Contents

1. Abstract

Erie Insurance currently works with its agents to help them display the dangers of distracted driving to their policyholders. This can often be very difficult for agents to do since the user is not able to experience the consequences of distracted driving for themselves in a safe way. In order to help solve this problem for the agents, we are creating a virtual reality experience to allow for better engagement between agents and the teen drivers they work with. This virtual reality experience will utilize the Unity 3D engine and the Google Cardboard SDK to give the policyholder different scenarios in which they will have to make decisions in which they will have to prevent the driver from texting. This virtual reality experience will help the policyholder to understand how they can influence dangerous driving activities as well as to help stop them. The overall goal of this virtual reality experience is to give young drivers a fun, memorable experience with their agent and to help encourage them not to drive while distracted.

2. Report Revision History

2.1 Changes in Version 1.5

In this version, we have made the changes recommended to us by our advisor. We have added a new user requirement and functional requirement detailing more information regarding the specific tasks that the AI driver should perform. The use case mapping diagram has updated as well. Along with that, we have changed the name of our use case “Begin Experience” to “Experience Loop” to make more sense. References are now available and are used in section 5 to further explain our exploratory studies.

2.2 Changes in Version 2.0

In this version, we have added our initial designs for the architecture, structure, interface, and behavior of the system. We have changed our architecture to the component-based architecture, which more accurately captures the way Unity objects build off each other to create the overall system. We have added and updated our requirements based off feedback from advisors and industry mentor. We have created test cases for our system, as well as the execution history. We have added the steps to set up the development environment and testing environment, build for the target platform, and install to the end user’s device.

2.3 Changes in Version 2.5

In this version, we have made further changes recommended to us by our faculty advisor and industry mentor. We have modified the layout of the report in section 6.2 to better organize the descriptions that go with each individual image. We have also updated section 6.3 to be contained within one page for further formatting improvements.

2.4 Changes in Version 3.0

In this version, we have modified our requirements and descriptions to match the new direction that the project is taking. We are going with a less realistic and serious approach, and the new information reflects these changes. We have also edited the report so that it now reflects the desired report structure specified on the Capstone project management system.

3. Problem Statement

3.1 Business Background

Erie Insurance is a Fortune 500 insurance company employing thousands of people. Erie Insurance has been a figure in the insurance world for 90 years, and currently serves over 4 million customers in 13 states. They utilize and manage smaller agencies to deal directly with customers, selling them auto, home, life, and business insurance.

With the rise of technology, distracted driving has become more of a risk than ever before. As Erie Insurance invests in protecting people, they are taking the initiative in informing families about the dangers of driving while distracted.

3.2 Needs

Currently, it is very difficult to display the dangers of distracted driving to a younger generation in a way that engages them. Erie Insurance is seeking an innovative solution in order to solve this problem. The business sponsors of this project are interested in a product that allows for an open discussion on the dangers of texting and driving. While this project is not aimed at teaching policyholders to refrain from texting and driving, it will be used to help the agent share something fun while having that discussion with the young driver. The business sponsors need a solution to the problem of getting young drivers to remember this discussion. This project will also encourage policyholders to share their experiences with their agent by discussing the project with others.

3.3 Objectives

This project aims to utilize virtual reality technology to create an immersive experience that engages users of all ages. The application will be distributed to agents around Erie's footprint and will effectively capture the younger audience. The main objective of this project is to meet the needs of our industry partner, Erie Insurance, which are explained above. The business sponsors of the project need a way to help young drivers remember the dangers of distracted driving that have been explained to them by their Erie Insurance agent, which is a gap that this project aims to fill.

4. Requirements

4.1 User Requirements

4.1.1 Glossary of Relevant Domain Terminology

Virtual Reality (VR) – A simulation of a three-dimensional environment

Cardboard – Google's SDK created for smartphone devices

Headset – A head mounted device that displays virtual reality devices

Scene – A Unity scene is an aggregation of components that can be executed on its own

4.1.2 User Groups

User – Any person engaging in our experience

4.1.3 Functional Requirements

4.1.3.1 Project Scope (Use Case Diagram)

Figure 4.1 displays the system's use case diagram. This gives a layout of the main user interactions that can occur as they use the system. The VR – Texting and Driving application consists of two main use cases: Experience Loop and Solve Scenario. The user will navigate through the experience and solve scenarios which will result in a final outcome at the end of the experience.

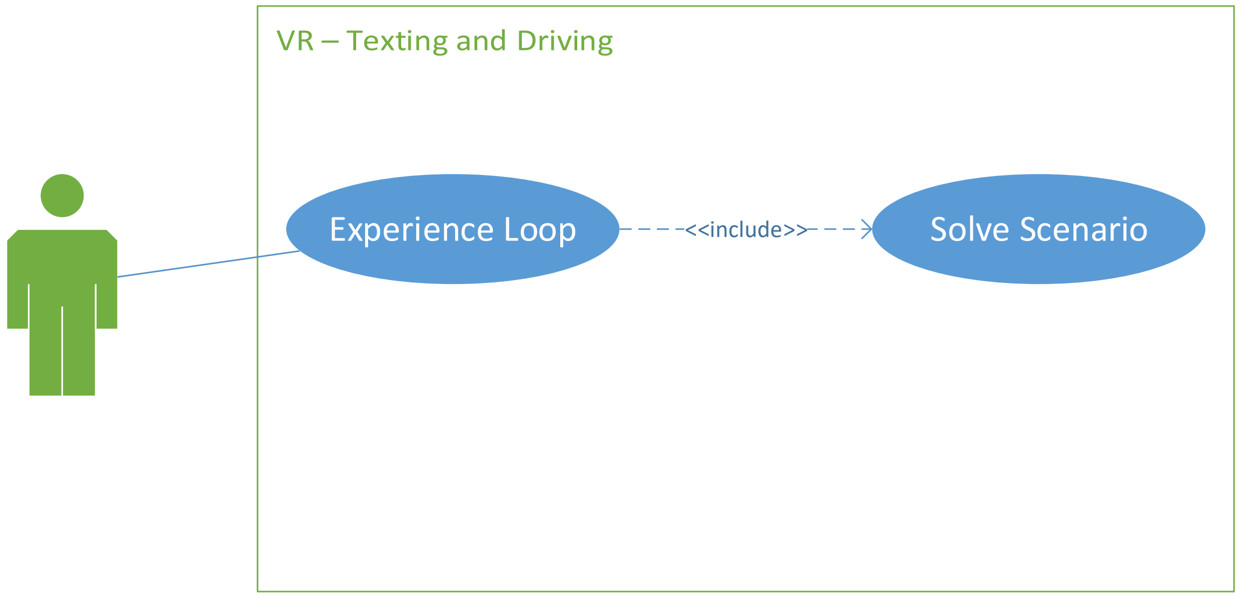

Figure 4.1 - Use Case Diagram

4.1.3.2 User Scenarios

Table 4.1 lists the details of the use cases that occur within the system. The use cases give an overview of the sequence of the interactions that occur with the user and the system. The scenario list consists of Experience Loop and Solve Scenario, which explained in greater detail in Table 4.2 and Table 4.3, respectively.

4.1.3.3 List of User Functional Requirements

User functional requirements describe functionality that the system should provide. The following will provide context to each of the system's user functional requirements.

Table 4.4 describes that the application should present a variety of different scenarios to the user. These scenarios should all feature a distracted driver. The user should be able to overcome the potential negative effects that may occur due to the carelessness of the driver.

Table 4.5 describes the main action of the user, which is to control the camera in the front passenger seat of the vehicle. The passenger will be in control of himself the entire time while the driver navigates through the various scenes.

Table 4.6 describes that the system should feature multiple outcomes that result from the actions of the driver influenced by the user. Similar to Table 4.4, there should be multiple types of scenarios as well as multiple types of outcomes resulting from those scenarios.

Table 4.7 describes that the user should be able to interact with the environment while within the experience loop. This may include things such as interacting with items in the car.

Table 4.8 describes that an AI will control the car driving. This AI will take the passenger's decisions into consideration while he engages in various tasks throughout the duration of the experience.

4.1.4 Non-functional Requirements

Non-functional requirements describe the constraints and quality of the functionalities, providing testable features and specifying restrictions.

4.1.4.1 Product: Usability Requirements

Usability requirements describe how easily a user interacts with the system.

4.1.4.2 Product: Performance Requirements

Performance requirements describe how well a system performs in terms of time and resource usage.

Table 4.9 describes that the system should run at an acceptable frame rate for virtual reality use. This is to prevent users from becoming ill during their time in the experience.

4.1.4.3 Product: Dependability/Security Requirements

Dependability/Security requirements describe the reliability and security concerns of the project.

4.1.4.4 Organizational: Development Requirements

Development requirements specify development practices and constraints.

Table 4.10 describes that the VR – Texting and Driving application should be designed with modern Android devices in mind. Virtual reality needs this to support our system and keep it running at an acceptable frame rate.

Table 4.11 explains that we developed the system for Google Cardboard. There are currently many VR headsets on the market, however, Erie Insurance would like to keep access to the application relatively cheap.

Table 4.12 states that the system must feature Erie Insurance logos throughout the duration of the experience. Erie Insurance would like to clearly show off their brand while keeping it within the context of the experience.

4.1.4.5 Organizational: Operational Requirements

Operational requirements describe conditions that a system must support.

4.1.4.6 Organizational: Environmental Requirements

Environmental requirements describe the look and feel of the system's interface.

4.1.4.7 External: Safety/Security Requirements

Safety/Security requirements detail how the system will interact with other systems, and the security concerns of these interactions.

4.1.4.8 External: Cultural and Social Requirements

Cultural and social requirements describe how the system conforms to cultural and social expectations.

4.1.4.9 External: Political Requirements

Political requirements detail how the system will influence different sections of the company.

4.2 System Requirements

User requirements tend to be vague, so they are refined into system requirements. System requirements engineer and refine the user requirements into many detailed requirements that are much more descriptive and implementable.

4.2.1 Functional Requirements

4.2.1.1 List of System Functional Requirements

Table 4.13 describes that the system should feature two possible scenarios for every decision presented to the user. This further defines the user requirement for use in the system.

Table 4.14 further discusses how the user views a scenario. Each scenario will have its own respective trigger to begin the scenario.

Table 4.15 describes the view and input that the user has during the experience. The user will have a first person view at all times, in which the user is able to move their head around to rotate the camera.

Table 4.16 explains that the user will be able to use on input device on the Google Cardboard, which is a single button. This button allows the user to interact with their environment, and will allow the user to make decisions for each scenario presented to them.

Table 4.17 explains how to solve the problem of the application beginning in the incorrect position. Due to the incorrect positioning, the user may not be facing towards the front of the car. To solve this, a double click system allows the user to re-center the camera to face the front of the car.

Table 4.18 gives a more detailed look into what outcomes will possible in regards to scenarios. Outcomes that can occur include falling objects, avoiding collisions, and moving off the path.

Table 4.19 describes what the user should be able to interact with in between scenarios. These things include grabbing a drink from the cup holder, opening and closing the window, and adjusting the radio.

Table 4.20 further details what the AI driver should do while driving the car. This includes texting while driving, however, he should stop texting when the user looks in his direction.

4.2.1.2 System Behavior

Figure 4.2 details the sequence of flow between user and system. The use cases are covered by the diagram, which provides an in-depth depiction of the use cases for the system.

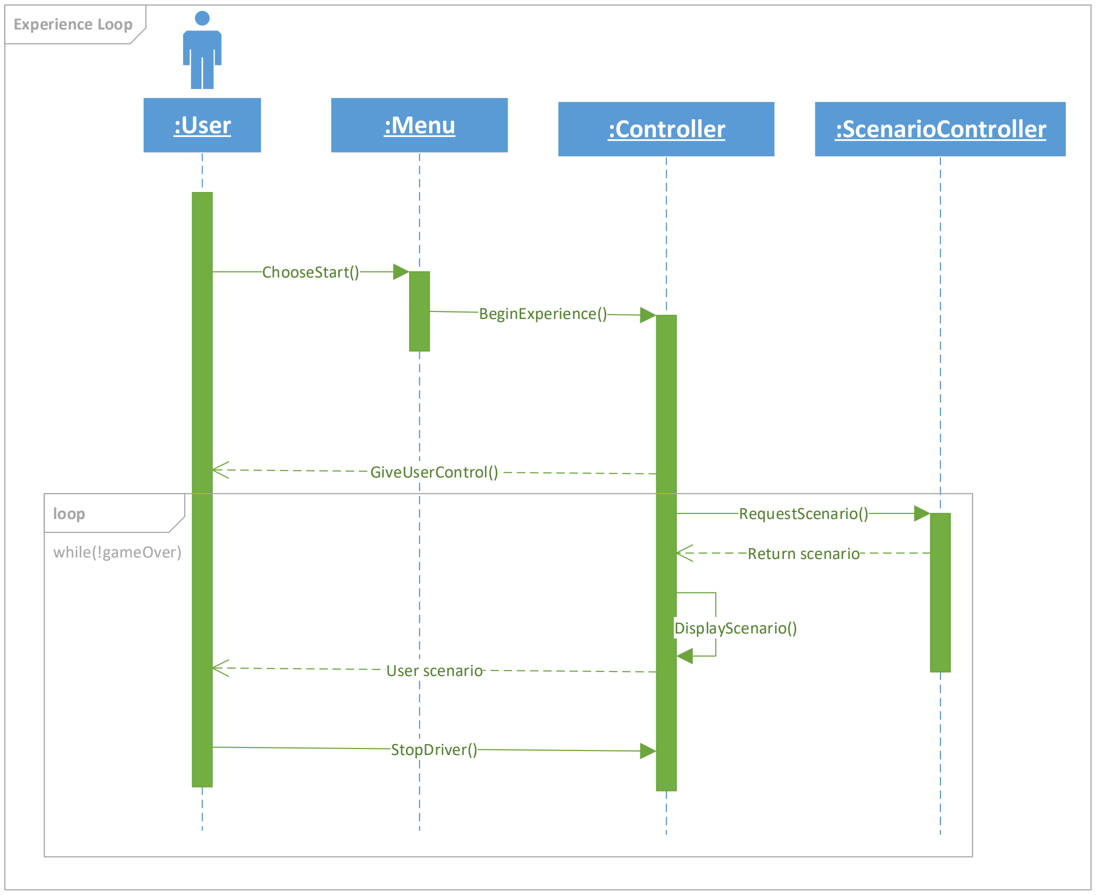

Figure 4.2 - Experience Loop Sequence

4.2.1.3 Data Requirements

N/A

4.2.2 Non-functional Requirements

4.2.2.1 Product: Usability Requirements

N/A

4.2.2.2 Product: Performance Requirements

Table 4.21 explains that the system should run at 30 frames per second. This will ensure a fairly smooth VR experience on a variety of hardware.

4.2.2.3 Product: Dependability/Security Requirements

N/A

4.2.2.4 Organizational: Development Requirements

Table 4.22 gives explicit detail as to which version of Android the application uses.

Table 4.23 discusses that the system will use features of the Google VR SDK to render the dual image to create the VR effect.

Table 4.24 explains in further detail how the environments feature Erie Insurance logos. Buildings, billboards, bumper stickers, and air fresheners may contain these logos.

4.2.2.5 Organizational: Operational Requirements

N/A

4.2.2.6 Organizational: Environmental Requirements

N/A

4.2.2.7 External: Safety/Security Requirements

N/A

4.2.2.8 External: Cultural and Social Requirements

N/A

4.2.2.9 External: Political Requirements

N/A

4.3 Requirements Trace Table

Table 4.25 gives a breakdown of the user requirements and system requirements mapping.

5. Exploratory Studies

5.1 Relevant Techniques

We will be using the Unity 3D game engine to create our application. We have chosen this engine because of its C# scripting, large community, and because it allows us to create an immersive VR experience very quickly. Along with Unity 3D, we will be using the Google VR SDK for Unity to adapt our project for VR use [6]. We also plan to take advantage of the Unity Asset Store to collect models, animations, and scripts to allow us to focus on implementing the requested features and not worry about having to create all of our assets from scratch. Within the Asset Store exists an important package called Unity Test Tools [4]. Unity Test Tools allows us various ways of testing including unit tests, integration tests, and assertion component to make sure our work is as bug free as possible. All of these technologies working together will allow us to create an experience that puts the user into the middle of a seemingly dangerous situation.

5.2 Relevant Packages/Products

The main products and packages we will be using include Unity 3D, Google VR SDK, a variety of assets from the Unity Asset Store, the Android SDK to build from within the Unity engine, Unity Test Tools to complete our application testing, Visual Studio for writing C# scripts, and potentially more as we move forward.

5.3 Broader Impacts

This virtual reality experience has the potential to help minimize distracted driving by providing a system to insurance agents that will help them better communicate with young drivers when explaining the dangers that come with distracted driving. Minimizing distracted driving means that there will be less accidents, less injuries, and less deaths because of distracted driving. Since the application runs on the Android operating system, which is used by millions of people every day, this application has the potential to reach a large number of drivers and passengers.

6. System Design

6.1 Architectural Design

The system will be using a component-based architectural design, which emphasizes the creation of components, which other components reuse to create a scene. Multiple scenes are sequenced together to create the overall system. Unity objects are a component that is self-contained, meaning that it can run on its own inside a scene. As objects are defined, they can be used in other objects to create large components that are combined to create complex scenes. Figure 6.1 shows our high-level architecture, which is consisting of a starting interface *GameObject* that has a composition with itself to allow the components to have other components that make it up.

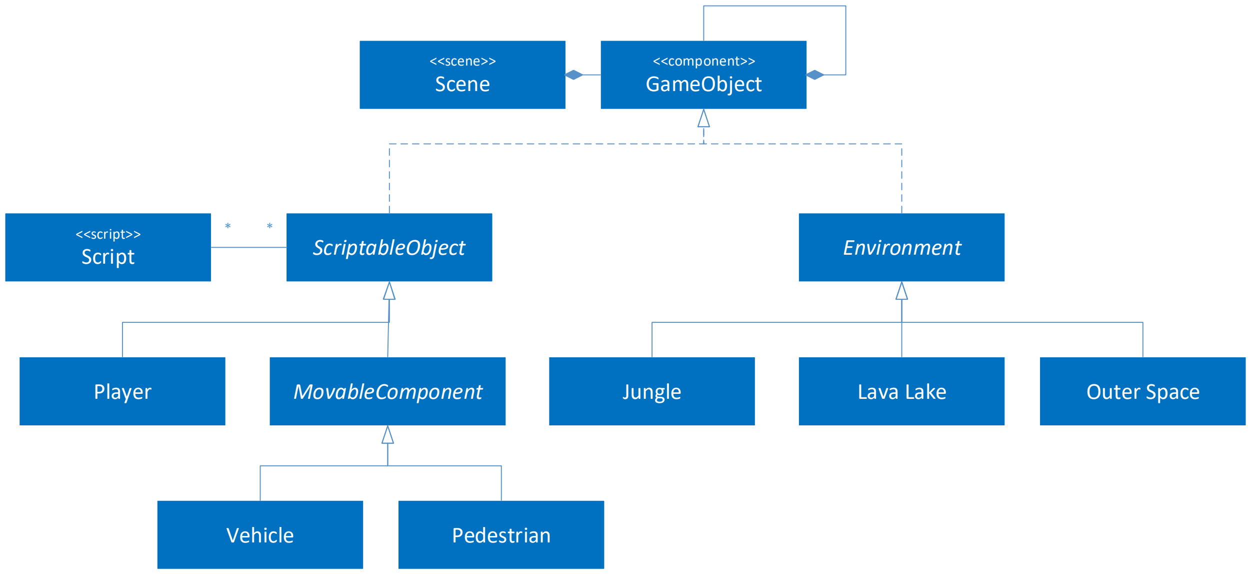

Figure 6.1 - Architectural Design

6.2 Structural Design

The structural diagram provides the detailed components that are defined in the architectural design. The basic components are refined into each individual component that create the overall layout of the Unity scene.

Figure 6.2 represents the Scenes package within the structural diagram. This package will contain each scene within the experience and show how they connect to each other. This package also contains the main GameObject interface that all other components will be inheriting from throughout the system.

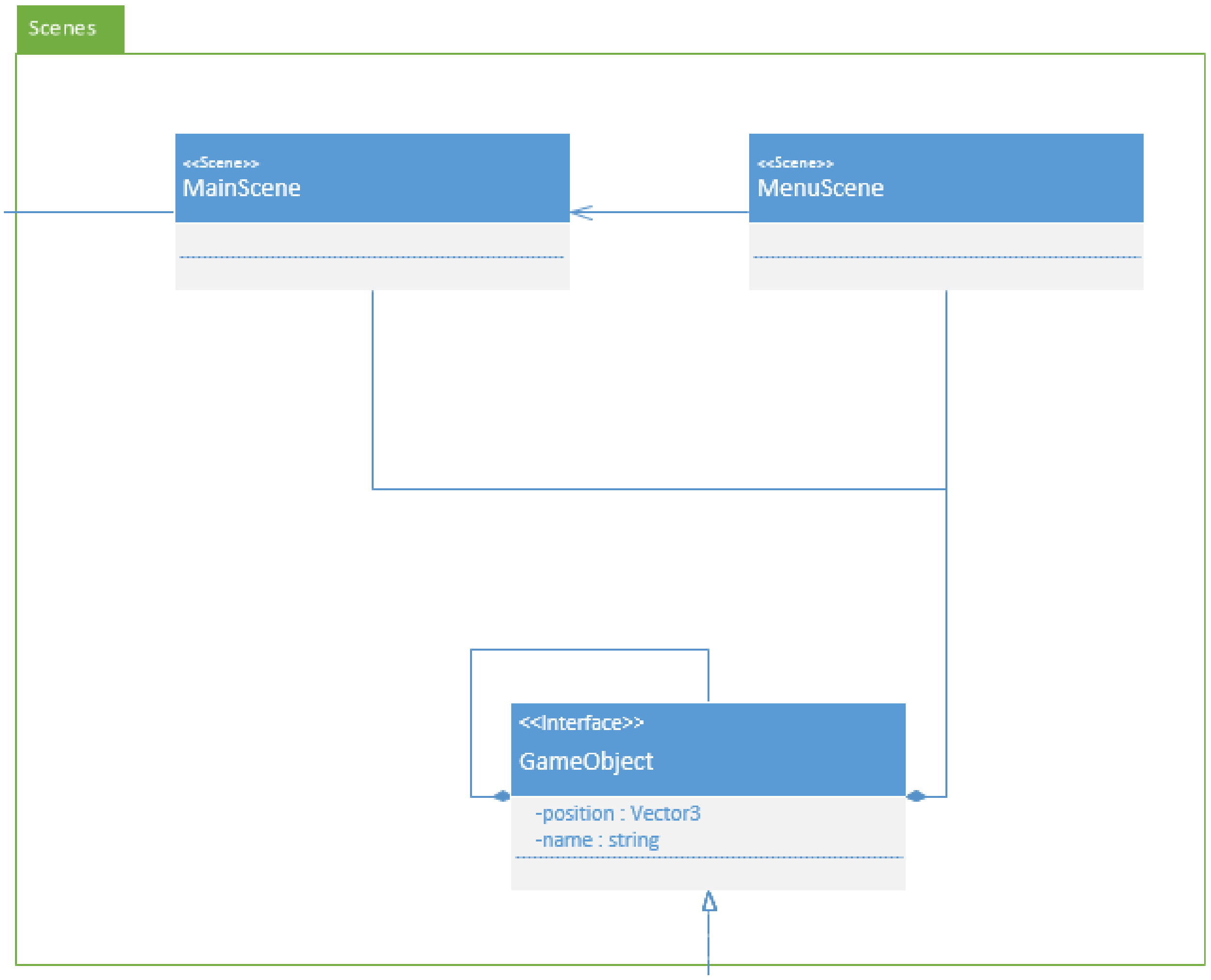

Figure 6.2 - Structural Diagram (Scenes)

Figure 6.3 is the package representing the player. The player is essentially a camera component using a player controller script to allow the user to move their head around to view and interact with what is happening in the scene. The player controller is able to perform actions with the environment such as reentering the user, which is shown below.

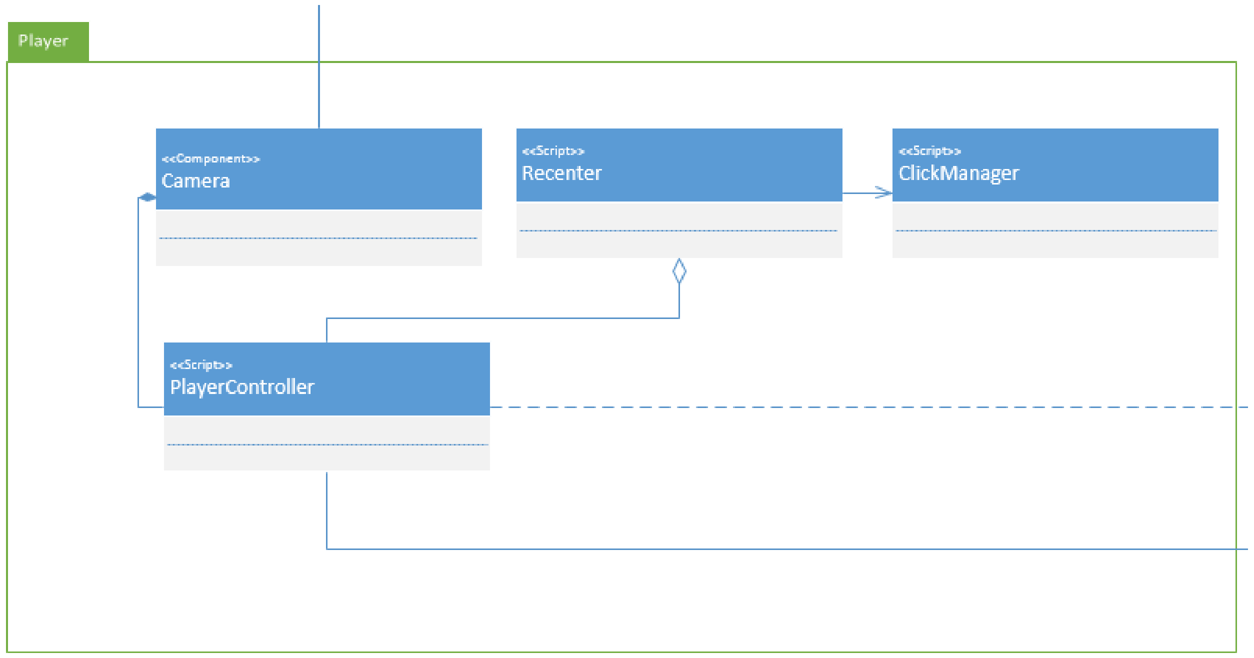

Figure 6.3 - Structural Diagram (Player)

Figure 6.4 shows many of the components that come from Google's VR SDK. As mentioned above, the player is a camera that is able to interact with the environment. To do this, the camera utilizes components, interfaces, and scripts in this package. This package allows components to be set as either objects causing interactions to happen or allows components to be the object that is interacted with.

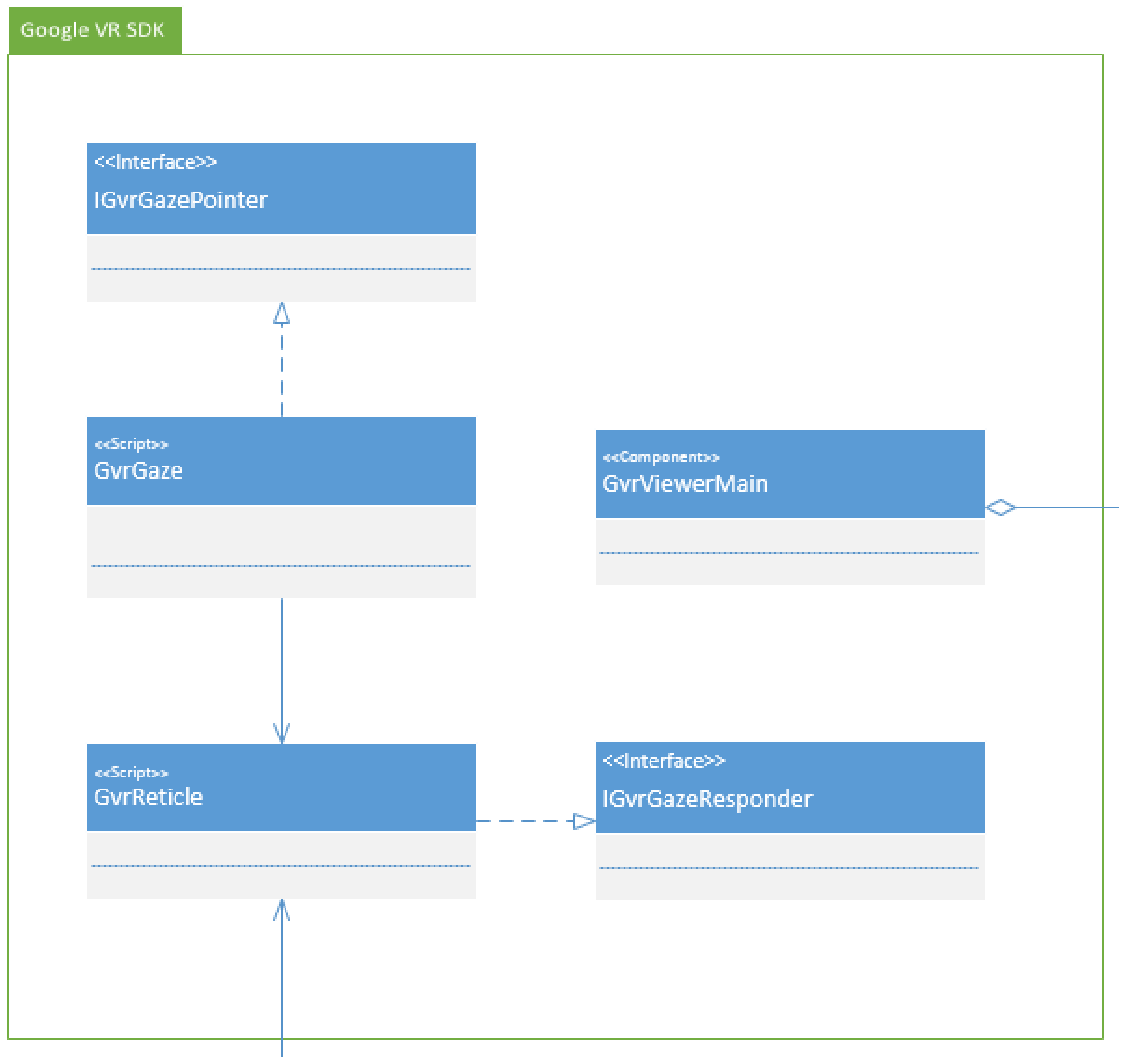

Figure 6.4 - Structural Diagram (Google VR SDK)

Figure 6.5 shows the MovableComponent package which consists of all components that will be moving in some way during the execution of the program. This package includes pedestrians (people, animals, etc.) and vehicles. The package also contains the scripts that these components will rely on to perform their movement and coordination.

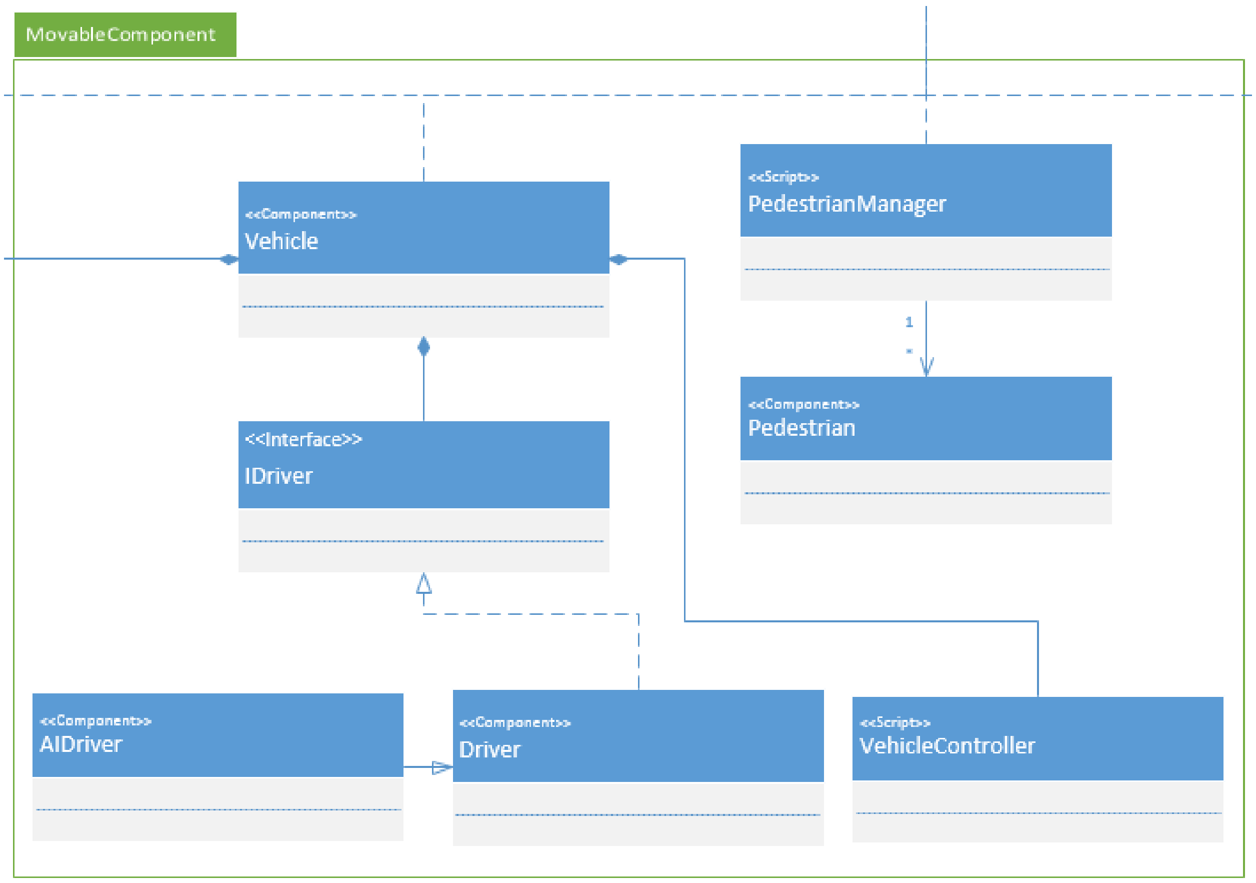

Figure 6.5 - Structural Diagram (MovableComponent)

Figure 6.6 shows the EnvironmentalObjects package which contains objects that are non-moving and exist in the environment such as plants, buildings, and roadways. The hierarchy below demonstrates how full environments will be made up of smaller components such as what was listed previously.

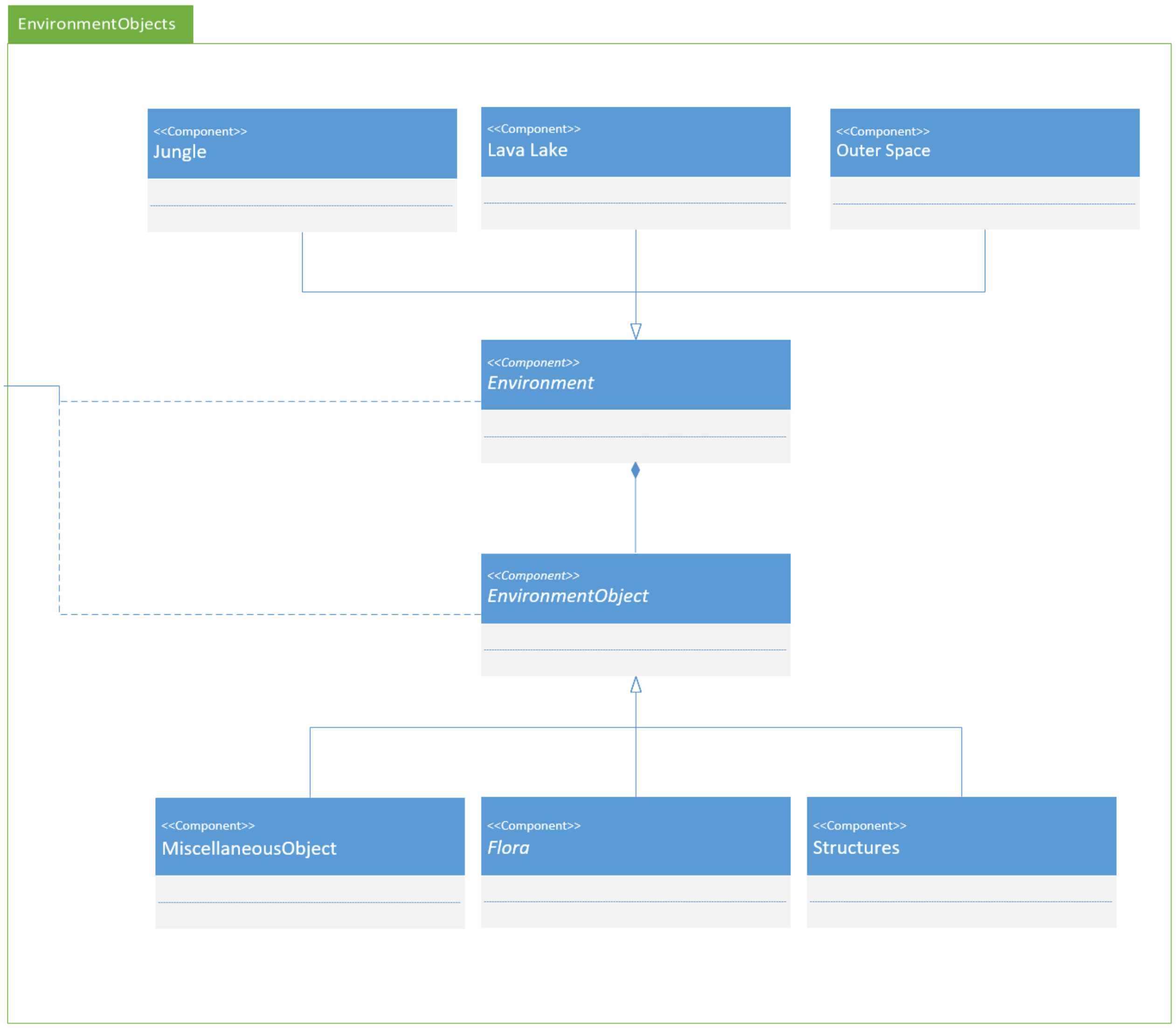

Figure 6.6 - Structural Diagram (EnvironmentObjects)

6.3 User Interface Design

Our user interface design is built around the technologies we are implementing. Virtual reality has a defined structure of displaying an image on two separate screens with logical angles that simulate what eyes see. With that, we are trying to create a very realistic depiction of riding in the car with a friend while the friend engages in dangerous activities. Google's SDK has provided many useful assets that have helped create the menu screens and input management to allow the player to control the experience. Figures 6.7 shows the user interface of the start menu. This is the first thing the user will see when starting the application. Figure 6.8 shows the view of the driver when the user is looking at him. This is a key view that the user will frequently look at. Figure 6.9 shows the tutorial level, which is designed to let the user get familiar with the goal and controlling the system. Figure 6.10 shows the first level of the experience, which is a jungle environment. Figure 6.11 displays the second level, a lava lake in a mountain. Figure 6.12 shows the final level, which is in outer space.

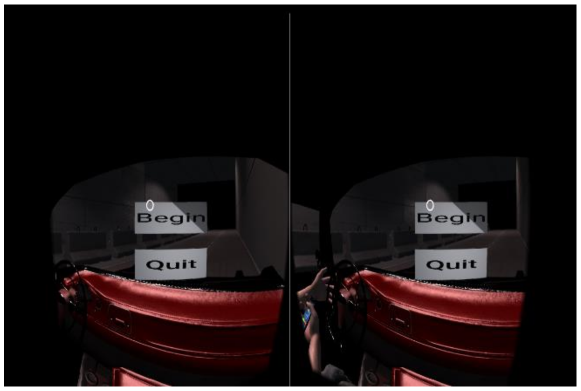

Figure 6.7 - Start Menu View

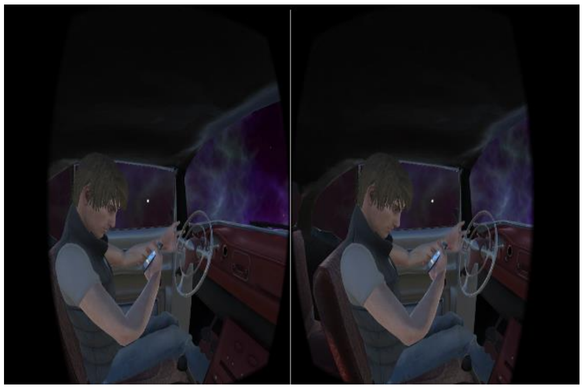

Figure 6.8 - Driver

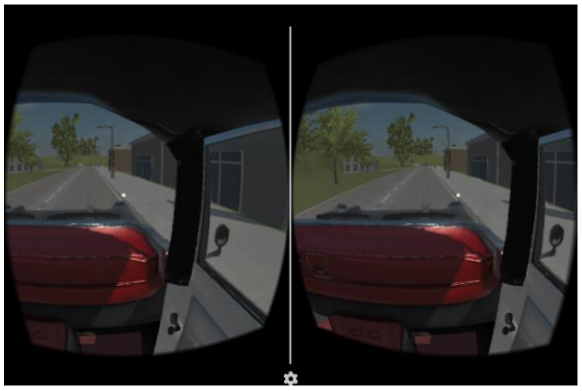

Figure 6.9 - Tutorial

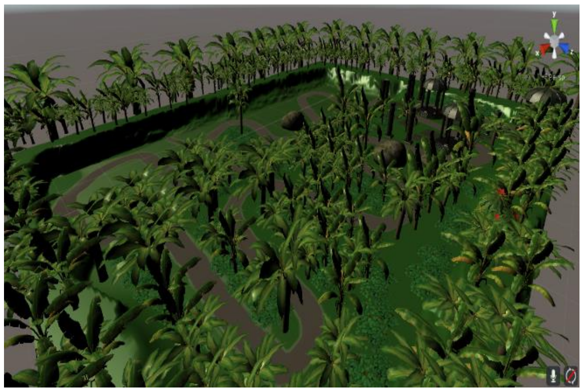

Figure 6.10 - Jungle

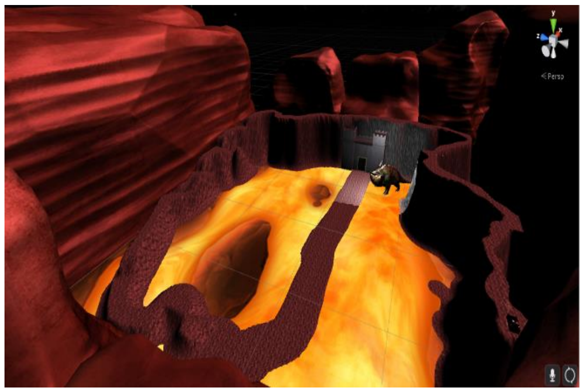
Figure 6.11 – Lava Lake

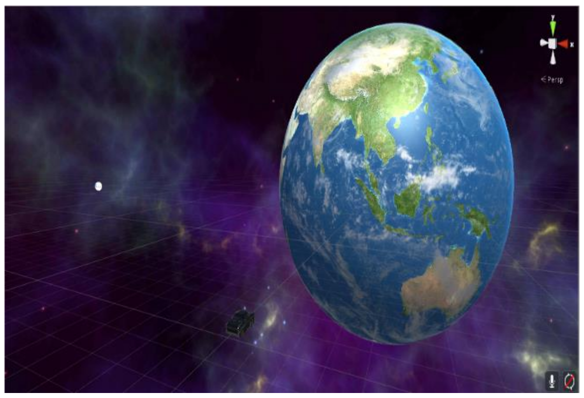
Figure 6.12 – Outer Space

6.4 Behavioral Design

In Figure 6.13, the behavior of the system is displayed. The activity diagram shows the flow of the experience and gives the steps required to succeed in the system, as well as the fail state requirements.

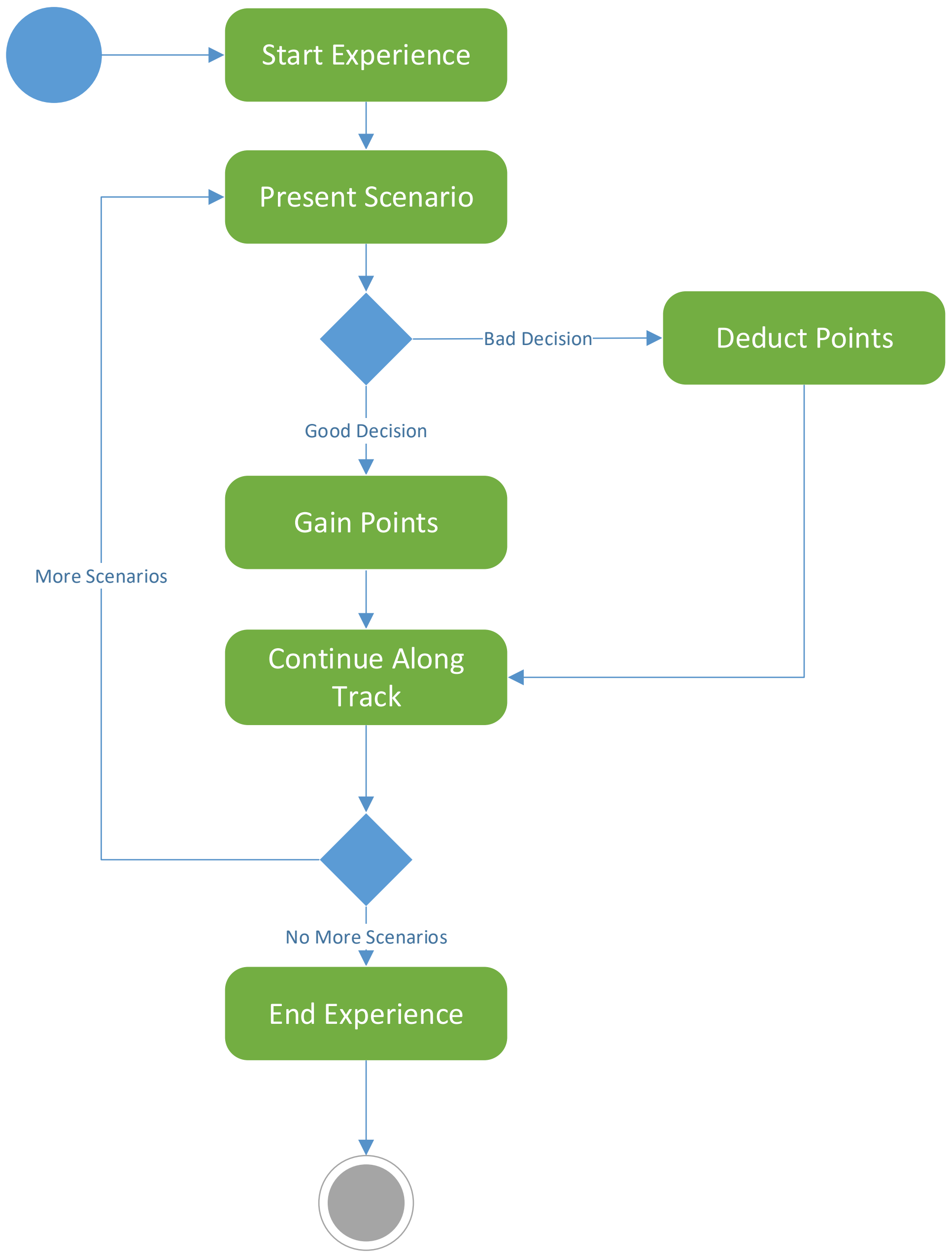

Figure 6.13 - Activity Diagram

6.5 Design Alternatives & Design Rationale

With our project, we are using Unity to create an experience that can run on mobile devices. Unity is designed with the component-based architecture in mind, and the way objects are implemented is based around that concept. Initially we looked into MVC, which is similar to our current design. However, each component in Unity essentially has its own model, view, and controller. The design would be complicated, and would not be as accurate as the component-based architecture. When designing environments, we started with a realist approach, having a highway, suburban, and city section. We were later asked by our business sponsors to take a less realistic approach. We have changed the environments into a jungle, a lava lake, and outer space. We still plan to feature our core systems in the game which includes the driver texting on his phone, interacting with the driver to solve scenarios, and creating a driving path that the user will be able to experience during each individual scene.

7. System Implementation

7.1 Programming Languages & Tools

We are implementing our project using Unity, which takes advantage of C# for creating scripts. Unity provides an IDE called MonoDevelop, however we are using Microsoft Visual Studio, which can be used instead of MonoDevelop.

7.2 Coding Conventions

We will adhere to the coding conventions designed around Unity development as well as Microsoft's C# conventions. We will also be following Unity best practice for component design, which will help improve maintainability and performance.

7.3 Code Version Control

As with all projects being worked on by multiple personnel, version control is very important for the efficiency of our workflow. We will be using a combination of Git and Unity SmartMerge for our version control which will handle branching and merge conflicts. We will be hosting our repository in GitHub.

7.4 Implementation Alternatives & Decision Rationale

One alternative development tool we could have used instead of Unity is Unreal Engine. Unreal is another game engine that is widely available and features mobile development and also has Google Cardboard SDK support. With Unreal we would also be developing using C++ instead of C#. Our team decided to use Unity over Unreal because we are all more familiar with C# and virtual reality development is more popular with Unity, so the documentation and resources available will be better defined. Erie Insurance has stated that they are aware of the terms of service with Unity and has agreed to allow us to proceed with Unity development.

7.5 Analysis of Key Algorithms

N/A

8. System Testing

8.1 Test Automation Framework

Our project is developed following the test-driven development methodology. In section 8, we will be covering the tests designed for our application as we continue developing it. In order for our system requirements to be verified, there will be tests created for each one to ensure correct implementation.

8.1.1 Steps for Installing Test Framework

Our tests are designed using Unity Test Tools, which is an asset that allows assertions on Unity objects and scripts to verify that everything is working correctly. In order to install the testing framework, all that needs to be done is download Unity Test Tools from the Unity Asset Store and add it to an existing project.

8.1.2 Steps for Running Test Cases

In order to run a test case, the test case must be opened in Unity Test Tools. From there, the tests can be run or modified to specified settings.

8.2 Test Case Design

8.2.1 Test Suites

Test suites are collection of test cases that test related functions of a program. In our project, we have separated tests into scenario interaction, environment interaction, and system performance.

Table 8.2.1 defines the tests belonging to scenario interactions. These will test all of the features of the application that require user input to continue.

Table 8.2.2 defines the tests belonging to environment interactions. These represent situations that involve objects interacting in the environment, outside of the user’s control.

Table 8.2.3 contains tests that test for the system performing at expected standards for VR.

8.2.2 Unit Test Cases

Unit test cases test all parts of an individual unit within a system or subsystem. Table 8.2.4 describes the test for resetting the camera to the center upon double clicks. Table 8.2.5 shows the test for allowing the user to change camera rotation in the application.

8.2.3 Integration Test Cases

System tests covers major system functionalities, and tests specific system requirements. Table 8.2.6 is the test for allowing the user to interact with the environment by clicking objects.

8.2.4 System Test Cases

Integration test cases test the connection between the units of a system or subsystem. Table 8.2.7 shows the test for scenario triggers in the environment. Table 8.2.8 shows the test for failing a scenario due to not preventing the driver from texting.

8.2.5 Acceptance Test Cases

These test cases reflect user requirements. The tests verify that specific requirements are working as planned for the user. They ensure that the system covers the most important requirements provided by the end user. Table 8.2.9 shows the test for the AI driver to be distracted during scenarios. Table 8.2.10 shows the test for the hardware requirement to run the application. Table 8.2.11 shows the test for the virtual reality representation of the application on the user's screen. Table 8.2.12 shows the test for the frame rate performance of the system, which provides a smooth virtual experience for the user.

8.3 Test Case Execution Report

The test case execution reports outline the steps taken to execute a given test case. They also provide the status of the test and any defects that will prevent the test from passing.

8.3.1 Unit Testing Report

Table 8.3.1 shows the execution steps for resetting the camera. The test is successful as we have implemented the feature.

Table 8.3.3 shows the execution steps for interacting with objects in the environment.

8.3.2 Integration Testing Report

Table 8.3.4 shows the execution steps for triggering a scenario.

8.3.3 System Testing Report

Table 8.3.2 shows the execution steps for moving the camera around in the environment.

Table 8.3.5 shows the execution steps for failing a scenario.

8.3.4 Acceptance Testing Report

Table 8.3.6 shows the execution steps for AI driver actions.

Table 8.3.7 shows the execution steps for running on mobile hardware.

Table 8.3.8 shows the execution steps for stereoscopic rendering.

Table 8.3.9 shows the execution steps for system frame rate.

9. Challenges & Open Issues

9.1 Challenges Faced in Requirements Engineering

We had trouble dealing with somewhat vague requirements provided by the industry sponsor, and were faced with the task of continuous meetings in order to get a clear understanding of the sponsor's needs in regard to the system.

9.2 Challenges Faced in System Development

Our first issue we faced was configuring version control to work with our system. Git alone does not work for Unity projects, and scenes are stored in binary files, so if a scene was worked on concurrently, it would not be able to merge. The documentation was confusing, and we failed to set it up properly a few times. We also had trouble with incompatible versions between Unity and the Google VR SDK. The SDK we originally had was out of date. Additionally, another challenge we faced was getting the driver into the car and being able to make him move in a realistic manner.

Another issue we faced was a changing direction of the project. When demoing progress to our business sponsors, they liked what we had. However, they wanted us to take a less realistic approach to everything and gamify the system more. Once we discussed our plan and modified the requirements everyone seemed positive about the new approach.

9.3 Open Issues & Ideas for Solutions

N/A

10. System Manuals

10.1 Instructions for System Development

In order to develop the application, the environment must be set up. After the required steps are completed, the project must be opened in Unity. From there, any part of the system can modified.

10.1.1 How to Set Up Development Environment

In order to develop the application, the developer must have Unity 5.4.1f1 installed as well as Git in order to pull from the repository. Once pulled, opening the project in Unity will allow for additional development.

10.1.2 Notes on System Further Extensions

10.2 Instructions for System Deployment

Steps to build and export to Android:

1. Select File > Build Settings
2. Select the platform as Android, then switch platform
3. Select Player Settings, in the resolution and presentation tab, select landscape left and use the 32-bit display buffer
4. Select Other Settings, change minimum API level to be Android 4.4 KitKat (API level 19)
5. Select Build to create APK

10.2.1 Platform Requirements

In order to build and deploy the application, Unity is required. Along with that, the Android SDK and Java SDK must be installed as well.

10.2.2 System Installation

To install on Android, the APK must be downloaded. After downloading, it can be installed and then started.

10.3 Instructions for System End Users

N/A

11. Conclusion

11.1 Achievement

11.2 Lessons Learned

11.3 Acknowledgment

12. References

[1] MSDN, C# Programmer's Reference, Accessed on 10/21/2016
https://msdn.microsoft.com/en-us/library/618ayhy6(v=vs.71).aspx

[2] Unity, Unity Scripting Reference, Accessed on 10/21/2016
https://docs.unity3d.com/ScriptReference/

[3] Unity, Unity Manual, Accessed on 10/21/2016
https://docs.unity3d.com/Manual/index.html

[4] Unity, Unity Test Tools, Accessed on 10/21/2016
https://unity3d.com/learn/tutorials/topics/production/unity-test-tools

[5] Unity, Unity Community, Accessed on 10/21/2016
https://forum.unity3d.com/

[6] Google, Google VR SDK for Unity, Accessed on 10/21/2016
https://developers.google.com/vr/unity/