo from one destination to next. The game combined with the Rift fully immerses the use, placing you inside a virtual truck in a virtual world.



Figure 2: Eurotruck Simulator 2 with the Oculus Rift [11]

DriveON, another virtual reality driving simulator, was created by a group of Serbian students for the 2007 imagine cup. Their application offers an eLearning driving experience. The user has a 360 degree field of view, the virtual world displayed by multiple projectors around a cylindrical panel. All the physical attributes one finds in a real car are found within the driving station. An automated virtual instructor navigates, guides and warns drivers about incorrect driving. A physical instructor can control weather and traffic conditions, and remove or place cars in the simulation while the user is driving [12].



Figure 3: DriveON's Physical Architecture [12]

Other virtual reality driving simulations were researched and they all shared a common pattern which can be seen in the following two figures.



Figure 4: Virtual Reality Driving Simulation with three screens [13]



Figure 5: iRacing, A Racing Game with Oculus Rift Integration [14] [15]

The common pattern is that the user sits behind a force feedback steering wheel and interacts with the application by looking at a screen or screens. The user then carries out tasks from a predefined list of objectives.

The Oculus Rift is the most immersive of all the different virtual reality driving simulators. It is also the simplest, most cost effective way of implementing virtual reality.

2.3.4 Most Common Driving Problems

Ollie Blake, an ADI, was questioned about the most problematic areas drivers have when learning how to drive. His answer was threefold, “Observation, parallel parking and reversing around a corner”.

2.3.5 Rules of the Road and How to Drive a Car

Learning the rules of the road and how to drive a car was vital research for this project. It made it clear what driving a real car on a real road is like, experiences which were translated straight into the application.

2.4 Conclusion

Research has been conducted and the following is a list of conclusions taken from this research.

- This project’s application needs to be built for a mid to high end PC.

- The best way to build this project's application is through a game engine. Unity was chosen for this project.
- The Rift offers a more immersive virtual reality experience than any other virtual reality attempt.
- The three main aspects of driving to focus on should be observation, parallel parking and reversing around a corner.
- Based on the short time period of this project and the large number of road rules, the most basic and important functionality requirements should be implemented first, i.e. the car, the driving environment, the different driving scenarios and the game logic. If possible, more features can be implemented near the end of the project's lifecycle.

3 Design and Software Methodology

3.1 Introduction

This chapter describes the application's design at a high level. This chapter also describes the software methodology used to structure, plan and control the application's development process.

3.2 Software Methodology

Dynamic systems development method (DSDM) is the software methodology used in guiding this project to completion. DSDM is a management and control framework for rapid application development (RAD). RAD is a set of prototyping techniques and tools used to produce software applications. DSDM's principles stress on product delivery and focus on fitness for purpose. This combined with DSDM's development lifecycle and timeboxing strategy lead to choosing it as the software methodology for this project.

3.2.1 DSDM Principles:

DSDM adheres to nine underlying principles [16].

1. Users are viewed as project team members and active user involvement is imperative.
2. Decisions can be made at any time which refine or even change system requirements.
3. Products are delivered within an agreed time period, the best suited approach being picked to ensure delivery. Time periods, known as "timeboxes" are kept relatively short, which help decide in advance what is feasible.
4. Acceptance of the final deliverable is based on the fitness of the application. Essential functionality of the application is delivered at the specified time.
5. Iterative and incremental development is necessary in creating an accurate solution. Incremental development allows user feedback which benefits future development. Delivery of a partial solution is accepted if it satisfies an immediate and urgent user need. The solution can be refined and developed further on.

6. Changes made in development are kept reversible. As a result changes are limited within a particular increment.
7. Requirements are initially agreed at a high level at the start of the project giving a basis for prototyping.
8. Testing is integrated throughout the development lifecycle. Software components are tested during development rather than after completion.
9. A collaborative and co-operative approach between all stake holders is essential.

3.2.2 DSDM Lifecycle

The DSDM lifecycle contains five phases: feasibility study, business study, functional model iteration, design and build iteration and implementation [16].

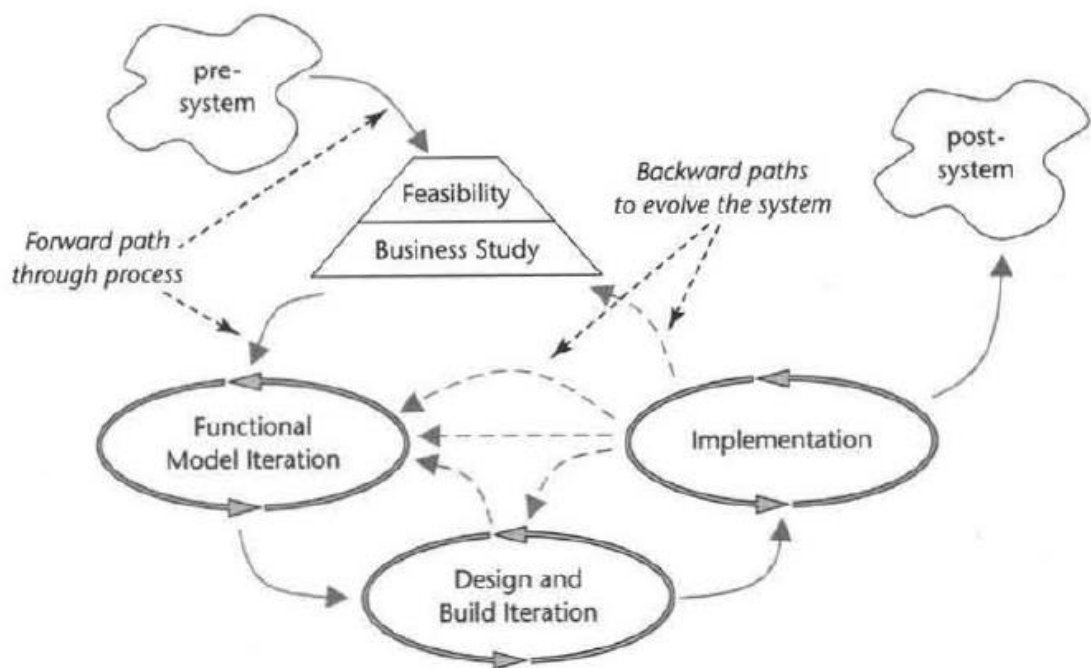


Figure 6: Simplified DSDM lifecycle (adapted from DSDM Consortium, 2004) [16].

The feasibility study checks whether the project is suitable for a DSDM approach. This study answers three questions: whether the project is technically possible, whether or not the benefit of the system will be outweighed by its costs and if the project operates acceptably within its organization.

The business study phase identifies the scope of the project and gives a high level analysis of the project's requirements (See **Application Design**). Maintainability objectives are set at this stage.

The functional model iteration phase deals with development of prototypes which elicit project requirements. A functional prototype is identified e.g. creating a moveable car. A schedule for the prototype is agreed, the prototype is created and then reviewed.

The design and build iteration phase is concerned with developing prototypes to the point where they can be used operationally. The design and build iteration phase and functional model iteration phase are similar and run concurrently.

The implementation phase is the final phase and deals with instalment of the latest increment into the project. If an element of the project was omitted due to time constraints the project goes back to the functional model iteration phase. If non-functional requirements need to be addressed the projects goes back to the design and build iteration phase. If a new functional area is identified, the project returns to the business study phase. If all the requirements have been met then the project is complete.

3.2.3 DSDM Timeboxing

Timeboxing, an allocation of resources including time, was used to allocate resources in this project. Smaller timeboxes were used to prioritize functionalities within allocated timeboxes. Timeboxes produce deliverables, which allow for assessment of progress and quality. Timeboxing relies on prioritizing requirements and follows a 'must, should, could, won't' requirements rule (MoSCoW). Must have requirements are crucial and cannot be omitted. Should have requirements are important but can operate without them. Could have requirements are desirable but provide less benefits to the user. Won't have requirements can reasonably be left for development in a later increment [16].

3.3 Application Design

When the application starts, a main menu is displayed to the user. The user may quit the application from the main menu or load a driving scene from a list of driving scenarios. The available driving scenes include free driving, parallel parking and reversing around a corner. When a driving scene is loaded the user is placed in the driving seat of a car, in an urban environment. Normal traffic conditions occur around the driver. The steering wheel, acceleration and brake pedals are used to control the user's vehicle. The Oculus Rift is used to visually display the scenario. The user can then practice driving at their own leisure. The user may stop gameplay at any time during the driving scene by pressing escape. This sends the user back to the main menu.

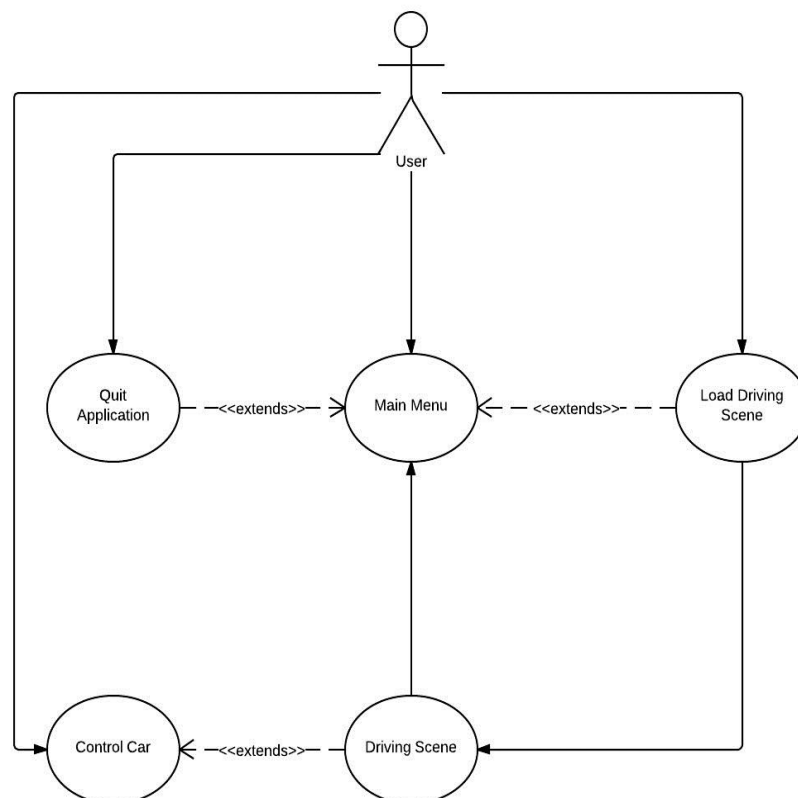


Figure 7: Application Use Case Diagram

The figure above is a use case diagram representing the application's design. For each interaction where a solid line connects to the user, that interaction may only take place if its corresponding interaction is taking place, e.g., the user cannot control the car unless a driving scene has been loaded, and the driving scene cannot be loaded unless the user selects a driving scene from the menu. Sub actions of 'Control Car' such as steer, accelerate and brake are not mentioned in this Use Case diagram but are encapsulated inside that action.

3.4 Horizontal Prototype – Screens

The following figure represents the initial horizontal prototype made for the application.

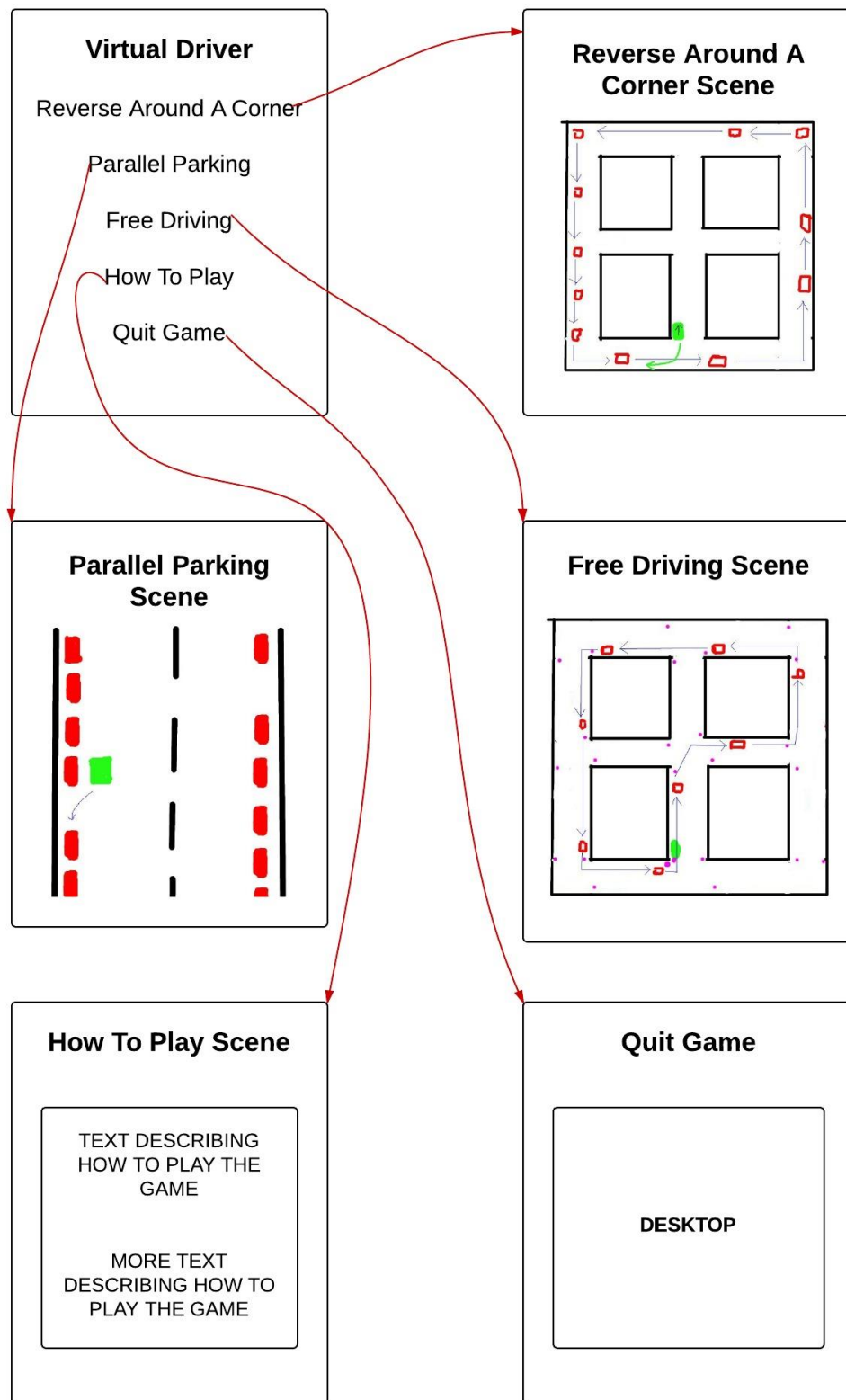


Figure 8: Horizontal Prototype Screens

3.5 Scene Layout

The horizontal prototype was used to modularise functionalities for the different scenes. The user has the ability to navigate between scenes. The following diagram shows the layout of the different scenes and their relationship to each other.

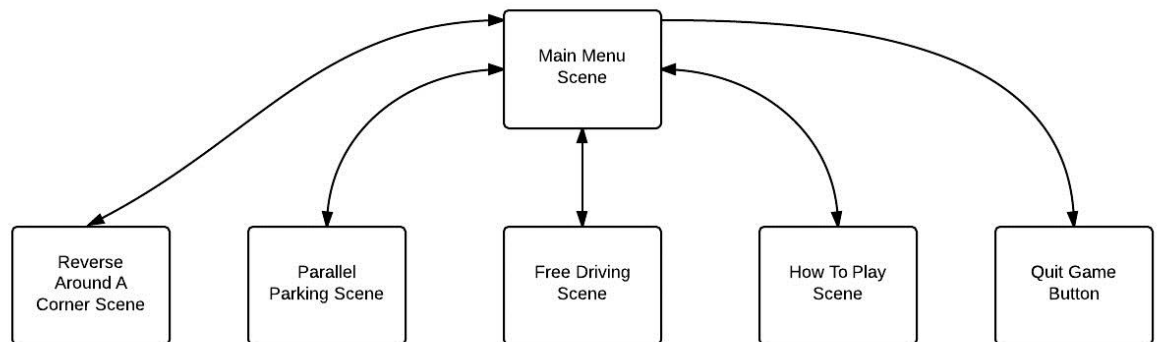


Figure 9: Scene Layout

3.6 Application Screen Shots

The following figures show screenshots of how each scene's design was implemented. The process of creating the scenes is described in the following chapter (see **Architecture and Development**).



Figure 10: Main Menu Scene

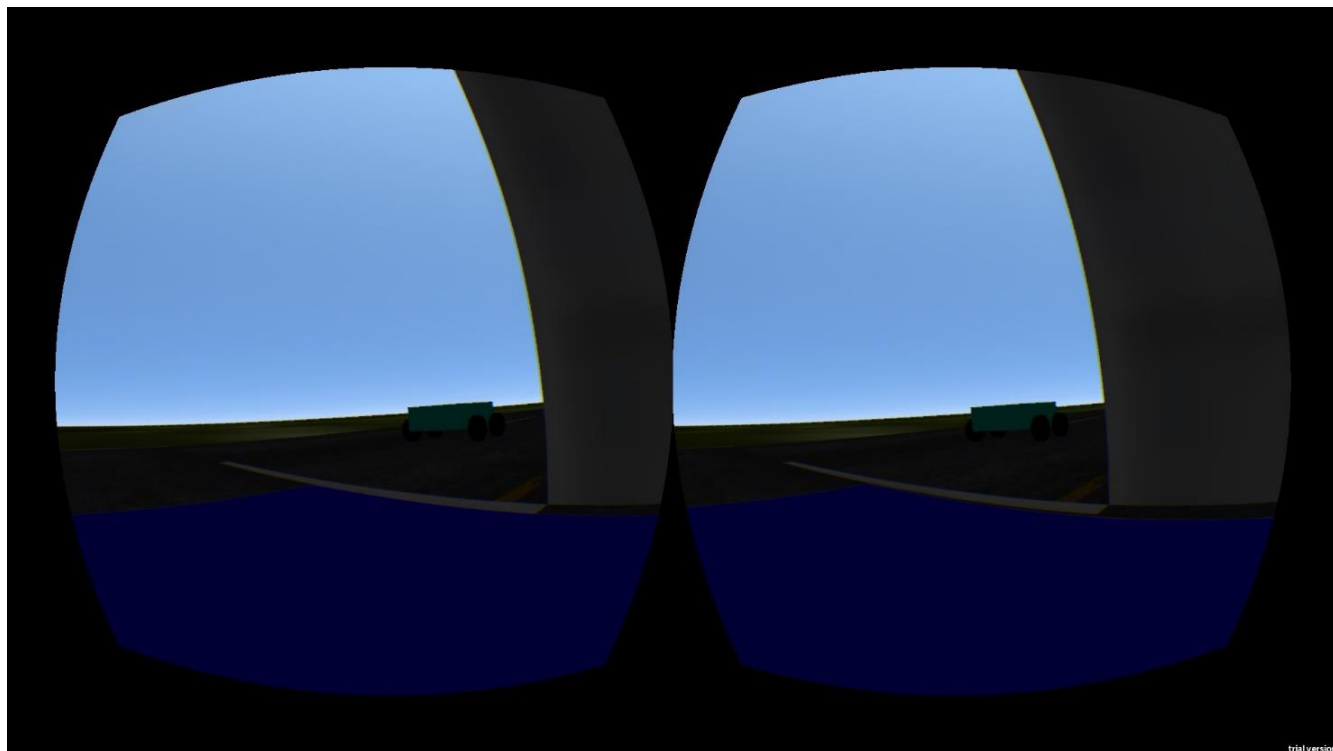


Figure 11: Reverse Around a Corner Scene

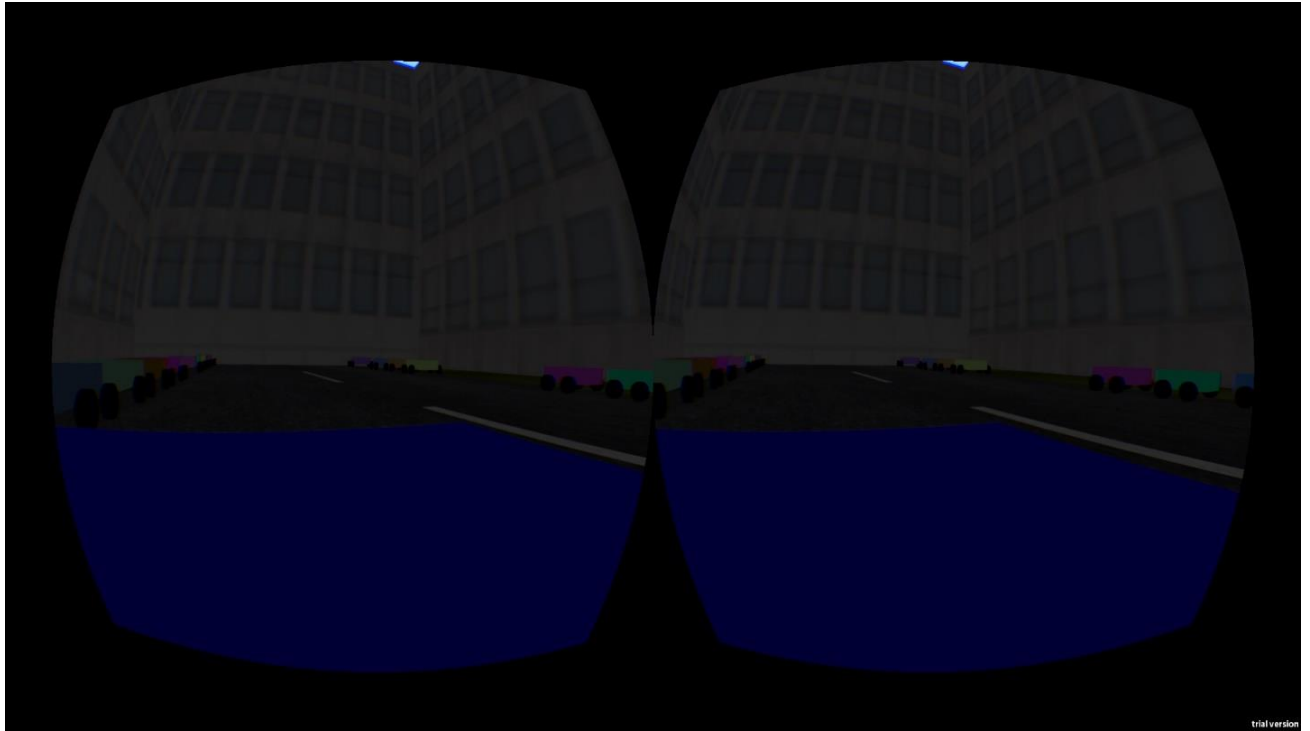


Figure 12: Parallel Parking Scene

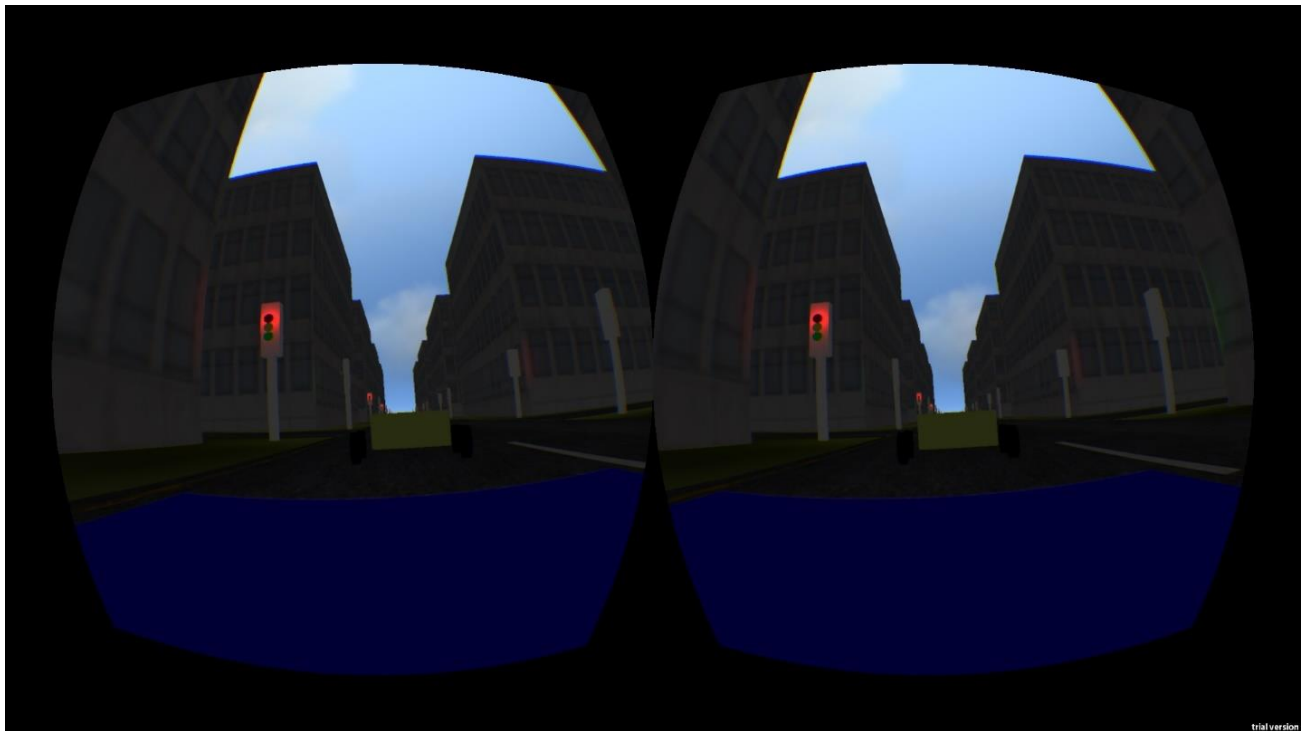


Figure 13: Free Driving Scene



Figure 14: How to Play Scen

4 Architecture and Development

4.1 Introduction

This chapter details the system architecture and the development process of the application. The chapter goes through the different design choices for the application, the programming language chosen and model generation techniques.

4.2 System Architecture

Hardware components of the system's architecture include a mid-range desktop PC, a force feedback steering wheel with pedals and the Oculus Rift. Software components of the system's architecture include the Unity game engine and the final game application. The user interacts with the application via the Rift and steering wheel. The application is run on a desktop-PC and is launched through a Unity executable. The following figure depicts the application's system architecture.

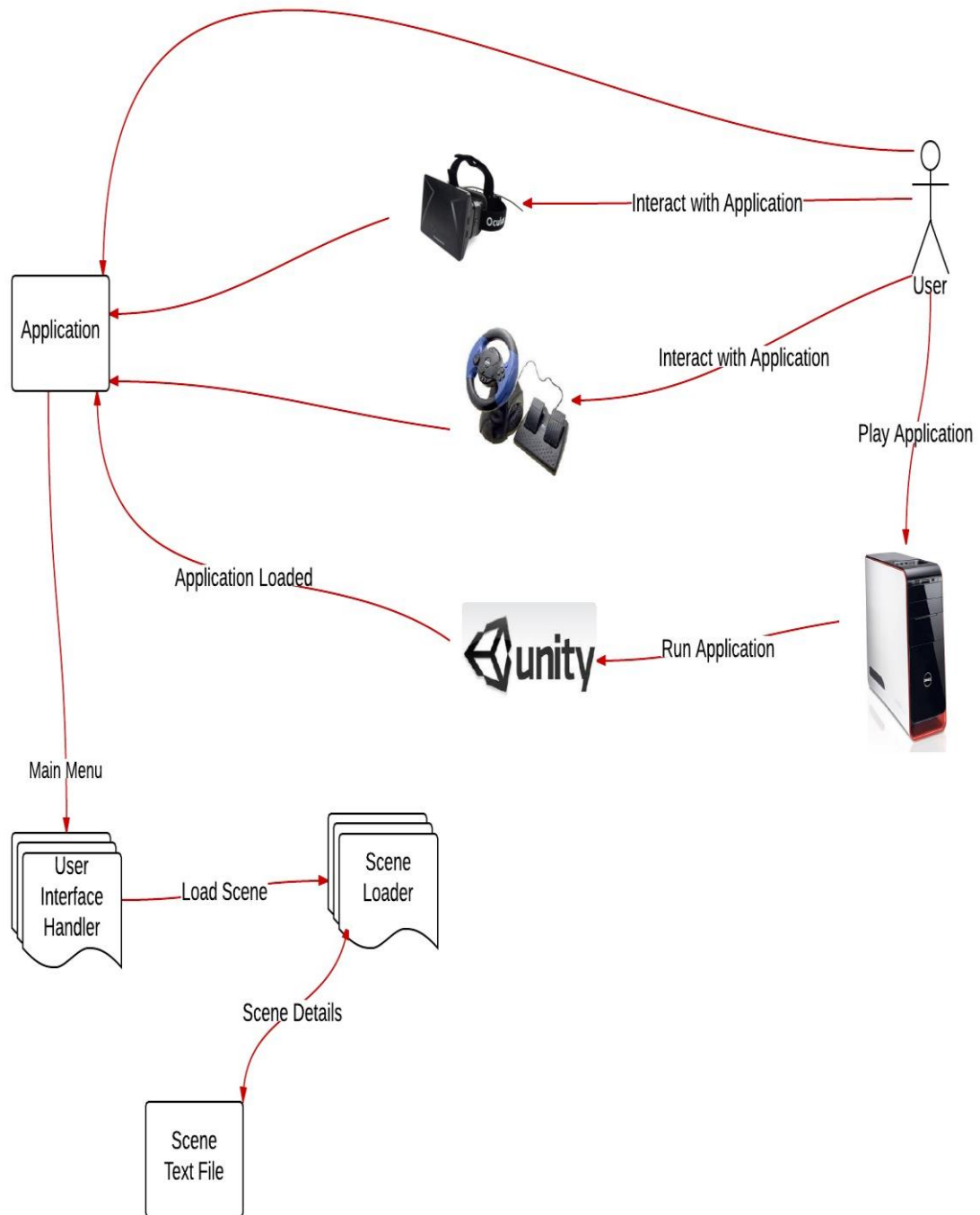


Figure 15: System Architecture

The user is an individual using the application. The application is encapsulated in a Unity executable which can run on PC, Mac or Linux. A mid to high spec desktop

computer is desired for faster frame rates, as OculusVR state on their website in relation to the minimum system requirements needed for the rift - “the more powerful the computer, the better the experience will be.” [17].

After the user has loaded the application he or she can interact with it. This is accomplished by using the rift and steering wheel, both of which are connected to the desktop. The rift acts as a head mounted virtual display as well as a pointing device. The steering wheel is used to manoeuvre the user’s car when a scene has loaded. Scenes are loaded by the Scene Loader. The Scene Loader reads in scene details from a text file and builds the scene based on the specified entities.

4.3 Development

4.3.1 Introduction

This section describes all development work completed on the application, what design choices were made, what environments were set up, what development tools were used, what third party simulations were sourced, what components of the application were created and what development challenges were faced.

4.3.2 Design Choices:

The application was created using C# scripts inside the Unity game engine. ‘Scenes’ were used to handle the applications different screens, such as the main menu and the parallel parking scenario. “**Scenes** contain the objects of your game. They can be used to create a main menu, individual levels, and anything else.” Unity supports three programming languages: C#, UnityScript (a branch of JavaScript) and Boo. C# was chosen as the programming language to write in as it is the most intuitive language Unity offers. C# offers more functionality than UnityScript and Boo as it gives the coder access to the .NET library [18]. C# is also the most popular language with Unity developers [19], which made trouble shooting and finding tutorials a lot easier.

4.3.3 Equipment and Environment

The following equipment and environment were required before development on the application could commence:

- A PC capable of running Unity and displaying games with Oculus Rift integration.
- An Oculus Rift.
- A force feedback steering wheel with pedals.
- Unity Pro – 4.2, Trial Version.

4.3.4 Development Tools and Third Party Simulations

Various third party Unity car simulators with source code provided were downloaded and inspected [20] [21] [22]. Source code was extracted and modified from these simulators, all of which is accounted for in the development components section, (see **Development Components**). An editor script, which allows testing of an Oculus Rift integrated Unity application was installed and used as a supplementary development tool [23]. The Unity scripting reference was used extensively to help solve developmental problems throughout the projects lifecycle [24]. Learning how to program for Unity was vital for this project's success. The Unity scripting reference details every aspect of coding within Unity.

4.3.5 Development Components

This section details all development components created for this application.

4.3.5.1 WorldLoader Script

WorldLoader is the most important script in the project and is used by every scene in the application. The scenes created for this application include MainMenu, ReverseAroundACorner, FreeDrive, ParallelParking and HowToPlay (see *Figure 16: Application Scenes*).

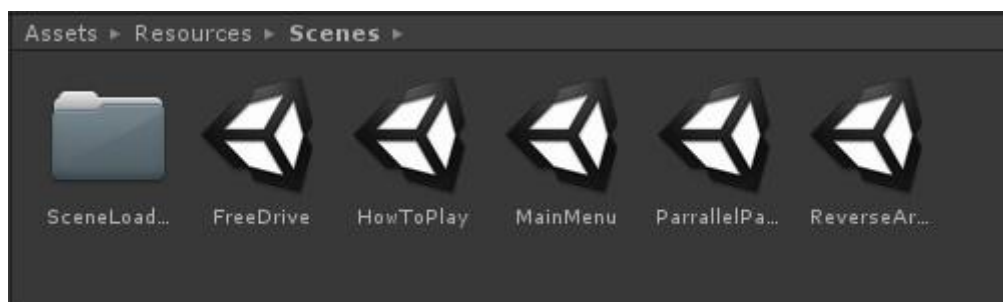


Figure 16: Application Scenes

Each scene in the application contains a singular empty game object with an attached “WorldLoader” script (see *Figure 17*), which acts as the scene builder.

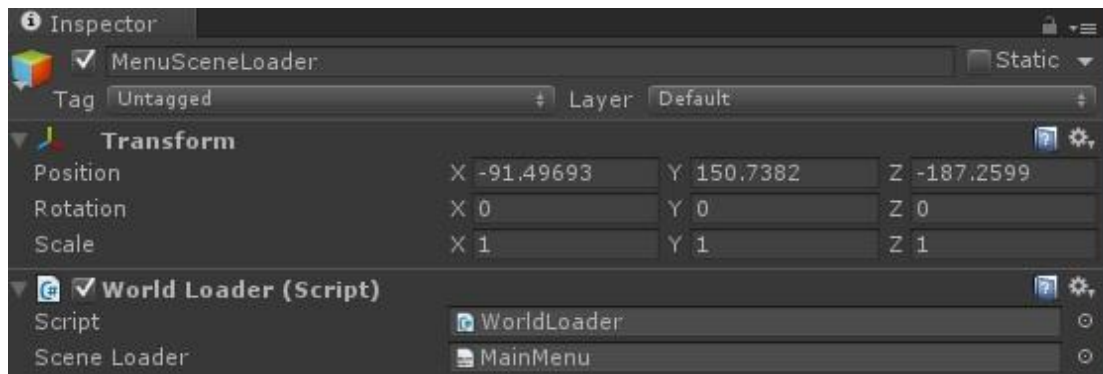


Figure 17: Empty Game Object with WorldLoader Script attached

WorldLoader reads in data from a predefined text file and builds the scene according to values within the text file. The following figure (see *Figure 18*) is a code extract from “FreeDrive.txt”.

```

15 path_parent,0,0,0,0,0,0,0,0,0
16 player_car,50,1.5,50,2,1,3,0,0,0
17 automotive_car,44.5,1.5,60,2,1,3,0,0,0,1
18 automotive_car,44.5,1.5,80,2,1,3,0,0,0,1
19 static_car,44.5,1.5,40,2,1,3,0,0,0,1
20 junction,-75,0.01,112.5,25,0.01,25,0,0,0
21 junction,-75,0.01,237.5,25,0.01,25,0,0,0
22 junction,-75,0.01,362.5,25,0.01,25,0,0,0
23 junction,-75,0.01,487.5,25,0.01,25,0,0,0
24 straight_road,-75,0.01,50,25,0.01,100,0,0,0
25 straight_road,-75,0.01,175,25,0.01,100,0,0,0
26 straight_road,-75,0.01,300,25,0.01,100,0,0,0
27 straight_road,-75,0.01,425,25,0.01,100,0,0,0
28 straight_road,-75,0.01,550,25,0.01,100,0,0,0
29 junction,-75,0.01,-12.5,25,0.01,25,0,0,0
30 junction,50,0.01,-12.5,25,0.01,25,0,0,0
31 junction,175,0.01,-12.5,25,0.01,25,0,0,0
32 junction,300,0.01,-12.5,25,0.01,25,0,0,0
33 junction,425,0.01,-12.5,25,0.01,25,0,0,0
34 junction,550,0.01,-12.5,25,0.01,25,0,0,0

```

Figure 18: Code Extract from FreeDrive.txt

Each line in the text file specifies the object name, position, scale, rotation and in special cases, such as automotive cars, the current target path point.

```

//Loads in the contents of a file
private void Load(TextAsset fileName)
{
    //Splits text file into individual lines
    string[] lines = fileName.text.Split("\n"[0]);

    //Reads through every text line
    foreach(string line in lines)
    {
        lineCounter++;

        //Comma delimited values
        string[] entries = line.Split( ',' );

        if ( entries.Length == 10 )
        {
            BuildObject( entries );
        }
        else if( entries.Length == 11 )
        {
            currentPathPoint = int.Parse( entries[10] );
            BuildObject( entries );
        }
        else
        {
            Debug.Log( "Error at Line: " + lineCounter );
        }
    }
}

```

Figure 19: Load() code snippet

Once a line has been read in, BuildObject() (see **Appendix A**) is called. This sets the objects name, position, scale and rotation and calls the appropriate build function. . A list of all game objects the application can create can be found in **Appendix A**. Note that additional scripts are added to game objects within the scene once their structure has been created.

4.3.5.2 Main Menu Scene

The main menu scene is created by loading the contents of MainMenu.txt via WorldLoader. MainMenu.txt contains a single line, which triggers BuildMainMenu() (see **Appendix A**). A menu is then created and displayed to the user (see *Figure 10: Main Menu Scene*). Text displayed to the user is created using a text mesh asset [25]. The user can look around their environment using the Rift and can interact with the

menu by looking at an item for a period of two seconds. RiftMenuChooser was created to interact with the main menu. This script is attached to the right eye of the main camera (see **Main Camera**). A ray is cast from the right eye's forward vector and performs a task if it hits a point of interest, e.g., the *How to Play* text mesh.

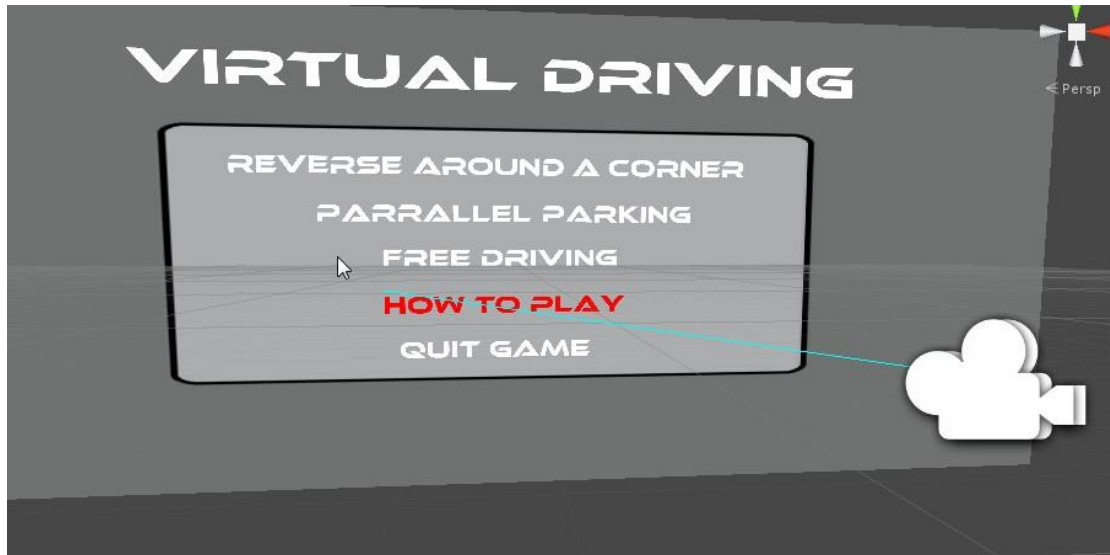


Figure 20: Ray cast beam from right eye

4.3.5.3 Reverse Around a Corner Scene

The reverse around a corner scene is created by loading the contents of *ReverseAroundACorner.txt* via *WorldLoader*. The player car (see **Car**) starts in a laneway between two buildings and the goal is to reverse around the corner, out of the laneway without colliding with any cars already on the main road. Automotive cars (see **prevents the** user car from travelling over 148 km/h, an approximate top speed for a normal car. *AntiRollBar* stops the car from flipping when it takes a sharp bend; The logic for this script was obtained from Edy on the Unity forums) in this scene follow a predefined path around the outsides of the building.

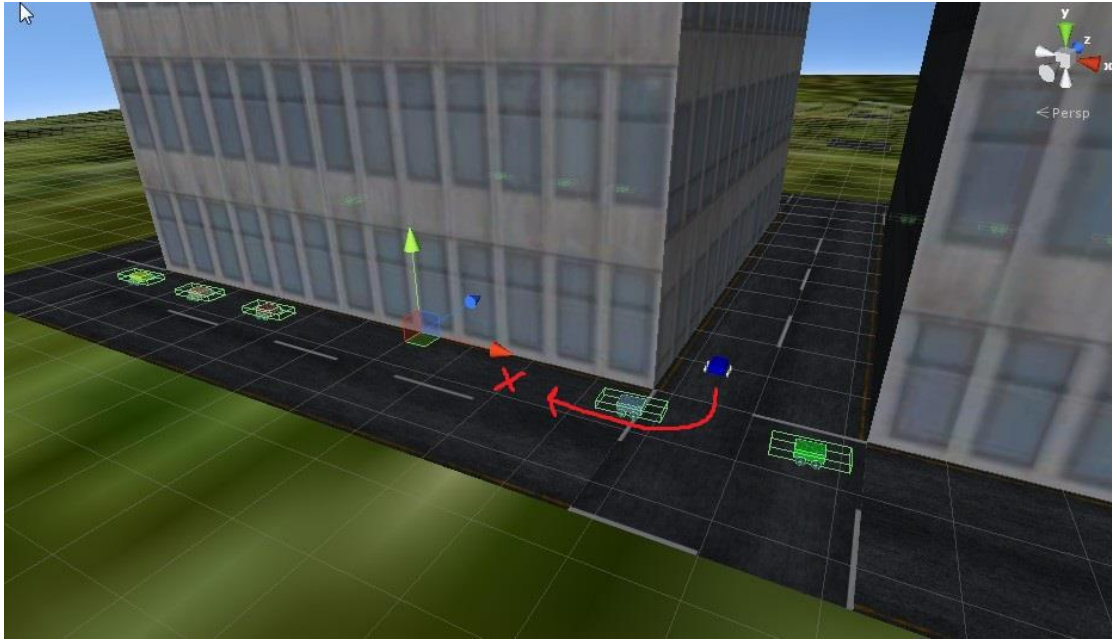


Figure 21: Reverse Around a Corner ScreenShot

4.3.5.4 Parallel Parking Scene

The parallel parking scene is created by loading the contents of ParallelParking.txt via WorldLoader. The goal of this scene is to perform a parallel parking manoeuvre. To accomplish this a straight road was created and static cars were placed at varying positions on each side of the road only leaving enough room for a single car to park. All static cars have a HandBrake script which applies a constant brake torque to the wheels of each car.

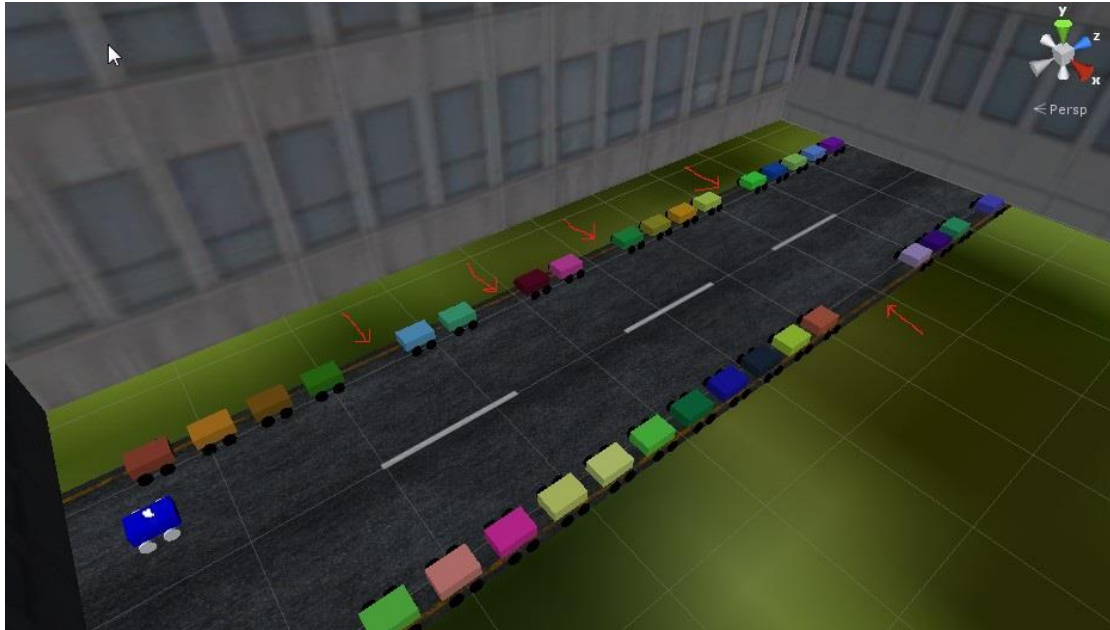


Figure 22: Parallel Parking Screen Shot

4.3.5.5 Free Drive Scene

The free drive scene is created by loading the contents of Freedrive.txt. The goal of this scene is to let the user drive around unhindered so they can get a feel for the road and experience real life traffic conditions. To accomplish this, a city was built in a grid layout, a traffic light system was implemented (see **Traffic Light**) and a traffic system was put in place.

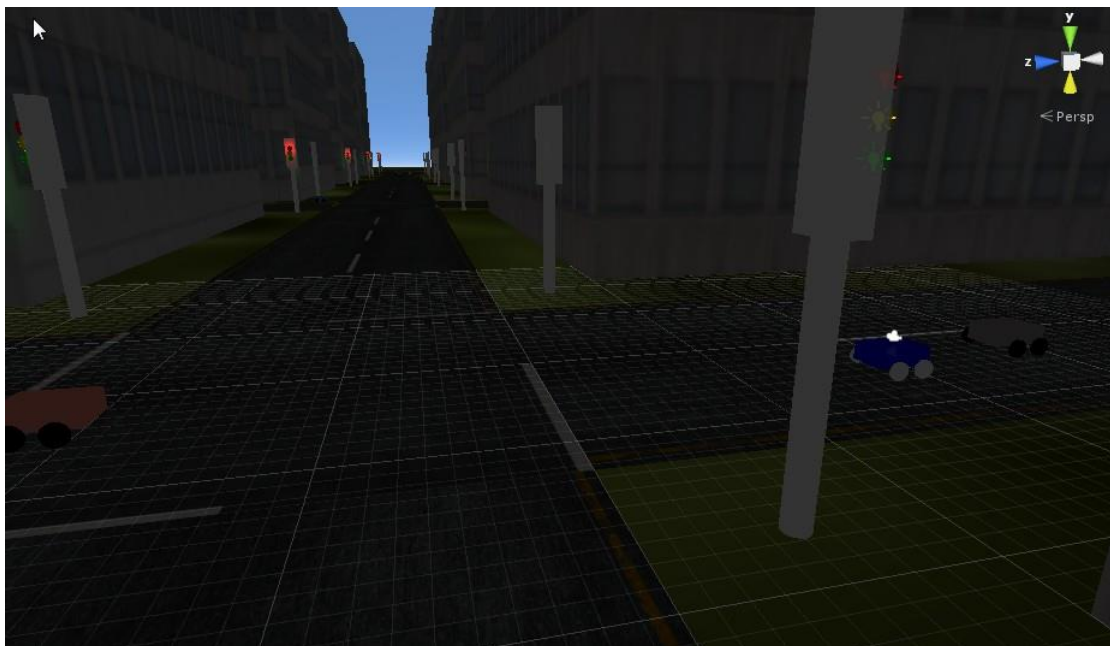


Figure 23: Free drive scene screenshot

4.3.5.6 How To Play Scene

This scene is created in the same way as the main menu screen, the only difference being that the text in this scene is changed.

4.3.5.7 Main Camera

The camera in this application is what allows the user to look around their environment when interacting with the application (see *Figure 13: Free Driving Scene*). The camera is also used as pointing device in the Main Menu scene (see **Main Menu Scene**). The camera is created when BuildRiftCamera() is called from WorldLoader (see **Figure 24**).

```
void BuildRiftCamera(GameObject gameObject)
{
    GameObject OVRCameraController = new GameObject();
    AddGameObjectDetails( OVRCameraController, "OVRCameraController",
                        null, gameObject.transform.position,
                        Vector3.zero, Vector3.zero, gameObject );
    OVRCameraController.AddComponent( "OVRDevice" );

    //Left Eye Camera
    GameObject cameraLeft = new GameObject();
    AddGameObjectDetails( cameraLeft, "CameraLeft", null,
                        OVRCameraController.transform.position,
                        Vector3.zero, Vector3.zero, OVRCameraController );
    AddRiftCamera( cameraLeft, 1, new Rect( 0.0f, 0.0f, 0.5f, 1.0f ) );

    //Right Eye Camera
    GameObject cameraRight = new GameObject();
    AddGameObjectDetails( cameraRight, "CameraRight", null,
                        OVRCameraController.transform.position,
                        Vector3.zero, Vector3.zero, OVRCameraController );
    AddRiftCamera( cameraRight, 0, new Rect( 0.5f, 0.0f, 0.499999f, 1.0f ) );

    OVRCameraController.AddComponent( "OVRCameraController" );
}
```

Figure 24: BuildRiftCamera() code snippet

Note that all Oculus Virtual reality scripts used in this application, OVRCameraController, OVRCamera, OVRComponent, OVRCrosshair, OVRDevice, OVRGamepadController, OVRGUI, OVRMagCalibration, OVRMainMenu, OVRMessneger, OVRPlayerController, OVRPresetManager, OVRUtils, OVRImageEffects and OVRLensCorrection are written by OculusVR [10]. Certain values in some of these scripts were tweaked to create a more desirable rift camera, e.g.,

the far clip plane in OVRCamera was changed to a larger value due to the large nature of the skybox surrounding the world.

4.3.5.8 Car Scripts

Each car in the application is created from a BuildCar() function in WorldLoader. The only difference between the three different types of cars - the player car, static cars and automotive cars - is the scripts attached to each.

PlayerCarController, VelocityLimiter and AntiRollBar scripts are all attached to the player car game object. PlayerCarStopTrigger script is added to the player car's box collider.

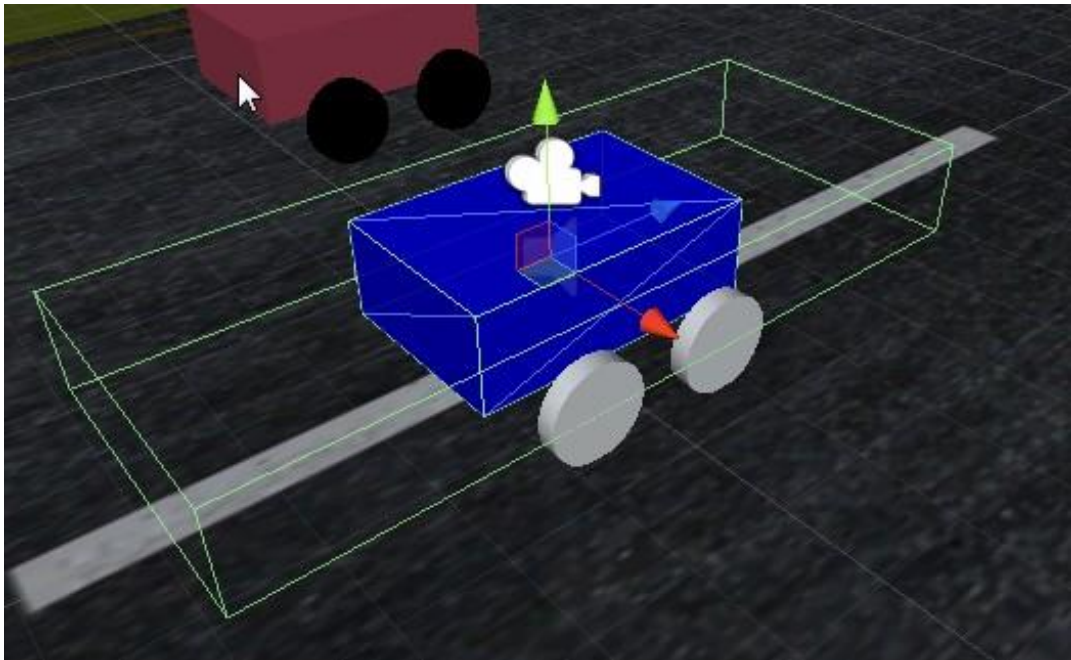


Figure 25: Player Car Screenshot

PlayerCarController is responsible for accepting input from the user and manoeuvring the car in the scene. Movement is based on Unity's wheel collider's components [26]. It is calculated within a fixed Update function, as recommended by Unity when dealing with physics on rigid bodies.

```

//Caclulate EngineRPM
engineRPM = ( backLeftWheelCollider.rpm + backRightWheelCollider.rpm ) / 2
    * gearRatios[ currentGear ];

//Moves car forward or backward based on what gear the user is in
if(reversing == false)
{
    backLeftWheelCollider.motorTorque =
        ( engineTorque / gearRatios[currentGear] ) *
        Input.GetAxis("Vertical");

    backRightWheelCollider.motorTorque =
        ( engineTorque / gearRatios[currentGear] )
        * Input.GetAxis("Vertical");
}
else
{
    backLeftWheelCollider.motorTorque =
        ( engineTorque / gearRatios[currentGear] )
        * Input.GetAxis("Vertical") * -1;
    backRightWheelCollider.motorTorque =
        ( engineTorque / gearRatios[currentGear] ) *
        Input.GetAxis("Vertical") * -1;
}

```

Figure 26: Player Car - WheelCollider Example

VelocityLimiter prevents the user car from travelling over 148 km/h, an approximate top speed for a normal car. AntiRollBar stops the car from flipping when it takes a sharp bend; The logic for this script was obtained from Edy on the Unity forums [21]. PlayerCarStopTrigger is activated if an automotive car enters the player car box collider and is behind the player car game object. This prevents automotive cars from ramming the player car from behind.

All automotive cars contain an AutomotiveCar script and an AutomotiveCarStopTrigger script. AutomotiveCar is responsible for making automotive cars in the scene travel along a path, moving from one point to the next using a 'seek' steering behaviour.


```
//Seek steering behaviour
Vector3 Seek( Vector3 targetPos )
{
    Vector3 desiredVelocity = targetPos - transform.position;
    desiredVelocity.y = 0;
    transform.rotation = Quaternion.Slerp( transform.rotation,
                                           Quaternion.LookRotation( desiredVelocity ),
                                           2.5f * Time.deltaTime);

    Vector3 moveVector = desiredVelocity.normalized *
                        |carSpeed * Time.deltaTime;

    return moveVector;
}
```

Figure 27: Seek Steering Behaviour - AutomotiveCar

AutomotiveCarStopTrigger's function is to prevent automotive cars from colliding with each other while they are following their chosen paths. This was accomplished by using the automotive car's box collider and the dot product.

Static cars contain a single script called HandBrake which sets a constant brake torque on the cars wheels.

```
public class HandBrake : MonoBehaviour
{
    WheelCollider backLeftWheelCollider;
    WheelCollider backRightWheelCollider;

    // Use this for initialization
    void Start ()
    {
        //Setting Wheel Collider Variables
        backLeftWheelCollider = ( WheelCollider )gameObject.transform.
            Find( "Wheels/BackLeftWheel/BackLeftWheelCollider" ).
            GetComponent( typeof( WheelCollider ) );
        backRightWheelCollider = ( WheelCollider )gameObject.transform.
            Find( "Wheels/BackRightWheel/BackRightWheelCollider" ).
            GetComponent( typeof( WheelCollider ) );

        //Brake Torque acts as a handbrake
        backLeftWheelCollider.brakeTorque = 600.0f;
        backRightWheelCollider.brakeTorque = 600.0f;
    }
}
```

Figure 28: HandBrake Script

4.3.5.9 Traffic Light Scripts

TrafficLightManager is the script used to implement traffic light behaviour in the scene. Each traffic light has a parent game object and based on the name of the parent game object the traffic lights get updated accordingly. The following figure shows the logic behind updating each traffic light in the scene.

```
//Updates Traffic Light
void UpdateTrafficLights( List< Light > greenLights, List< Light > orangeLights, List< Light > redLights, ref float timer )
{
    //Change From Red To Green
    if( timer >= trafficLightTimes.z && redLights[0].enabled == true)
    {
        for( int i = 0 ; i < greenLights.Count ; i++ )
        {
            EnableLights( greenLights[i], orangeLights[i], redLights[i], true, false, false );
        }

        timer = trafficLightTimes.y;
    }

    //Change From Green To Orange
    if( timer >= trafficLightTimes.x && greenLights[0].enabled == true)
    {
        for( int i = 0 ; i < greenLights.Count ; i++ )
        {
            EnableLights( greenLights[i], orangeLights[i], redLights[i], false, true, false );
        }

        timer = 0.0f;
    }

    //Change From Orange To Red
    if( timer >= trafficLightTimes.y && orangeLights[0].enabled == true)
    {
        for( int i = 0 ; i < greenLights.Count ; i++ )
        {
            EnableLights( greenLights[i], orangeLights[i], redLights[i], false, false, true );
        }

        timer = 0.0f;
    }
}
```

Figure 29: Update Traffic Light Code Snippet

As can be seen in the following figure, each traffic light within the application is assigned a box collider a few meters back from the traffic light. This is to help simulate traffic behaviour. Attached to each of these colliders is a TrafficLightStopTrigger script which stops automotive cars if the traffic light the box collider is associated with is red or orange.

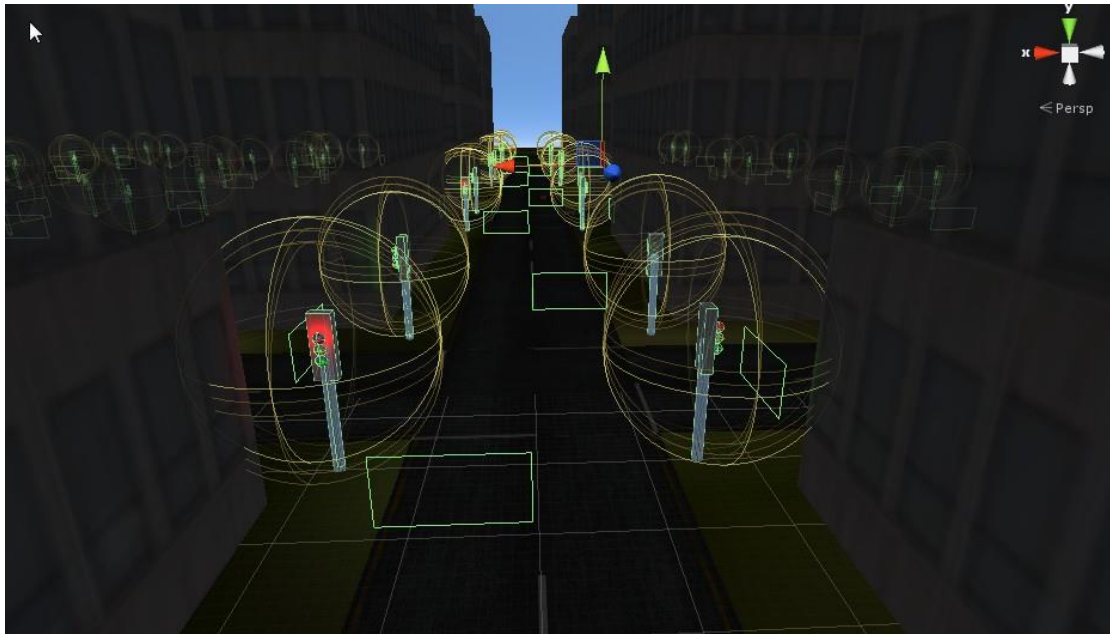


Figure 30: Traffic Light Box Collider Screenshot

4.3.6 Development Challenges

Developmental challenges occurred throughout the project lifecycle. But these challenges did not slow down the process of creating the application for very long. Unity is well documented, and it has a very large user base. Almost every problem which could occur has occurred already and has been documented, usually with solutions. Given enough time and resources there is no problem which cannot be solved eventually.

5 System Validation

5.1 Introduction

Testing is a vital part of system validation and must be completed before introducing a final piece of software to a user. Testing a system ensures that every feature is functional, efficient and easy to use. Without testing a piece of software, multiple bugs which could render the system unusable could not be accounted for. Testing which was carried out on this application includes unit testing, module testing, black box testing and usability testing. An expert evaluation of the application is also discussed in this chapter.

5.2 Unit Testing

Unit testing, also called white box testing involves validating smaller blocks of code in a system. The code and its inputs are manipulated to test specific internal workings of a project. Unit testing was performed throughout the development life cycle of this project. Each time a new functionality was created a unit test was implemented before it was placed in the final product.

A unit test was carried out on each new build function written for the application. The tests sometimes indicated errors in code logic. Sometimes an object would not be created and sometimes an object would be scaled incorrectly or have an unexpected rotation. To give an example, it was found during testing that you have to scale and rotate a parent game object after all children game objects have been scaled and rotated. If you don't, the scale and rotation of the child game objects multiply by the scale and rotation of the parent game object, causing the geometry of child game objects to skew in an undesirable fashion.

5.3 Module Testing

Module testing, also known as Integration Testing, tests large blocks of code to evaluate if interfaces between different parts of a systems are communicating properly with each other.

Module testing was carried out on the application's scenes close to the end of the project lifecycle. Every scene within the application was tested to ensure the different scenes could communicate properly with each other. The relationships between the scenes can be seen in the scene layout figure (see *Figure 9: Scene Layout*). This ensured that the application would not crash unexpectedly when a particular scene was chosen. This test yielded positive results and no fixes were required.

5.4 Black Box Testing

Black box testing, also known as system testing, involves testing the system as a whole. Generally this is done by creating different paths through the system which corresponds to the paths that a final user might take. Testing in this manner validates a systems functionality as a whole.

Black box testing was carried out on the application when development was completed. Hypothetical paths and actions that the final user could take were written out on paper and were then implemented within the application. This test yielded a surprising result. A script which had been working previously had stopped working and was causing the automotive cars to crash into the player car. This happened while the player car was waiting at a traffic light, as expected. The issue was tracked down to a line in the application which was commented out previously for debugging purposes.

5.5 Usability Testing

Usability testing is concerned with how easy and intuitive a system is to use. Test users are given software and are observed to see how they go about using it. Often they are also given a survey to fill out afterwards rating the system on different criteria.

Usability testing for the application was carried out on seven students from the Dublin Institute of Technology. They were given the application with all the hardware set up and were asked to interact with the application. Afterwards they were asked to fill out a questionnaire, which comprised of nine questions. The survey questions and answers are listed in the table below.

Question:	Overall Score
1: Can you drive?	42%
2: Do you own a full driving license?	10%
3: On a scale of 1 - 10, how easy was it to figure out how to use the application?	7.9
4: On a scale of 1 - 10 how enjoyable was the experience?	8.7
5: On a scale of 1 - 10, how realistic was the experience?	5.1
6: On a scale of 1 - 10, how likely would you be to use this application as an aid to learn driving?	7.14
7: Did you have any problems using the application? Please describe them.	See usability testing conclusion.
8: Apart from adding a more realistic car and more rules of the road, could you improve the application? If so how?	See usability testing conclusion.
9: Please rate the application on a scale of 1 - 10?	7.9

Table 2: Survey Questions and Answers

5.5.1 Usability Testing Conclusions

The answers received to question seven regarding problems using the application revealed that the steering wheel sensitivity was too high for everybody who used the application. This was attributed to the steering wheel controller provided for the simulation. The wheel only had ninety degrees of rotation and over-turning was an issue because of this. A steering wheel with more rotation, nine hundred degrees of rotation for example, would solve this issue and would be a necessary change for a final product distribution. Another problem people mentioned was shifting between the forward and reverse gears. They found it slow and unresponsive. This was noted and added to the future work section.

A problem which was not mentioned by the test subjects but was observed was that people didn't know what to do when they started the game initially. Most had never used the Rift before and did not know how to interact with the main menu by using the rift as a pointing device. To help fix this problem a help screen, explaining how to interact with the application needs to be displayed to the user before the main menu is

displayed. This menu could be skipped if the user knows how to interact with the application. This has been added to the future work section. The biggest problem observed was that nobody took the application seriously and tried to crash into as many different obstacles as they could with no apparent desire to learn how to drive. Too much freedom was given to the user and the application's core fundamentals were abused. A strict set of rules, tasks and challenges needs to be implemented to prevent the user from deviating from the goal of learning. This task has been added to the future work section.

The answers received from question eight regarding possible improvements for the application revealed that the test subjects wished for different terrain physics, different driving environments, a virtual driving instructor, rear view mirrors, more realistic car physics, pedestrians and realistic driving experiences such as bouncing. All of these, excluding simulating bouncing have been added to the future work section.

5.6 Expert Evaluation

Getting the opinion and advice of experts is a vital part of creating any application. Experts have the experience, they know what the public want and they know what you need to add. Gaining insight and advice from an expert is an invaluable asset.

Expert indie game developers from Batcat games and bitSmith games were contacted and the application was expertly evaluated. All comments suggested that the idea behind the virtual reality driving tutor simulation was a good, but the application needed more functionality as well as code fixes. It was also suggested that the application could be turned into a hit and run type game as opposed to an eLearning driving application.

6 Project Plan Analysis

As with all projects carried out using an agile software development methodology such as DSDM, changes throughout this project were inevitable. A flexible approach means that the project was still completed in the time available and with most of the required functionality although it followed a different plan to the one set out at the beginning of the project.

6.1 The Original Plan

The following is the original plan as defined in the Project Proposal Document.

October 2012:

- Start the Requirements Specifications phase and Software design phase thereafter. Research technologies required and specify the architecture of my project. Create use case and class diagrams
- Update Final Year Project Report.

November 2012:

- Complete the Software Design phase if not already finished.
- Make the interim report and interim presentation.
- Update Final Year Project Report.

December 2012:

- Hand in the interim report and present the interim presentation.
- Implementation and integration phase. Start coding and develop a test plan to use later in the testing phase.
- Update Final Year Project Report.

January 2013:

- Continue the implementation and integration phase.
- Update Final Year Project Report.

February 2013:

- Move onto the testing phase. Feedback surveys will be carried out on drivers and non-drivers who have tested the application. Black box testing to ensure functionality is correct. Test plans created from the implementation and integration phase will be used here.
- Update Final Year Project Report.

March 2013:

- Deployment and Maintenance phase. Deploy code and if any errors are found fix them or known issue them.
- Finish Writing the Final Year Project Report.

April 2013:

- Create project presentation
- Final Checks and finishing touches on Project, Project Report and Project Presentation.

6.2 The Actual Plan

The plan stayed the exact same as the original plan up until December, when the Interim report and presentation were handed in. It is difficult to attribute values to the actual plan after this as the plan changed multiple times throughout the project lifecycle.

6.3 Reasons for Changes

The main factors in causing the project plan to change were time and complexity. Originally it was assumed that the scope of the project was reasonable and could be completed in the time given. However, for various reasons, that was not the case.

Generally, the project was too ambitious. Although the various functionality was defined at the beginning, the implementation was more involved than expected. The core functionality required more time than anticipated, with fundamentals such as world generation and traffic lights taking longer. As a result, extra features such as mirrors and a driving instructor as well as the full list of initially defined driving scenes were not implemented. The project report was also not updated regularly as planned due to time constraints.

A certain amount of work was planned for December and January but time taken up by examinations was not accounted for. As a result, over a month of time was deducted from expected project work.

Overall the ratio of time available to work necessary was much more skewed than originally anticipated, which resulted in fewer features in the final software as well as a mad dash at the end of the project to get everything finished on time.

6.4 Future Prevention of Problems

Looking back on the various reasons that the project got derailed, there are some measures that could be put in place to prevent the same shortcomings in future projects.

Development should have begun much earlier than it did. Not only would this allow for more features to be implemented, the large scope would have been realised earlier and the project could have been redefined to suit the constraints. This also would have somewhat compensated for the time lost around examinations.

More research into the lower level development required for certain features such as the traffic lights could have been done closer to the start of the project. In doing this, a more thorough knowledge of what was involved in various aspect of implementation would have been achieved and features may have taken less time to complete.

In general, a more proactive approach to design and time management would have resulted in a more smooth and effective project lifecycle.

7 Evaluation, Conclusion and Future Work

7.1 Evaluation

Evaluation can be done against the objectives initially set out in this project.

- Research which game engines support the Oculus Rift:
 - ➔ This objective was achieved. The results concluded that many game engines supported the rift and three of the most popular game engines on the market supported it. This objective allowed for a choice to be made on which game engine should be used.
- Research which game engine is best for this project:
 - ➔ This objective was achieved. The results concluded that Unity was the best game engine for the application to be built on. This objective resulted in the most effective application being built with the least amount of stress.
- Research what driving is like and the rules of the road:
 - ➔ This objective was achieved. The Irish learner driver theory test was taken and passed. Also Practicing driving in a parking lot took place gaining an inside view into what driving was like. This research allowed information and experiences to directly affect the application. It also showed that there were more rules of the road than could possibly have been implemented in the project timeline.
- Learn how to develop and create applications on the chosen engine:
 - ➔ This objective was accomplished before any development work took place on the application. This objective was important as the Unity environment needed to be understood before development could commence.
- Create a moveable car which the user can control via a steering wheel and pedals:
 - ➔ This objective was completed although the result was not as realistic as it could have been. The friction between the road and the wheel could have been more

realistic. The motor torque and brake torque applied to the cars WheelColliders could have been more accurate. The ratio between the shifting gears could have been more accurate. It would have been a project in and of itself creating a fully realistic car inside of Unity.

- Create the driving environment:
 - ➔ This objective was accomplished but the environment could have been more realistic. Artists and 3D modellers would need to be employed to make the art assets better in this application.
- Create game logic:
 - ➔ This objective was completed to a certain degree. There is no real rhyme or reason to the different driving scenarios. The user is given full authority to do whatever they want. There needs to be a more structured learning process implemented in the game logic.
- Create different driving scenarios:
 - ➔ This objective was completed. Three different driving scenarios were created which the user can choose from. The only thing which could be changed is to make it easier to navigate in between the driving scenarios. Creating a more structured learning process would achieve this goal.
- Integrate the Oculus Rift into the application:
 - ➔ This objective was completed. Full rift support is offered by the application. Achieving this goal produced the end goal of this application, to create a virtual reality driving tutor simulation.
- Create a user interface for the application:
 - ➔ This objective was completed. Achieving this goal allows the user to navigate between the different driving scenes within the application.
- Test the completed application:

- ➔ This objective was completed. The application was fully tested using multiple different testing techniques. The results of this objective uncovered useful data which would not have been spotted if this task was not carried out.

7.2 Conclusion

To conclude, a virtual reality driving tutor simulator has been created. The objectives set out in the beginning of the project were achieved and this project can be deemed a success. The application as it stands is nowhere near a final product but is a big step in the right direction.

Technically speaking a complete and working system has been created. The application is not perfect in regards to visual effect or efficiency, it does however successfully perform the tasks it was originally set out to do.

With regards to project management, development could have gone a lot smoother. A realization has occurred that very rarely does a project go according to plan and contingency time should be allotted for such occurrences.

7.3 Future Work

If this project was being developed further a multitude of fixes and updates would be applied to the project in its current form before additional functionalities would be implemented.

List of required fixes:

- Fix the reverse gear toggle.
- Make the menu easier to navigate, make it more intuitive.
- Make the car physics more realistic

The list of additional functionalities which can be added to the application is endless. The following is a list of the more important features which would be implemented first:

- Create a more realistic car, with driver seats, rear-view mirrors, speedometer, steering wheel, revolution counter, etc.
- Add different types of terrain physics. Have hills and bumps on the roads.

- Add different weather, e.g., rain or snow.
- Add driving scenes which deal with dangerous driving conditions such as black ice or a site of a car crash.
- Add lessons on learning the rules of the road.
- Implement a virtual driving instructor.
- Implement a strict set of tasks, rules and guidelines for the user to follow when learning how to drive. The user must not be allowed deviate from the learning activities.
- Implement animation to cars in the application.
- Add functionality for a manual transmission car.
- Introduce functionality to allow a physical instructor to interact with the application as it is running. Allow placing of cars and obstacles at runtime.
- Create own game engine and implement scripts used in this application.

Appendix A

```
//Builds object based off words entered from the text file
private void BuildObject( string[] entries )
{
    string objectName = entries[0];
    objectPosition = new Vector3( float.Parse( entries[1] ),
                                   float.Parse( entries[2] ),
                                   float.Parse( entries[3] ) );
    objectScale = new Vector3( float.Parse( entries[4] ),
                               float.Parse( entries[5] ),
                               float.Parse( entries[6] ) );
    objectRotation = new Vector3( float.Parse( entries[7] ),
                                   float.Parse( entries[8] ),
                                   float.Parse( entries[9] ) );

    //Different objects to build
    switch ( objectName.ToLower() )
    {
        //Skybox
        case "skybox":
            BuildSkybox();
            break;

        //Main Menu
        case "main_menu":
            BuildMainMenu();
            break;

        //How To Play:
        case "how_to_play":
            BuildHowToPlayMenu();
            break;

        //Road parent
        case "road_parent":
            roadParent = new GameObject();
            SetParentDetails( roadParent, "RoadParent", 0, null );
            break;

        //Straight road parent
        case "straight_road_parent":
            straightRoadParent = new GameObject();
            SetParentDetails ( straightRoadParent, "StraightRoads",
                               straightRoadCounter, roadParent );

            break;

        //Sideways road parent
        case "sideways_road_parent":
            sidewaysRoadParent = new GameObject();
            SetParentDetails ( sidewaysRoadParent, "SidewaysRoads",
                               sidewaysRoadCounter, roadParent );

            break;

        //Junction parent
        case "junction_parent":
            junctionParent = new GameObject();
            SetParentDetails ( junctionParent, "Junctions",
                               junctionCounter, roadParent );
    }
}
```



```

        break;

//Buildings parent
case "buildings_parent":
    buildingsParent = new GameObject();
    SetParentDetails ( buildingsParent, "Buildings",
                        buildingCounter, null );

    break;

//Static car parent
case "static_car_parent":
    staticCarParent = new GameObject();
    SetParentDetails ( staticCarParent, "StaticCars",
                        staticCarCounter, null );

    break;

//Automotive car parent
case "automotive_car_parent":
    automotiveCarParent = new GameObject();
    SetParentDetails ( automotiveCarParent, "AutomotiveCars",
                        automotiveCarCounter, null );

    break;

//Trafficlight parent
case "traffic_light_parent":
    trafficLightParent = new GameObject();
    SetParentDetails ( trafficLightParent, "TrafficLights",
                        0, null );

    break;

//Straight trafficlight parent
case "straight_traffic_light_parent":
    straightTrafficLightParent = new GameObject();
    SetParentDetails ( straightTrafficLightParent,
                        "StraightTrafficLights",
                        straightTrafficLightCounter,
                        trafficLightParent );

    break;

//Reverse trafficlight parent
case "reverse_traffic_light_parent":
    reverseTrafficLightParent = new GameObject();
    SetParentDetails ( reverseTrafficLightParent,
                        "ReverseTrafficLights",
                        reverseTrafficLightCounter,
                        trafficLightParent );

    break;

//Left trafficlight parent
case "left_traffic_light_parent":
    leftTrafficLightParent = new GameObject();
    SetParentDetails ( leftTrafficLightParent,
                        "LeftTrafficLights",
                        leftTrafficLightCounter,
                        trafficLightParent );

    break;

//Right trafficlight parent
case "right_traffic_light_parent":
    rightTrafficLightParent = new GameObject();
    SetParentDetails ( rightTrafficLightParent,

```

```
        "RightTrafficLights",
        rightTrafficLightCounter,
        trafficLightParent );

    break;

//Right trafficlight parent
case "path_parent":
    pathParent = new GameObject();
    SetParentDetails ( pathParent, "PathParent",
        pathPointCounter, null );

    break;

//Plane
case "plane":
    BuildPlane();
    break;

//Straight Road
case "straight_road":
    BuildStraightRoad();
    break;

//Sideways Road
case "sideways_road":
    BuildSidewaysRoad();
    break;

//Junction
case "junction":
    BuildJunction();
    break;

//Building 1
case "building":
    BuildBuilding();
    break;

//Player Car
case "player_car":
    BuildPlayerCar();
    break;

//Static Car
case "static_car":
    BuildStaticCar();
    break;

//Automotive Car
case "automotive_car":
    BuildAutomotiveCar();
    break;

//Straight Traffic Light
case "traffic_light_straight":
    BuildTrafficLightStraight();
    break;

//Left Traffic Light
case "traffic_light_left":
    BuildTrafficLightLeft();
    break;
```

```
//Right Traffic Light
case "traffic_light_right":
    BuildTrafficLightRight();
    break;

//Reverse Traffic Light
case "traffic_light_reverse":
    BuildTrafficLightReverse();
    break;

//Path
case "path":
    BuildPath();
    break;

//Default Value
default:
    break;
}
```

Bibliography

- [1] G. C. Burdea and P. Coiffet, "Virtual Reality Technology - Introduction," in *Virtual Reality Technology*, 2nd Edition., John Wiley & Sons, Inc, 2003, p. 444.
- [2] "Oculus Rift - Virtual Reality Headset for 3D Gaming | Oculus VR." [Online]. Available: <http://www.oculusvr.com/>. [Accessed: 01-Dec-2013].
- [3] "RSA.ie - An Introduction to EDT." [Online]. Available: <http://www.rsa.ie/en/RSA/Learner-Drivers/Driver-Training/Car-Training-EDT/An-Introduction-to-EDT/>. [Accessed: 07-Dec-2013].
- [4] "Learner drivers must be accompanied at all times." [Online]. Available: <http://www.rsa.ie/Templates/RSA/Pages/ContentPage.aspx?id=416&epslanguage=en>. [Accessed: 31-Mar-2014].
- [5] M. Ma, L. C. Jain, and A. Oikonomou, *Serious Games and Edutainment Applications*. Springer, 2011.
- [6] S. A. Schroeder, "Adopting Game Technology for Architectural Visualization."
- [7] "Unity - System Requirements." [Online]. Available: <http://unity3d.com/unity/system-requirements>. [Accessed: 08-Dec-2013].
- [8] "Readme | Unreal Technology - System Requirements." [Online]. Available: <http://www.unrealengine.com/udk/documentation/readme/#system-requirements>. [Accessed: 08-Dec-2013].
- [9] "• CryEngine 3 SDK — Requirements, How To Install, Documentation « Game Engines: CryEngine 3 SDK •." [Online]. Available: <http://3dg.me/game-engines/cryengine-3-sdk/cryengine-3-sdk-requirements-how-to-install-documentation>. [Accessed: 08-Dec-2013].
- [10] "Official Oculus Rift VR Forums | Oculus Rift Developer Forums." [Online]. Available: <https://developer.oculusvr.com/forums/>. [Accessed: 08-Dec-2013].
- [11] "► Making Deliveries in Eurotruck Simulator 2 with the Oculus Rift! - YouTube." [Online]. Available: <http://www.youtube.com/watch?v=ipbtMJagv5o>. [Accessed: 08-Dec-2013].
- [12] "► DriveON ® cross - YouTube." [Online]. Available: <http://www.youtube.com/watch?v=NhgT1q497Us>. [Accessed: 08-Dec-2013].
- [13] "2009-ITS-America.pdf."
- [14] T. Lorentzen, Y. Kobayashi, and Y. Ito, "VIRTUAL REALITY DRIVING SIMULATION."
- [15] "News about iRacing.com's online racing | iRacing.com." [Online]. Available: <http://www.iracing.com/category/press-coverage-news/>. [Accessed: 08-Dec-2013].
- [16] S. Bennet, S. McRobb, and R. Farmer, "Object-Oriented Systems Analysis and Design using UML - Chapter 21," in *Object-Oriented Systems Analysis and Design using UML*, 3rd Edition., McGraw-Hill Companies, p. 698.
- [17] "What are the minimum system requirements for the Oculus Rift developer kit? : Oculus VR Support." [Online]. Available: <https://support.oculusvr.com/entries/24794452-What-are-the-minimum-system-requirements-for-the-Oculus-Rift-developer-kit->. [Accessed: 03-Dec-2013].
- [18] "Unity - Scripting - Script your gameplay in a world-leading programming environment." [Online]. Available: <http://unity3d.com/unity/workflow/scripting>. [Accessed: 04-Apr-2014].
- [19] "Boo, C# and JavaScript in Unity - Experiences and Opinions." [Online]. Available: <http://forum.unity3d.com/threads/18507-Boo-C-and-JavaScript-in-Unity-Experiences-and-Opinions>. [Accessed: 03-Dec-2013].

- [20] "Car Tutorial by Unity Technologies -- Unity Asset Store." [Online]. Available: <http://u3d.as/content/unity-technologies/car-tutorial/1qU>. [Accessed: 03-Dec-2013].
- [21] "How to make a physically real,stable car with WheelColliders." [Online]. Available: <http://forum.unity3d.com/threads/50643-How-to-make-a-physically-real-stable-car-with-WheelColliders>. [Accessed: 01-Nov-2013].
- [22] J. Arndt, "Creating a driveable vehicle in Unity 3D - YouTube." [Online]. Available: <http://www.youtube.com/watch?v=21zuMIsy2GM>. [Accessed: 31-Oct-2013].
- [23] "[Script] Test your Rift inside the Unity Editor | Unity 4 Integration | Oculus Rift Developer Forums." [Online]. Available: <https://developer.oculusvr.com/forums/viewtopic.php?f=37&t=2456>. [Accessed: 03-Dec-2013].
- [24] "Unity Scripting Reference:" [Online]. Available: <http://docs.unity3d.com/Documentation/ScriptReference/>. [Accessed: 31-Oct-2013].
- [25] "Unity - Text Mesh." [Online]. Available: <http://docs.unity3d.com/Documentation/Components/class-TextMesh.html>. [Accessed: 05-Apr-2014].
- [26] "Unity Script Reference - Wheel Colliders." [Online]. Available: <http://docs.unity3d.com/Documentation/ScriptReference/WheelCollider.html>. [Accessed: 31-Oct-2013].