INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA (ISEL)

DEPARTAMENTO DE ENGENHARIA ELETRÓNICA E DE TELECOMUNICAÇÕES E COMPUTADORES (DEETC)

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA

UNIDADE CURRICULAR DE PROJETO

VR Cop - Approaching a Vehicle

Pablo Hernández (47518)

Tiago Correia (46314)

Orientador(es)

Professor Hélder Bastos

Professor Diogo Lopes

Liaison Officer Jérémy Kespite

julho, 2023

Resumo

Este documento contém a informação pertencente ao projeto final de curso e Unidade Curricular de Projeto, lecionada no curso de Engenharia Informática e Multimédia no Instituto Superior de Engenharia de Lisboa.

A proposta para a realização deste projeto foi disponibilizada pelos Professores Hélder Bastos e Diogo Lopes, em colaboração com o Liaison Officer dos Innovation Labs da Europol, Jérémy Kespite.

O trabalho proposto consiste na investigação e implementação de um simulador em Realidade Virtual que pretende treinar novos agentes da lei sobre como se aproximarem de um veículo. O projeto tem como objetivo principal a elaboração de uma *proof of concept* de um sistema virtual moderno que visa otimizar o processo de aprendizagem de agentes da lei e, simultaneamente, requer menos recursos para a sua preparação.

Esta proposta foi escolhida como projeto final de curso principalmente porque o tema de Realidade Virtual chamou a atenção dos alunos. O seu interesse por *game development* e o seu conhecimento do *software* Unity fizeram com que este projeto final fosse perfeito para os alunos.

A contribuição mais relevante que foi feita na elaboração deste projeto foi a arquitetura do simulador. Este esqueleto é a base de um produto cujo objetivo é a escalabilidade para criar um eventual produto completo que poderia ser verdadeiramente usado para a aprendizagem, não só de agentes da lei, mas também todo tipo de profissões como, por exemplo, bombeiros, cirurgiões, dentistas, enfermeiros, etc.

Em conclusão, é um projeto no qual os alunos estiveram entusiasmados para participar desde o início, e que acharam interessante, divertido e relevante.

Abstract

This document contains information related to the final project of the course and also the Project Curricular Unit, taught in the Computer Engineering and Multimedia course at the Instituto Superior de Engenharia de Lisboa.

The proposal for this project was provided by Professors Hélder Bastos and Diogo Lopes, in collaboration with the Innovation Liaison Officer of the Innovation Labs at Europol, Jérémy Kespite.

The proposed work consists of the research and implementation of a Virtual Reality simulator aimed at training new law enforcement agents on how to approach a vehicle. The main objective of the project is to develop a proof of concept of a modern virtual system that optimizes the learning process for law enforcement agents while requiring fewer resources for its preparation.

The most relevant contribution made in the development of this project was the architecture of the simulator. This framework serves as the basis for a product that aims for scalability to create a potential complete product that could truly be used for learning, not only by law enforcement agents but also by all kinds of professions, such as firefighters, surgeons, dentists, nurses, etc.

In conclusion, this is a project the students have been excited to be a part of from the beginning, and found interesting, fun, and relevant.

Acknowledgments

We would like to thank our advisors, Hélder Bastos and Diogo Lopes, for the opportunity to participate in this project. We also extend our gratitude to our reviewer, Professor Rui Jesus, and Professor Paulo Trigo from the Project Curricular Unit, as well as Jérémy Kespite for following our progress throughout the semester and providing crucial help, opinions, and advice to make our project more realistic and aligned with our objectives.

We want to express our appreciation to all the students, professors and colleagues who tested our project in the laboratory and provided feedback to help us improve our work.

We'd like to thank our families for supporting us during our studies.

We'd like to leave a special thanks to the editorial assistant, Nádia Silva, that took her time to review our report and criticize every lack of cohesion and punctuation.

Lastly, we are grateful for the opportunity to be part of an "international" project through the collaboration of Jérémy Kespite.

Monkey see, . . .

. . . monkey do.

Table of Contents

List of Tabels

List of Figures

Acronyms

VR: Virtual Reality is a simulated experience that uses screens near the eyes (inside VR goggles) and wireless controllers to give the user an immersive sensation of a virtual world.

NPC: Non-playable characters (NPCs) are all the characters (human, in the context of the project) that cannot be directly controlled by the user. When referring to an NPC in this project, it specifically refers to the car driver.

AI: Being an important element in the project, the Artificial Intelligence can be abbreviated as AI. Artificial Intelligence refers to the development of computer systems capable of performing tasks that would typically require human intelligence, such as learning, problem-solving, and decision-making. It encompasses various subfields, including machine learning, natural language processing, and computer vision.

AR: Augmented Reality is the technology that integrates a virtual image or interface into the user's view of the real world. It provides real-time visualization and interaction with the computer-generated images, to create a seamless experience with instant feedback to the user.

XR: Extended Reality (XR) is an umbrella term that encompasses virtual reality, augmented reality and mixed reality, combining digital and physical environments to create immersive experiences. It extends human perception and interaction by blending the real and virtual worlds. In this project, the students refer to XR toolkit, which is used for the integration of the virtual reality headset.

NLP: Natural Language Processing (NLP) is a subfield of Artificial Intelligence that focuses on the interaction between computers and human language. It involves the development of algorithms and techniques to enable computers to understand, interpret, and generate human language in a way that is meaningful and useful. NLP plays a key role in various applications, such as language translation, sentiment analysis, chatbots, and voice assistants. In the context of this project, the students use NLP through Wit.ai to create a voice command system in order to provide an intuitive, dynamic and reactive way to interact with the environment, to ensure a certain degree of realism.

SUS: System Usability Scale is a widely used questionnaire designed to measure the usability of a system, such as a software application or website. It provides a standardized way of evaluating users' subjective perceptions of the usability of a system. The SUS consists of ten statements related to usability, and participants rate their level of agreement with each statement on a 5-point Likert scale ranging from "Strongly Disagree" to "Strongly Agree."

Chapter 1

Introduction

In an era marked by rapid technological advancements, it is important that law enforcement agencies adapt and innovate to stay ahead of the curve. As augmented and virtual reality continue to develop and become more accessible, one could ask oneself what new uses this technology can accomplish. Police checkpoints play a critical role in maintaining public safety and upholding the law. Whether they are utilized for traffic control, counter-terrorism efforts or crime prevention, effective training of police officers in checkpoint operations is of utmost importance. It would be a mistake not to take advantage of these innovations to aid training new police officers to situations where anything can happen. This project comes as a solution to this problem and hopes to be the groundwork for this innovation.

1.1 Motivation

This proposal was chosen as the final project mainly because the topic of Virtual Reality caught the students' attention and, combined with their interest in game development and their knowledge of Unity software, it constituted a perfect final project for them.

1.2 Objectives

This project seeks to study the capabilities of VR technology in training law enforcement agents. To do so the students will develop a virtual reality simulation to train new and experienced law enforcement agents on how to approach a vehicle in a checkpoint operation.

Chapter 2

Related Work

VRCop does not exist in its entirety anywhere, but the ideas and fundamentals are already out there. Having said that, there are already several systems that are similar enough to this one that are worth mentioning. The following 3 systems are either similar in nature, objectives, technologies or techniques used.

2.1 SimX: Virtual Reality Medical Simulation

SimX: Virtual Reality Medical Simulation is a simulator developed by SimX with the goal of training nurses for a variety of different scenarios. It is the closest system one can find to this project's root idea. It is a virtual reality simulation where one or more users can interact with the simulation and with each other with the goal of saving the life of a person in the simulation. Like VRCop, this system's mission is to train and to better prepare nurses for all kind of scenarios in their line of duty.

Figure 2.1: SimX logo

2.2 Police Simulator: Patrol Officers

Figure 2.2: Police Simulator Patrol Officers Game Cover

Police Simulator: Patrol Officers is a video game developed by Aesir Interactive and published by Astragon Entertainment in which the player lives the day-to-day of a police officer. The game includes a variety of police procedures but what is most important to mention is their dynamic transit system with accidents, road blockages and checkpoint operations. Much like VRCop, this system has a variety of checkpoint operation scenarios and it could arguably be educational but not officially.

2.3 Phasmophobia

Figure 2.3: Phasmophobia Game Cover

Phasmophobia is a cooperative horror game developed and published by Kinectic Games in which the players have to explore a haunted location, discover what kind of ghost is haunting it and escape with their lives. Although an horror game is far from this project's goals and theme, Phasmophobia is similar to VRCop in its game interaction. Like VRCop, Phasmophobia can be played using a VR set, has similar controls, has an utility belt to store items and, most important, has voice recognition interactions in the game.

2.4 Overview

In the following table, it is clear what similarities were sought in the previous mentioned systems:

Applications	Police Sim	VR Supp.	Socket Int.	Voice Recog.	Educational
SimX: VR Medical Simulation	No	Yes	Yes	No	Yes
Police Simulator: Patrol Officers	Yes	No	No	No	Debatable
Phasmophobia	No	Yes	Yes	Yes	No

Table 2.1: Related Work Similarities

The main aspects that were taken into account when making the table above were the following:

- Is the app a Police Simulator?
- Does the app support VR?
- If it does support VR, does it use VR socket interactions*?
- Does the system use voice recognition technology?
- Lastly, can the app be used for educational purposes?

*VR socket interaction: It is a system that allows the user to store in-game objects in slots (sockets) which can be visible or invisible, attached to the user's virtual body or stationary in the world.

Chapter 3

Support Technologies

This project is reliant on a number of different technologies, packages, libraries, and models that must be understood and accounted for in order to understand VRCop's functionality and development. That being said, in this chapter, the technologies will be explained in the context of VRCop.

3.1 Unity

Firstly, the simulation was developed using Unity [A] (version 2021.3.21f), a popular cross-platform game Engine developed by Unity Technologies. The Unity Editor was utilized to design and develop the environment where the simulation takes place, the user interface and every asset and component configuration in the scene. When it came to choosing the Engine, it could either be Unity or Unreal Engine (two of the best free game Engines). Due to Unity's much larger documentation, support community, and the students' and advisor's (Diogo Lopes) experience with Unity, Unreal Engine was discarded.

3.2 XR Interaction Toolkit

In order to develop a VR experience there are certain requirements when it comes to the interaction of the user with the world, how they move, how they look around, how they reach for different objects and interact with them, etc.. To be able to use a virtual reality set in a Unity simulation, one can use the Unity XR Interaction Toolkit [A] . According to the Unity Manual, the XR Interaction Toolkit package is described as follows: "The XR Interaction Toolkit package is a high-level, component-based, interaction system for creating VR and AR experiences. It provides a framework that makes 3D and UI interactions available from Unity input events. The core of this system is a set of base Interactor and Interactable components, and an Interaction Manager that ties these two types of components together. It also contains components that you can use for locomotion and

drawing visuals" (Unity Technologies, 2022, para. 1). Using the XR Interaction Toolkit, this project can support the following interaction tasks:

- Cross-platform XR controller input: Meta Quest (Oculus), OpenXR, Windows Mixed Reality, and other platforms.
- Basic object hover, select and grab functionality.
- Visual feedback, such as tinting and line rendering, used to indicate possible and active interactions.
- Basic canvas UI interaction using XR controllers.

3.3 Wit.ai

One of the more important aspects the students took into consideration while developing this project was the usability of the user and how they would interact with the Driver NPC in the simulation. That being said, it was decided that every interaction the user would have with NPCs would be through gestures and voice recognition instead of using buttons and interfaces. To achieve that goal, there needed to be a way of implementing voice recognition on an Unity simulation. Wit.ai [A] is a NLP tool developed by Wit.ai (owned by Meta) to enable developers to turn Utterances (input voice clips) into Intents (commands or functions that are called when an utterance is recognized). With this package, VRCop's simulation expects a pre-selected list of voice commands that can be used as inputs and improve the flow of use in this project.

3.4 Audacity

In order to improve the user's experience and make the simulation more immersive, the driver NPC replies to most of the user's voice inputs. The replies were recorded and edited in the software by the students. Audacity [A] is a free and open-source digital audio editor and recording application software that was used to modify the pitch and the cadence of the recorded audio clips in order to replicate the feeling of a stranger.

Chapter 4

Proposed Model

In this section, the proposed model for accomplishing the objectives of this project will be presented. Firstly the Requirements at [4.1] will be presented, then the General Description at [4.2] followed by the Specific Description at [4.3]. After that comes the Fundamentals at [4.4] and the Concept at [4.5].

4.1 Requirements

The requirement analysis is conducted during the early evaluation phase of the project. In this phase, it is necessary to define the system requirements and organize them into use cases.

4.2 General Description

As stated in the introduction, this project seeks to study the capabilities of VR technology in training law enforcement agents. To do so the students will develop a virtual reality simulation to train new and experienced law enforcement agents on how to approach a vehicle in a checkpoint operation. The students were given freedom to research and utilize the technologies available to them personally or provided by ISEL in the form of a laboratory with an HTC Vive VR set and a VR-ready desktop computer. The students were also given the choice of their preferred engine, and decided Unity was more adequate since they already had experience working with the software.

4.2.1 Summary of Objectives

This project aims to create a product that will reduce the time and expenses needed to set up a scenario in order to practice. A VR simulation that can be executed fast and intuitively can prove to be the next step in modernizing professional training. Secondly, the students found it relevant to implement software that optimizes the analysis and feedback

that one can give after practicing a law enforcement procedure. The ability to record and replay a scenario that was just carried out increases the efficiency and effectiveness of the training.

4.2.2 Target Audience

The main target audience for this project would be law enforcement agents, whether it be agents in training or experienced ones. The students seek to create the base of what could one day become a product for any professional who wishes to undergo training or practice, so long as the software eventually supports it (currently only law enforcement).

4.3 Specific Description

For a more detailed description of the project's main structure and functionalities, a series of tables have been created in order to illustrate the system's functions, attributes and use cases.

describe the specific actions a software should be able to perform. They are defined to meet the requirements of the project. System functions are useful because they outline the capabilities of the system.

System attributes, also known as non-functional requirements or quality attributes, are the characteristics that describe the overall behavior and qualities of a system. Unlike system functions, which focus on what the system does, system attributes focus on how well it performs certain functions. Some common examples of system attributes are Ease of Use, Platforms, Feedback, etc.

Use cases refer to a description or representation of a specific interaction or scenario between users (actors) and a system. Use cases help understand how users will interact with the system and what functionality it should provide to meet their needs. They are important in the analysis and design phases of the project, as they provide a different and more user-centered to ensure the system meets the needs and expectations of its users.

4.3.1 System Functions

The System Functions are gathered in a table that lists all of the main features of the implemented system, generally but not exclusively following a rule where the Function completes the following sentence: ”The system does [Function]” in order to normalize the information in the table.

The table contains three columns titled ”Requirement”, ”Function” and ”Category”, respectively. The first column assigns a decimal number starting with 1 or 2, depending on whether the function refers to the actual carrying out of the simulation (trainee) or to the

Requirement	Function	Category
R1.1	Display virtual environment	Evident
R1.2	Allow user to move	Evident
R1.3	Allow user to interact with NPCs	Evident
R1.4	Allow user to interact with objects	Evident
R1.5	Conclude simulation	Evident
R1.6	Save simulation	Invisible
R1.7	Voice recognition	Optional
R1.8	Multiplayer	Optional
R2.1	Display simulation	Evident
R2.1	Change simulation options	Evident
R2.3	Save simulation presets	Invisible

Table 4.1: System Functions table

learning / preparation aspect of the project (evaluator). The second column is the name of the function. The third and last column categorizes the functions into ”evident” (visible and mandatory), ”optional” (treated as possible details to add in a later stage) and ”invisible” (feature that isn’t represented visually to the user).

4.3.2 System Attributes

Attribute	Detail / Restriction	Category (pref. / mand.)
Ease of use	D - intuitive, straightforward	Text
Platforms	D - Unity	Text
Interface metaphor	D - user of VR Headset and controllers	Text
Feedback	D - human feedback	Text
Retail cost	R - [400, 1000](€) per kit R - [600, 1000](€) per desktop	Text

Table 4.2: System Attributes table

The System Attributes are organized in a similar fashion, a table that describes some general aspects to take into account in terms of how the functionalities are presented to the user. This table does not exclusively list required attributes, but also preferable ones. It is divided into three columns labeled ”Attribute”, ”Detail / Restriction” and ”Category (preferable / mandatory)”. The first column lists the aforementioned criteria, but the most relevant in the context of this project. The second column states an initial - ”D” or ”R” depending on whether the subject in question is a chosen detail(D) or a limiting restriction(R) of the system and also specifies the detail or restriction in question. The last column divides the attributes into preferable and mandatory, depending on the degree of obligation it has.

4.3.3 Use Cases

Name:	Prepare a scenario
Summary:	The supervisor puts on the VR Headset and navigates the menu system to select the desired options for the scenario that the trainee will participate in.
References:	R2.2, R2.3
Name:	Play a scenario
Summary:	The trainee puts on the VR Headset and presses the Start button in the menu system. The simulation starts, the user exits the vehicle and carries out the necessary actions to complete the scenario. The user interacts with the vehicle passengers and with relevant items using the VR controllers.
References:	R1.1, R1.2, R13, R1.4, R1.5

Table 4.3: Use Cases table

The tables use to present the use cases have three rows labeled ”Name”, ”Summary” and ”References”. The references column points to a system function that is related to the use case by mentioning its number.

4.3.4 Actors

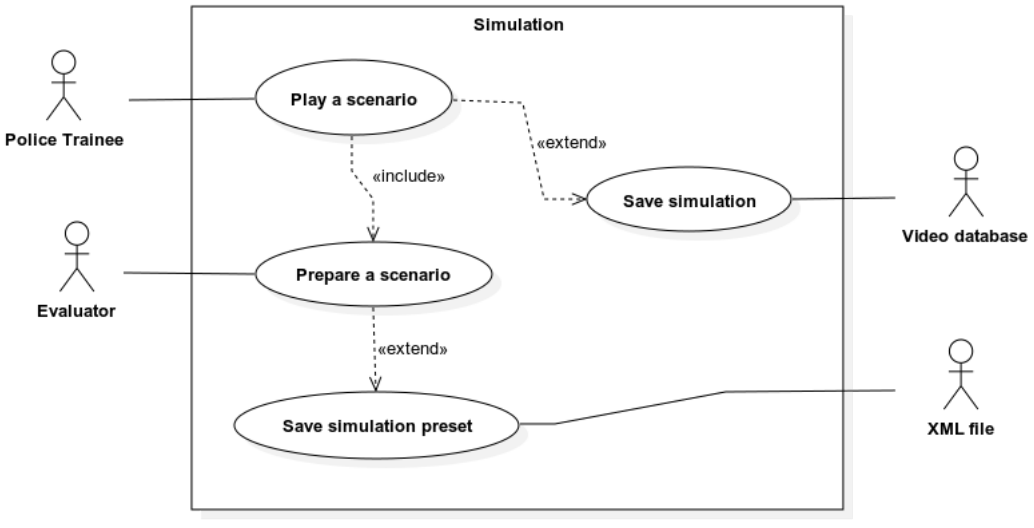

Figure 4.1: ”Use Case Diagram”

The diagram portrays the dependency between the actors and use cases. Playing a scenario depends on the preparation of said scenario by the Evaluator, which in turn can be saved into a preset and stored in an XML file. The simulation can also be saved and later reviewed, by saving it in a video file repository.

4.3.5 Detailed Use Cases

Name:	Simulation scenario 1
Summary:	The user starts approaching the vehicle. When the user gets to the driver's window they ask the driver to turn off the engine by pressing the A button and uttering the command ”turn off the engine please”. The driver complies. The user presses A again and asks for the driver's license and vehicle registration by saying ”license and registration please”. The driver extends their arm through the window with the documents. The user grabs the documents and puts them in their waist inventory slot. The user walks back to the vehicle and grabs the interactable UI element near the driver seat of the police vehicle to end the simulation. A menu appears with the options to Replay Simulation, Change Simulation Options and Exit Simulation. A text area displaying ”Simulation Saved” appears.
Name:	Simulation scenario 2
Summary:	The user starts approaching the vehicle. The driver gets out of the vehicle and starts talking, asking what's going on. the user tells the driver to get back inside the vehicle with the command ”please enter the vehicle”. The driver does not comply and starts running away. The user aims their weapon in the general direction of the driver and tells the driver to stop with the command ”stop!”. The driver stops and is told to get on their knees with the command ”on your knees!”. The simulation ends.

Table 4.4: Detailed Use Cases table

The second part of the use cases consists of two more tables describing ”Simulation Scenario 1” and ”Simulation Scenario 2”. These tables do not have a References row as they are examples of an already presented table, labeled ”Play a scenario” and thus have the same references.

In the first Simulation Scenario, the user is portrayed carrying out a simple operation with an expected outcome for a complying civilian.

In the second Simulation Scenario, the civilian is not as compliant and will act unpredictably, requiring decision-making from the user.

4.4 Fundamentals

While studying and developing this project, some fundamental concepts made themselves clear and its understanding will aid in understanding this project. The fundamental con-

cepts identified were: Law enforcement procedures in a checkpoint operation at [4.4.1], State machine at [4.5.3] and AI voice recognition at [4.4.3].

4.4.1 Law enforcement procedures in a checkpoint operation

Checkpoints operations are operations where traffic enforcement checks vehicular/pedestrian traffic in order to enforce, control and measure circulation, laws, orders, and regulations. When conducting a checkpoint operation, law enforcement agents must approach the vehicle slowly, check for any danger or illegalities, stand at either side of the car's windows and ask to see the driver's documents. After analysing the situation, if everything checks out, send the driver off.

Since anything can happen, the course of the operation depends on what sort of law enforcement agents are behind it. That being said, agents must react to any anomaly and decide on the spot their course of action.

4.4.2 State Machine

A State Machine or Finite State Machine is a finite behaviour model that consists of a finite number of States, Events and Actions.

Based on the current State and an input Event (or a combination of an Event and one or more variables), the machine perform transitions and outputs Actions to be executed. Each State can have one or more Entry Actions (Actions to be executed when entering the State), one or more Do Actions (Actions to be executed when staying in the State), and one or more Exit Actions (Actions to be executed when exiting the State).

4.4.3 AI voice recognition

AI voice recognition refers to the technology that enables artificial intelligent machines to understand and interpret human speech. It involves converting spoken language into written text or commands, allowing seamless interaction between humans and machines. AI voice recognition systems analyze acoustic patterns, linguistic context and semantic meaning to accurately transcribe and comprehend spoken words.

4.5 Concept

This section will explain the analysis conducted throughout the development of this project. Firstly a Real Life Experience will be mentioned at [4.5.1]. Then an Activity Diagram section at [4.5.2]. After that a State Machine related section at [4.5.3], a System Package Architecture section at [4.5.4] and a Sequence Diagram section at [4.5.5].

4.5.1 Real Life Experiences

Before development started and in order to better understand the context and functionality of this projects, several real life simulations were conducted at a parking lot near ISEL(`https://goo.gl/maps/ypENrkVd3BbB5otS8`).

The students of this project met with the advisors Hélder Bastos and Diogo Lopes in person to carry out an experiment in said parking lot. The activity consisted of simulating a police checkpoint operation, which started with two cars in line, separated by about 6 meters. The front car had two civilians and the car behind it, which was the police car, had two agents who, in this particular experiment, were the students.

After organizing the initial scene, the students were given some brief instructions on how to conduct a checkpoint operation: how to approach the car and ask for documents, what to do if a person exits the car or does something unexpected, etc.

The first iteration didn't have any significant incidents outside the norm, but in subsequent attempts, different spontaneous actions and reactions were introduced, and the student-agents had to try to resolve the situation in the best possible way.

Examples of spontaneous situations that occurred were:

- The driver got out of the car, concerned, and asked what was going on.
- A passenger in the vehicle handed something to the other before the agents approached.
- The passenger got out of the vehicle and started running away from the agents.
- The driver of the vehicle got out of the car and started approaching the agent.

The objective of this activity was to give the students a first-hand experience of how a checkpoint operation unfolds, including some of the many situations that can occur in the moment and how quickly they happen. This type of practical activity was aimed to help the students develop a mental sketch of the problem that would be later implemented in Unity.

4.5.2 Activity Diagram

Thanks to real life experiments, the students were able to start making decisions and draw diagrams regarding the behaviour of the Driver NPC that were refined with the help of the External Advisor Jérémy Kespite.

After deciding on the system attributes, functions and use cases, a simple Activity Diagram was drawn with the goal of better understanding what was intended to develop and what was work to be developed in future iterations. The diagram in question can be found in appendix at [A.2].

It was decided that what would be made in the first version was only the best case scenario. The user would approach the car, ask for the driver's documents, evaluate the documents, return the documents and tell the driver to drive away.

In the second version, there wouldn't be any violence or arrests but the driver could resist to oblige a command, exit the vehicle and either approach the user, walk away or run away.

In the third and more complete version, the driver would be able to attack or threaten to attack the user after exiting the car. The user could then point the gun at the driver and even arrest them.

4.5.3 State Machine

After deciding on what interactions were being implemented, it was decided that every single NPC behaviour would be decided and controlled by a State Machine.

A State Chart Diagram was drawn to better understand what needed to be developed regarding the ”brain” of the Driver NPC. Due to time constraints, in this simulation there can be no arrests and no violence by the Driver NPC. The State Chart Diagram in appendix at [A.1], represents the State Machine to be developed without violence or arrests.

The State Chart diagram represents the states of the Driver NPC, the actions that it would act on each state, the events that trigger each transition of state and the random values required.

4.5.4 System Package Architecture

After knowing what needed to be developed regarding the State Machine, there were other important components to plan out:

- A class State Machine that would save the current State of a NPC and evolve when it receives an Event.
- An abstract class State from which every unique State would extend and be built upon.
- An enum for every Event and Action and a Class to store both types of enums.
- A class called Event System that could send Events to any NPC with a State Machine and a class called Event Listener that would ”listen” to what's happening in the simulation.
- A class to store every constant used in the simulation and, more importantly, the State Machine thresholds for simulated randomness.

- A class for the NPC with every method required to execute the actions it receives from its State Machine.

A package structure was drawn to help implement the code and can be found in appendix at [A.3].

The class SimulationEvent would have an object SimulationEventType to store its type of event. The class Action would have an object ActionType to store its type of action. The class EventListener would have access to an instance of EventSystem to be able to send SimulationEvents that it listens to. The class EventSystem would have access to an instance of DriverNPC to send SimulationEvents it receives. The class DriverNPC would have an object StateMachine and would have access to an instance of EventSystem to send events to itself's StateMachine. The class StateMachine would have an object State to hold its currentState and it would update it when evolving the state machine. Every unique state would extend the abstract class State. The visibility of all classes is public.

4.5.5 Sequence Diagrams

After designing the package architecture, 2 sequence diagrams were drawn to better understand the inner works of this system and to facilitate its implementation in the future.

The first diagram represents the beginning of the simulation and how the Unity Engine controls its MonoBehaviour* classes. It can be found in appendix at [A.4].

As was shown in the first diagram, the State machine is created by the DriverNPC in its *Start()* method. Afterwards the class DriverNPC calls the method *Innit()*. The method *Innit()* "decides" what the first State will be according to the State Chart Diagram shown earlier on (link para o start chart diagram) and returns any Entry Actions of that State in the form of a List of Actions. DriverNPC adds what the method *Innit()* returns to its queue of Actions through the method *AddActionsToQueue(actions)*. When the Unity Engine calls the method *Update()*, if there are actions in the queue, DriverNPC executes said actions one by one, removing them from the queue, through the method *Act(action)*.

*By inheriting from MonoBehaviour, any class gains access to several important Unity callbacks and functions, such as *Start()* and *Update()*. *Start()* is called before the first frame of every execution and *Update()* is called once per frame.

The second diagram represents how the system reacts from Events triggered by the user, starting from the Event Listener and can be found in appendix at [A.5].

As was shown in the second diagram, when the user calls any method of the EventListener, the EventListener calls the method *SendEvent(ev)* of the EventSystem, that in turn calls the method *ReceiveEvent(ev)* of DriverNPC. DriverNPC then evolves its StateMachine by calling the method *Evolve(ev)* of StateMachine. StateMachine calls the method

Evolve(ev) of its currentState and if there is a transition to be made and its nextState is not null, StateMachine compiles the exit actions of its currentState and the entry actions of its nextState using the methods *GetExitActions()* and *GetEntryActions()* on the respective States and returns them using a List of Actions. Afterwards DriverNPC adds the actions returned by its StateMachine to its queue of Actions through the method *AddActionsTo-Queue(actions)*.

Chapter 5

Implementation

In this section, the implementation of the previous proposed model will be presented. The Simulation Environment at [5.1], the Driver NPC at [5.2], the XR Interaction Toolkit implementation at [5.3], the Wit.ai Implementation at [5.4], the Events and Event System at [5.5], the States and State Machine at [5.6] and the actions at [5.7.

5.1 Simulation Environment

The first section to be developed was the simulation environment. As this simulation is a prototype, the main focus when designing the environment was its functionality. That being said, a Unity Terrain was used as the environment's foundation, with a plane representing a stretch of a low traffic road in the middle of the country. Two cars were placed in the simulation to represent the police car and the car of whomever the police is investigating/intervening. Furthermore, some trees, grass and houses in the distance were added to aid in immersion.

Figure 5.1: VRCop Environment

5.2 Driver NPC

The Driver NPC consists of a 3D model downloaded from `https://www.mixamo.com/#/` which possesses a variety of components in Unity in order to make it behave like a civilian. The components are the following:

- Audio Source: This component can receive an Audio Clip and play it in the simulation. In this case, the Audio Clips are clips of the civilian's responses to the user's voice commands.

- Animator: This is the component that manages how the NPC moves when performing an action. The animator possesses an Animator Controller which visually represents the animations the NPC can perform and the transitions between them.

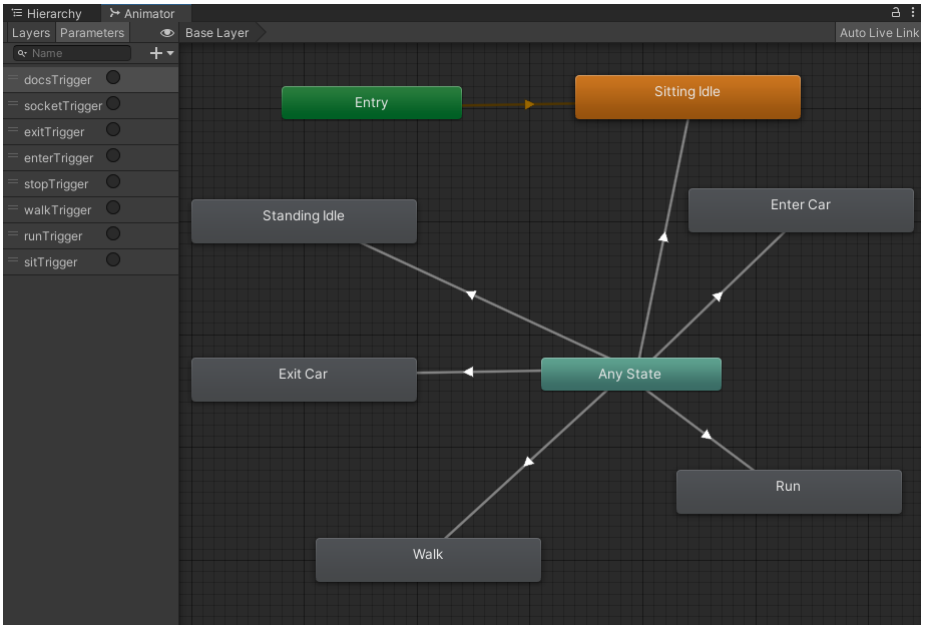

Figure 5.2: Base Layer do animator controller do NPC

Firstly, in the base layer of the controller, there is a starting state labeled ”Sitting Idle” which places the NPC in a sitting position to properly portray it sitting down inside its vehicle. The ”Any State” state, as the name suggests, declares that any animations to which this state has transitions from, can be activated at any time when their respective triggers are toggled. In this case, the NPC can enter and exit its car, idly stand, walk and run.

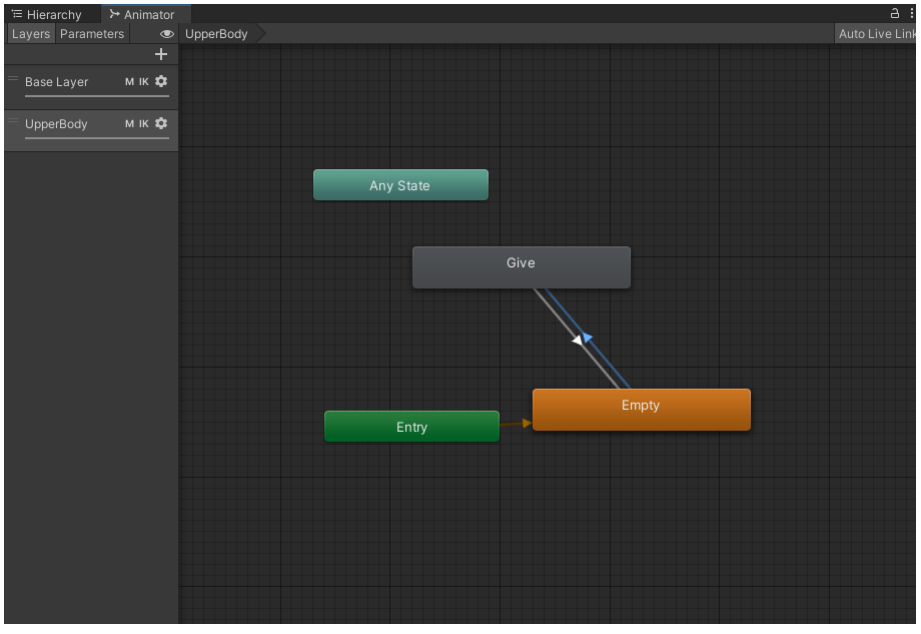

Figure 5.3: UpperBody Layer do animator controller do NPC

In the Upper Body layer of the animator controller, there is a simpler machine for the animations, since the only use it has is to make the NPC move its torso and arm to hand its identification to the user, and to receive it after inspection.

- The most complex component of the Driver NPC, is the DriverNPC C# Script that programmatically manages the actions the NPC takes throughout the simulation. It receives all of the Audio Clips it needs to reply to voice commands, as well as the EventSystem, the animators for the wheels and door of the vehicle, the Identification card Game Object (ID); the waypoints for leaving the simulation, getting back in the vehicle and the player's position; the walk and run speed.

- The Nav Mesh Agent is a component which activates and deactivates when the NPC exits and enters the vehicle respectively. It provides the NPC with the capability to process the generated Navigation Mesh for the environment and also to move along it with physics-based characteristics.

5.2.1 DriverNPC (C# script)

The DriverNPC C# script is the script that, as previously mentioned, programmatically manages the actions the NPC takes throughout the simulation.

Its methods are:

Method	Parameters	Return	Description
Start	None	void	MonoBehaviour method. Initializes every private component of this class.
Update	None	void	MonoBehaviour method. Controls some behaviours that need to be controlled by this script.
MuteEngine	None	void	Toggles the mute value of the Car Engine AudioSource
Reply	AudioClip reply	void	If the Driver isn't "speaking", it plays the AudioClip received as a parameter.
DriveOff	None	void	Begins the DriveOff animation of the Car and the respective sounds
ToggleId	string str	void	Given the parameter str received, toggles the ID gameObject in the simulation.
AddActionsToQueue	List<Action> actions	void	Adds each Action in actions to it's actionQueue List.
Act	Action ac	void	Given the Action received as a pramaneter and a Switch case for the Action's type, this methods executes the given Action, either by replying a voice recording, triggering some animation, control the Driver's NavMeshAgent component, controls the Car and it's sonds and animations, or a combination of the 4.

Table 5.1: DriverNPC methods

RefreshMove	None	void	Method that is called by the animator component when an animation finishes so that DriverNPC knows when it can start another animation.
SocketInteraction	None	void	Controls when the driver's Id is taken or given back to trigger its animation of retreating its arm.
LeaveCar	None	void	Moves the Driver out of the Car and enable its NavMeshAgent component.
EnterCar	None	void	Moves the Driver inside of the Car and disables its NavMeshAgent component.

Table 5.2: DriverNPC methods cont.

5.3 XR Interaction Toolkit implementation

The XR Interaction Toolkit is an important package used in this project as it provides an infrastructure for most popular VR sets and greatly reduces the difficulty of implementing a movement and interaction system controlled in Virtual Reality.

The first part of this package is the XR Origin Game Object, which essentially represents the user, as it possesses several scripts that prepare everything from the camera movement, an Input Action Manager for button presses, to a locomotion system and a Character Controller to traverse the environment.

Aside from this, it is necessary to choose the OpenXR option in the Project Settings in order to populate automatically all of the actions related to each button in the controllers. After doing so, it is possible to access each Input Action and use it to trigger interactions like opening a menu, grabbing an object, etc.

To make a Game Object be able to be grabbed by the user, the object must have an XR Grab Interactable component (aside from the usual collider and rigidbody components), which enables the object to be grabbed when the user presses and holds (by default) the

inner grip button on the controller. It is possible to change the Movement Type of each XR Grab Interactable to create different interactions between objects. The Movement Type parameter specifies how an object moves when selected, either through setting the velocity of the Rigidbody, moving the kinematic Rigidbody during Fixed Update, or by directly updating the Transform each frame.

Another way to interact with an XR Game Object is by adding an XR Socket Interactor component. A socket interactor allows the user to store an XR Grab Interactable object in a designated position which is highlighted in the colour blue if the object can be placed in the socket, or red if the socket is already holding an object or if that object type cannot be stored in that socket.

5.4 Wit.ai Implementation

Wit.ai (`https://wit.ai/`) is a natural language processing (NLP) platform that enables developers to incorporate voice recognition and understanding capabilities into their applications. By leveraging machine learning algorithms, Wit.ai allows for the extraction of valuable information from spoken language and turns it into structured data.

To integrate voice recognition into a Unity project using Wit.ai, these steps were taken:

- Training the Wit.ai platform with relevant voice data. This involves providing sample phrases or utterances and associating them with their corresponding intents or actions.
- Defining the different actions or intents the system should recognize from voice input.
- Creating utterances or phrases that Wit.ai must recognize to trigger an intent by typing it into an input field.
- Utilizing the Wit.ai Unity SDK to integrate the Wit.ai service into the Unity project. This SDK provides functionalities for recording audio, sending the recorded audio to Wit.ai for analysis and receiving the recognized intents or actions.
- Once the voice input is sent to Wit.ai, it will analyze the input and identify the associated intent. The Unity project can then process the recognized intent and trigger the corresponding actions.

The following images display the steps taken to create a voice command in a unity project:

Name ↑↓	Entities
Change_Colour	shape:shape, color:color

Figure 5.4: Creation of an intent

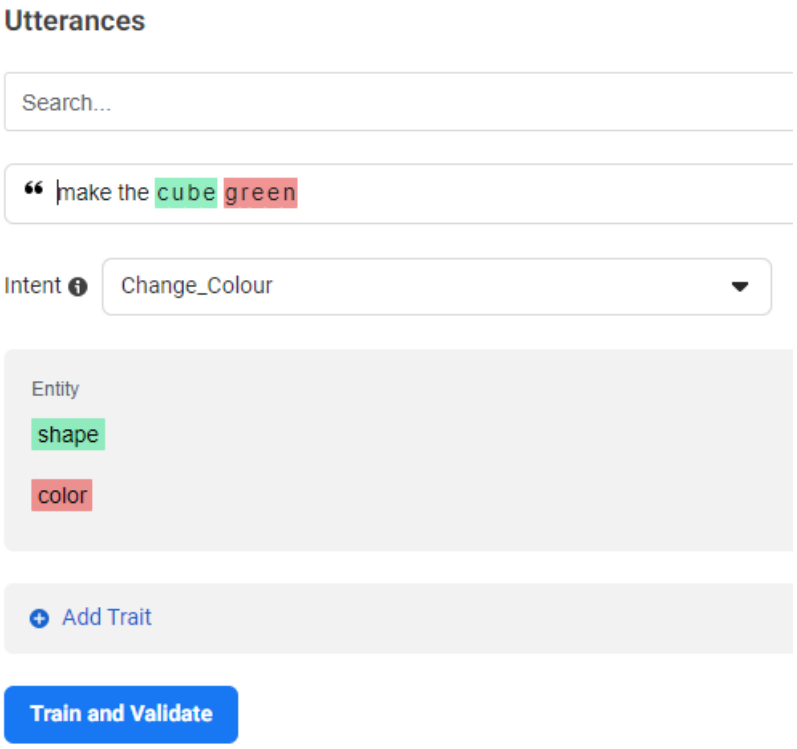

Figure 5.5: Associating an utterance to an intent

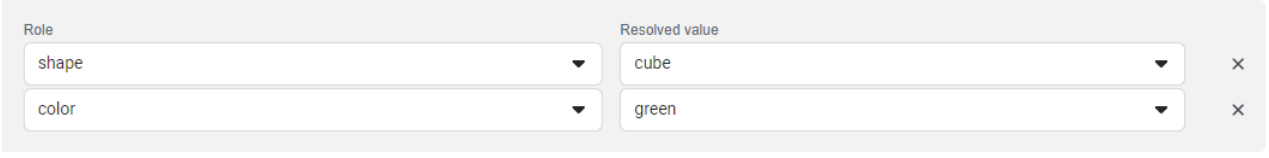

Figure 5.6: Binding words to entities

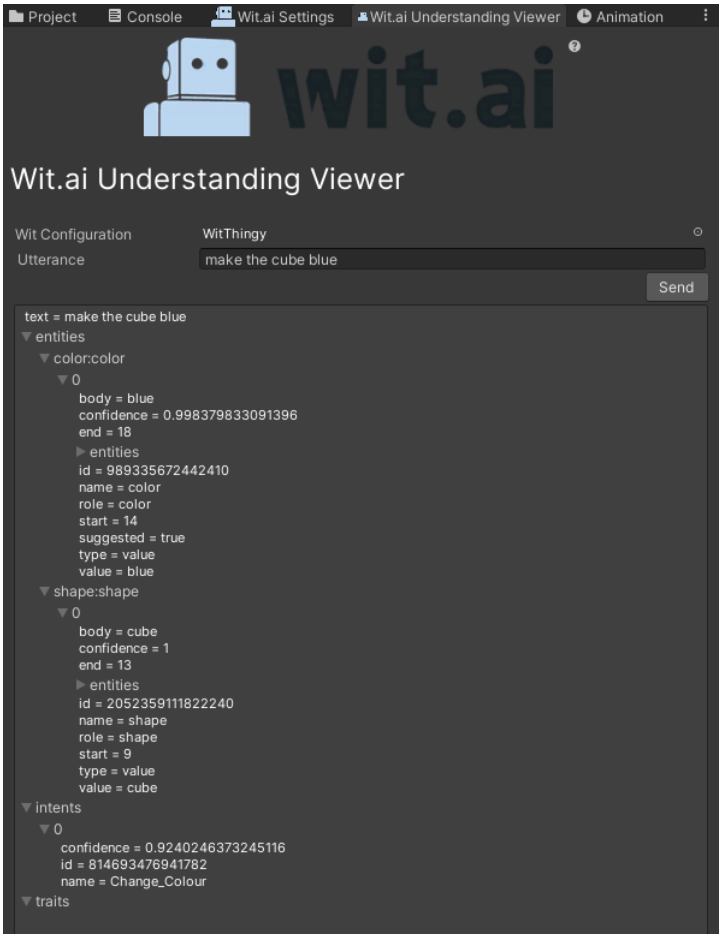

Figure 5.7: Testing the recognition in the Wit.ai Unity SDK

Lastly, the values obtained can be consumed programmatically to trigger any type of change in the simulation the user desires.

The commands and respective actions are presented in the table below, although only the word in bold has to be recognized in order to trigger the command:

Command	Intent
”Give me your **documents**”	Give_Documents
”Turn off the **engine**”	Engine_Off
”you can **go on**”	Drive_Off
”**here you go**”	Take_Documents
”**get in** the vehicle”	Get_In_Car
”**Stop**”	Stop

Table 5.3: Table with commands and intents

5.5 Events and Event System

Events are crucial in this project as they are the way in which the DriverNPC perceives the user's interaction with them. The class SimulationEvent is a simple container of SimulationEventType. It has a constructor that receives a SimulationEventType, and the following methods:

Method	Parameters	Return	Description
GetEventType	None	SimulationEventType	Returns its SimulationEventType type.
Equals	SimulationEvent ev	bool	Returns true if received SimulationEvent ev's type is equal to its type.
ToString	None	string	Returns its type as a string.

Table 5.4: SimulationEvent methods

The different types of Events described in SimulationEventType Enum are as follows:

- askedForDocuments
- askedToEnterVehicle
- askedToLeave
- askedToStop
- askedToTurnOffEngine
- enteredCar
- leftCar
- receivedDocuments
- turnedOffEngine

After implementing the different Events and classes that hold them, there needs to be a way of sending them. The class EventListener has 4 methods to be called by the Wit.ai component in the Unity Scene and EventSystem is a class with only 1 method: *SendEvent(SimulationEvent ev)*. When EventListener gets any of its methods called, it calls *SendEvent(SimulationEvent ev)* method from its instance of EventSystem. The method *SendEvent(SimulationEvent ev)* calls the method *ReceiveEvent(SimulationEvent ev)*. *ReceiveEvent(SimulationEvent ev)* of every DriverNPC it has access to.

5.6 States and State Machine

There can't be a State Machine without States. The first step in implementing the State Machine was the implementation of each State.

Firstly, an abstract class State was created with two private Lists of Actions, entryActions and exitActions, a constructor that initializes them and the following methods:

Method	Parameters	Return	Description
Evolve	SimulationEvent ev	State	Abstract method to be implemented in each State.
AddEntryAction	Action ac	void	Adds the received parameter ac to it's List of Actions entryActions
AddExitAction	Action ac	void	Adds the received parameter ac to it's List of Actions exitActions
GetEntryAction	None	List<Action>	Returns it's List of Actions entryActions
GetEntryAction	None	List<Action>	Returns it's List of Actions entryActions
Equals	State state	bool	Returns true if the received State is equals to itself

Table 5.5: State methods

After the abstract State was implemented, the remaining unique States were implemented, each one with its own Evolve method, expecting different SimulationEvents and with different thresholds of randomness as depicted in the State Chart Diagram previously mentioned (Link). The implemented States were:

- ApproachState
- EnterCarState
- ExitCarState
- FleeState
- LeaveEngineOffState

- LeaveState
- RunState
- TurnOffEngineEnforcementState
- TurnOffEngineState
- WaitEnforcementEngineOffState
- WaitEnforcementState
- WaitEngineOffState
- WaitOutsideState
- WaitState

The class StateMachine was implemented after every State was done. In it, there was a private State currentState to, as the name implies, store the State Machine's current State. The following methods were implemented:

Method	Parameters	Return	Description
Innit	None	List<Action>	Method in which the State Machine "decides" on it's Starting State.
Evolve	SimulationEvent ev	List<Action>	Method that evolves its currentState into its nextState if the conditions apply.
GetCurrentState	None	State	Returns it's currentState.

Table 5.6: StateMachine methods

After DriverNPC creates its StateMachine, it calls the method *Innit()* to initialize it. This method calculates 3 thresholds and randomly chooses the State Machine's starting State, as depicted in the State Chart Diagram previously mentioned (link). When DriverNPC calls the method *Evolve(SimulationEvent ev)* of its StateMachine, the State Machine calls its currentState's *Evolve(SimulationEvent ev)* method, and in it, the State checks its transitions and returns a new State if the conditions apply. Afterwards, if there is a nextState, the Exit Actions and Entry Actions of the currentState and nextState respectivly are compiled, the currentState is updated to the nextState and the method returns the List of Actions. This was explained in the Sequence Diagram previously mentioned (link).

5.7 Actions

After implementing all Events, EventSystem, EventListener, States and StateMachine, it was time to implement the Actions that the DriverNPC would execute during the simulation runtime.

Like when Events were implemented, the class Action is a simple container of ActionType. It has a constructor that receives an ActionType and the following methods:

Method	Parameters	Return	Description
GetActionType	None	ActionType	Returns its ActionType type.
Equals	Action ac	bool	Returns true if received Action ac's type is equal to its type.
ToString	None	string	Returns its type as a string.

Table 5.7: Action methods

The different types of Actions described in ActionType Enum are as follows:

- Approach
- DriveAway
- DriveAwayNoEngine
- EnterCar
- ExitCar
- ProvideDocuments
- ReceiveDocuments
- RunAway
- Stop
- TurnOffEngine
- WalkAway

The following list contains what voice line the DriverNPC replies with each action:

- ”Of course, no problem.” - Turn off Engine

- ”Yes sir. Her you go.” - Give Documents
- ”Is there a problem?” - Exiting vehicle
- ”Yeah, alright. Sorry.” - Getting back in vehicle
- ”Thank you sir, good afternoon.” - Leaving simulation after enforcement

Chapter 6

Usability

In order to evaluate the usability of this project, some in-person testing was hosted in the laboratory at ISEL. Some police officers, the project's advisors and even some LEIM students had the chance to try the simulation and give crucial feedback about it.

The objective of this testing session was to elaborate a survey with a variety of questions to evaluate the overall experience of the software. The survey was divided into 6 parts:

- Project contextualization: To briefly characterize the demographic of people who tested the project.
- Simulation realism: To evaluate if the animations, sounds and visuals have a natural and slightly realistic look, or if something feels unnatural or out of place.
- Simulation Experience: For questions about the pace and unpredictability of the events carried out in the simulation.
- About the objective: Questions to evaluate the learning experience and utility of the simulation.
- System Usability Scale (SUS, view Acronyms): The last section has several questions to evaluate the usability of the system. The score will be calculated and displayed.

It is important to note, from the characterization of the users who filled in the survey, that half of users responded saying they were or had been in law enforcement. This is because a group of the advisor Hélder's colleagues came to the laboratory to test the project.

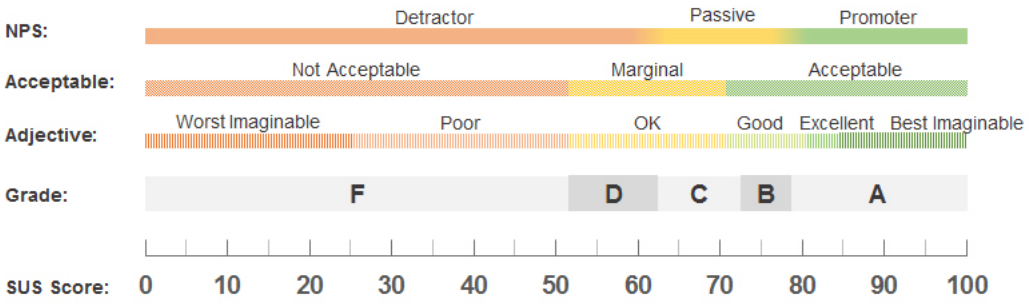

Figure 6.1: System Usability Scale

In terms of the questions about the simulation, the results were mostly positive, with scores of 4 to 5. In the question about the volume of the car's engine, most people voted "four", meaning the volume was slightly too loud. In the section about the simulation experience, one person selected 2 and another chose 3 as their answers in the question "the pace of events is close to reality". This could mean that the events carried out slightly slower than expected. Even though the objective was to make the reactions as quick as possible to create a realistic experience, the processing of the voice command takes about half a second to complete and that can disrupt the realism. Most importantly, before the SUS questions, a score of 5 out of 5 was dominant in the question: "I feel this simulator is useful to implement in law enforcement training". Since this is one of the main objectives of this project, the students conclude that the results are positive and the project mostly satisfied the user's expectations.

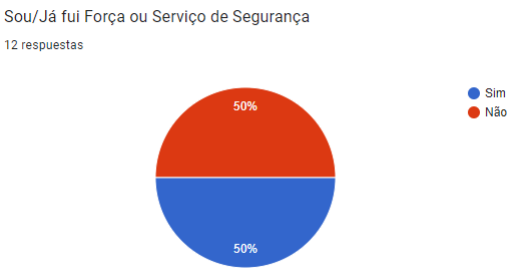

Figure 6.2: "I am or have been in law enforcement"

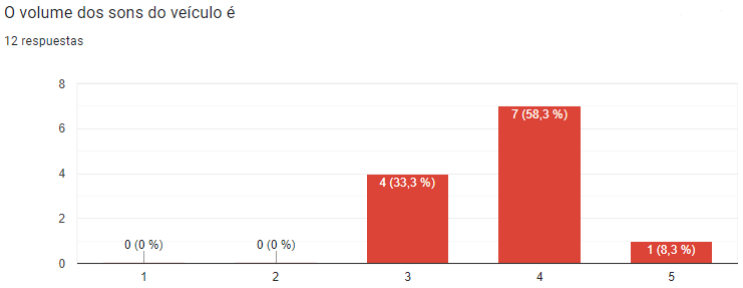

Figure 6.3: ”The volume of the vehicle sounds is adequate”

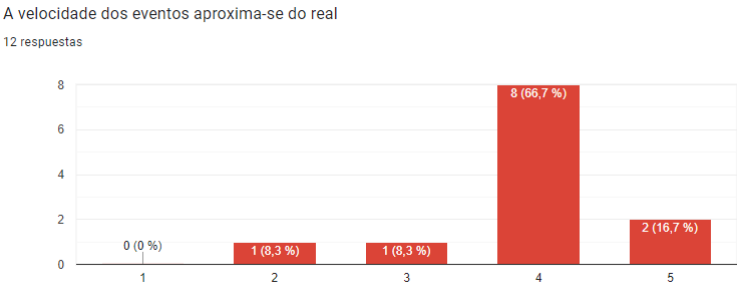

Figure 6.4: ”The pace of the events resembles reality”

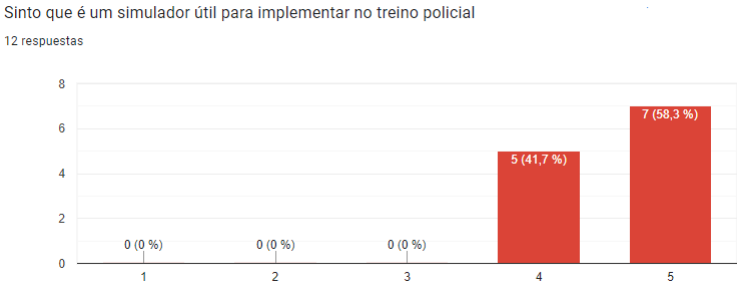

Figure 6.5: ”I feel the simulator is useful to implement in law enforcement training”

Lastly, there is a final ”additional comments” section where the users can freely write any feedback they deem necessary to complement the replies in the survey. One user wrote: ”the simulation lacks (and deserves!) complementary situations... hand to holster, have ambidextrous holsters, more variety of scenarios, more dangerous scenarios, etc... the aplicability in training is evident!”. While an overall positive review, this comment gives important suggestions about details that could make the project more complete and give a better user experience.

After calculating the SUS score of each participant, the final score attributed to the SUS survey of this project is 70.2, which falls into the ”Good” category, right behind ”Excellent” and inside the ”Acceptable” margin of the System Usabilty Scale.

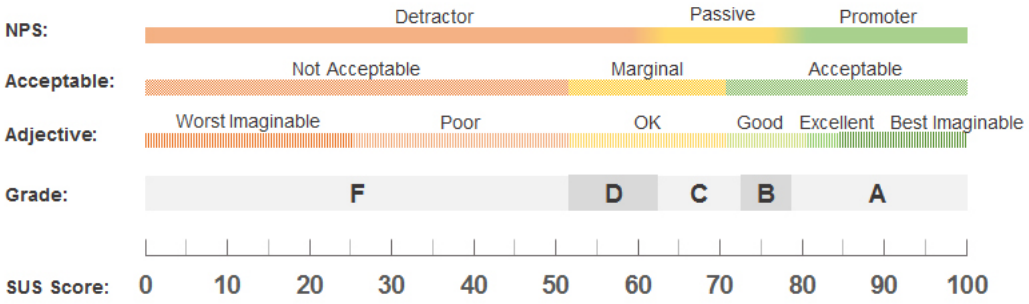

Figure 6.6: System Usability Scale

Apart from the above results, the users gave the following extra verbal feedback:

- Documents fall too easily, and are not easy to pick back up.
- The gun is not on the right side of the belt.
- The ability to run is important.
- Commands are not easy to remember.
- Reaching for the gun should be an event to deescalate the situation.

After evaluating the user's experiences and how they completed each simulation the students arrived at the following conclusions:

- Commands should be on both controllers and not only the left one.
- Users tend to hold the button to speak instead of only pressing it.
- In order for the users to learn how to use the system and navigate the simulation there is the need of a tutorial or at least an hands-on experience.

Chapter 7

Conclusions and Future work

7.1 Conclusions

The students proposed to develop prototype of a simulation for law enforcement agents in training to use and to more quickly, with less effort and better learn how to conduct a checkpoint operation where anything can happen. The students thing they managed to build a strong foundation to any possible future work that could be built on top of what they already built. The students didn't implement any violence from the Driver and didn't implement a way of arresting the Driver.

After evaluating the results of usability they came to the conclusion that a system like this could be extremely beneficial to train new law enforcement agents and that even the less adept with technology can learn how to maneuver this simulation with ease.

The students are proud of the work they did and eager to see it take off as either a real simulator (not a prototype) or a foundation for future projects in ISEL.

7.2 Future work

Being a prototype, this project still has a lot that can be done and improved. The following list depicts the major addictions that the students found could be added to this project in future iterations:

- The ability to run.
- More complex behaviors.
- Reaching for the gun should be an event to deescalate the situation.
- Additional people in the car.
- Different environments.

- A Tutorial simulation to teach basic movements, controls, and voice commands.
- Different simulation settings(post pursuit for example).
- Multi-user experience (Police usually work in pairs).
- More advanced AI that responds differently to different behaviors and events.
- Save simulation presets.
- Recording the screen and replaying it.

As far as what they thing can be improved a list was compiled with the improvements they found more important:

- Documents fall to easily, and are not easy to pick back up.
- The gun is not on the right side of the belt.
- Commands are not easy to remember.
- Commands should be on both controllers and not only the left one.
- Users tend to hold the button to speak instead of only pressing it.

Appendix A

Models

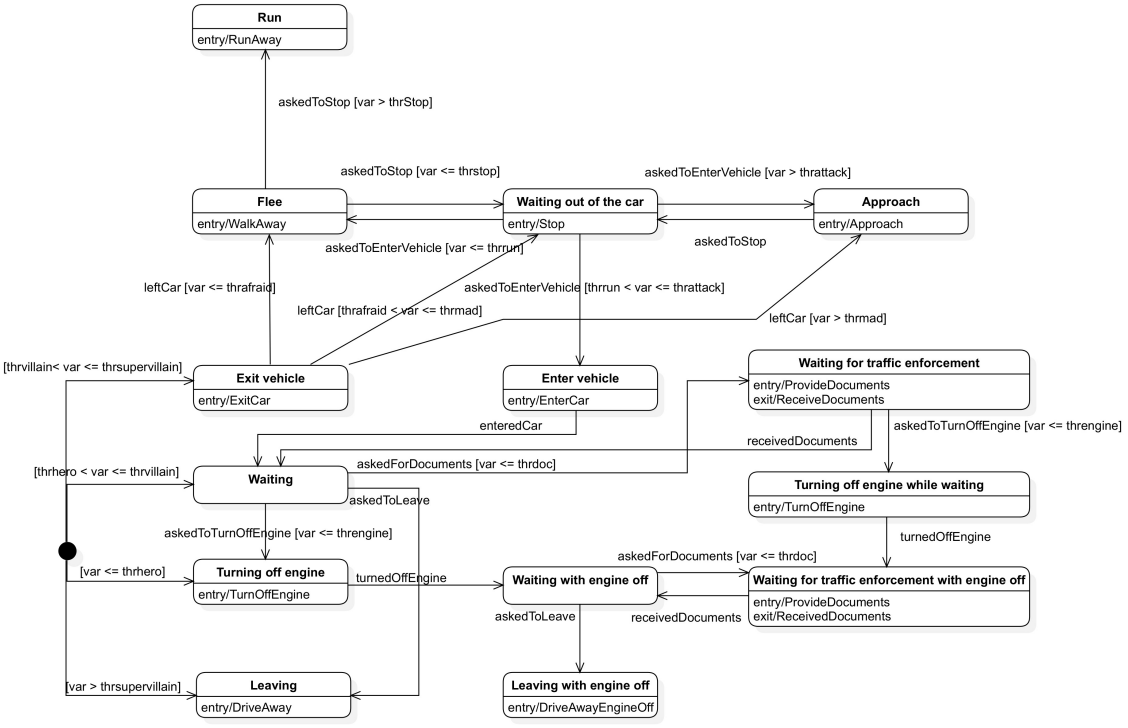

Figure A.1: VRCop NPC State Chart Diagram

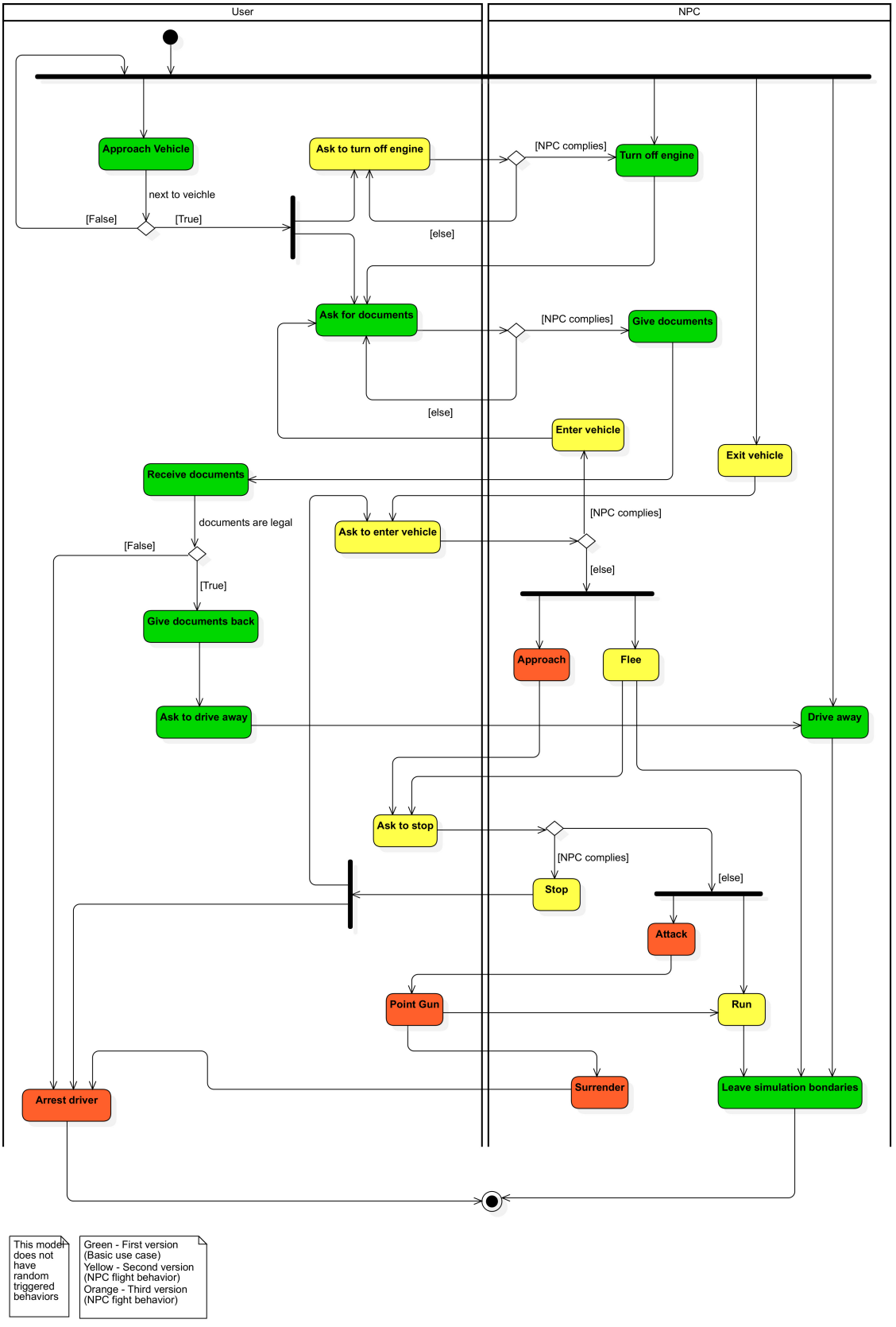

Figure A.2: VRCop Activity Diagram

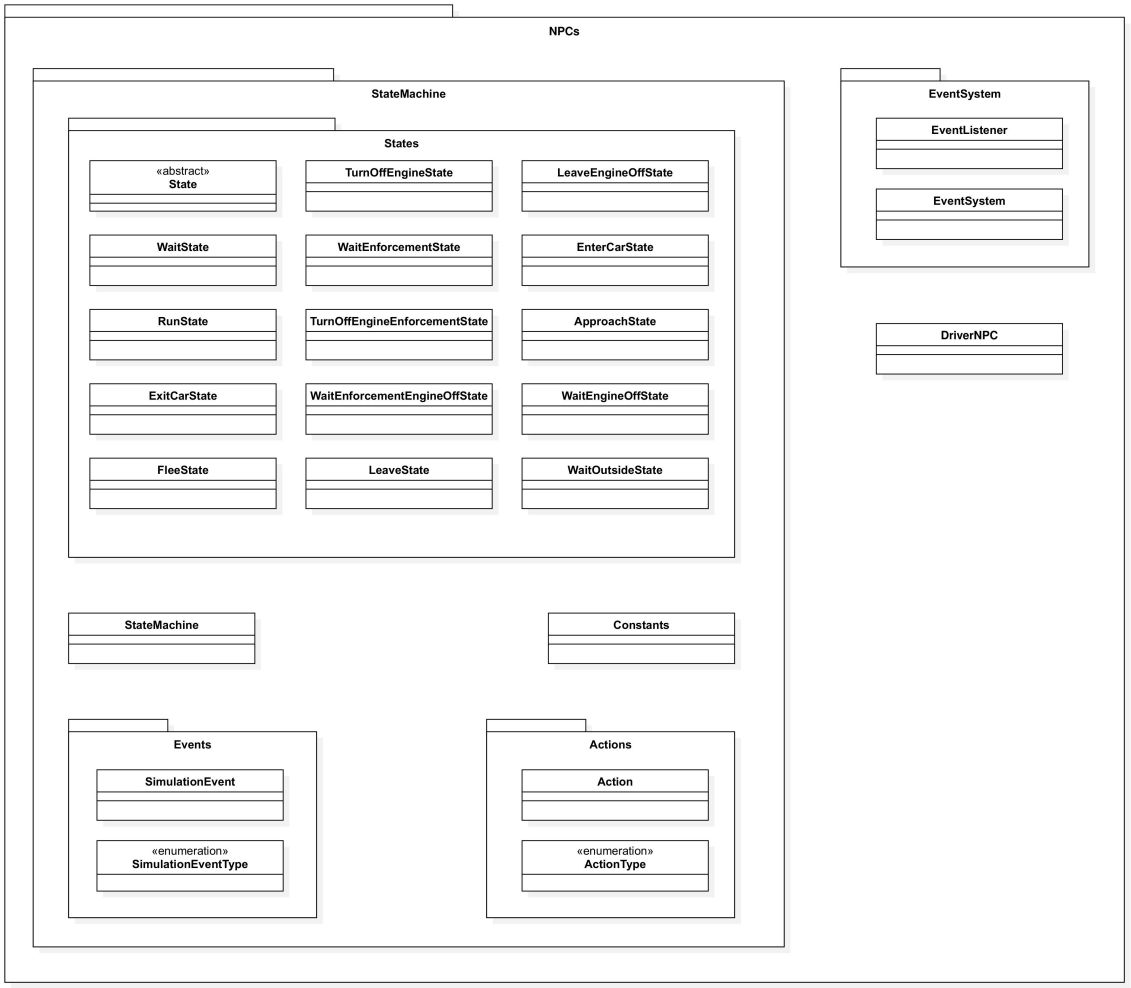

Figure A.3: VRCop NPC Packages

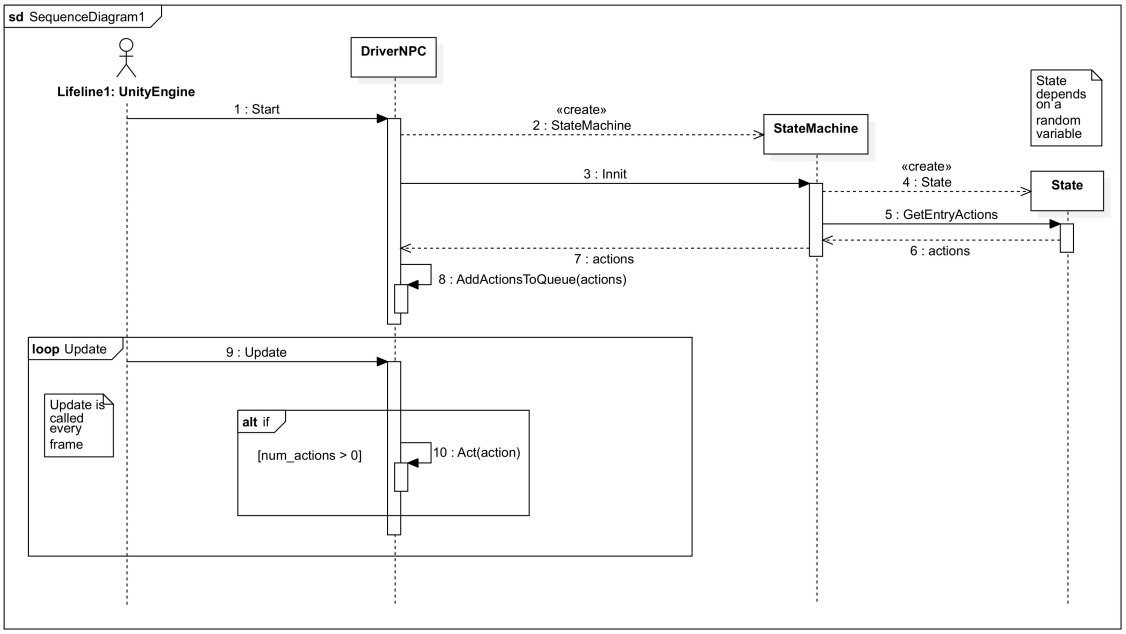

Figure A.4: VRCop Simulation run-time in the NPC's perspective Sequence Diagram

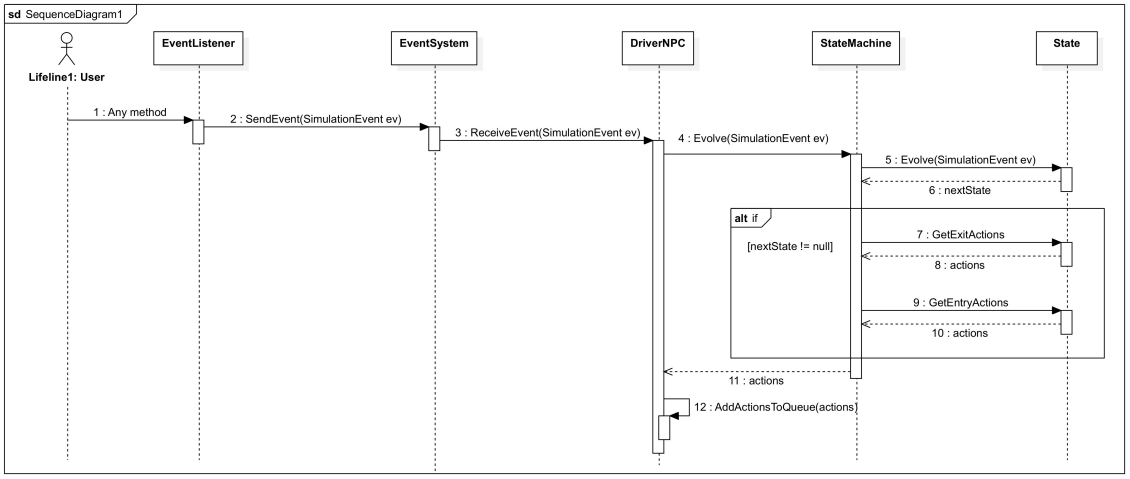

Figure A.5: VRCop Event Management Sequence Diagram

Bibliography

1 *SimX, SimX: Virtual Reality Medical Simulation. SimX; 2021. Available At* `https://www.simxvr.com/`

2 *Aesir Interactive, Police Simulator - Patrol Officers. astragon Entertainment; 2022. Available at:* `https://store.steampowered.com/app/997010/Police_Simulator_Patrol_Officers/`

3 *Kinetic Games, Phasmophobia. Kinetic Games; 2020. Available at:* `https://store.steampowered.com/app/739630/Phasmophobia/`

4 *Unity Technologies, Unity 2021.3.21f. Unity Technologies; 2023. Available at:* `https://unity.com/releases/editor/whats-new/2021.3.21`

5 *Unity Technologies, XR Interaction Toolkit. Unity Technologies; 2018. Available at:* `https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.3/manual/`

6 *Wit.ai, Wit.ai. Meta; 2013. Available at:* `https://wit.ai/`

7 *Audacity, Audacity. Audacity; 2000. Available at:* `https://www.audacityteam.org/`