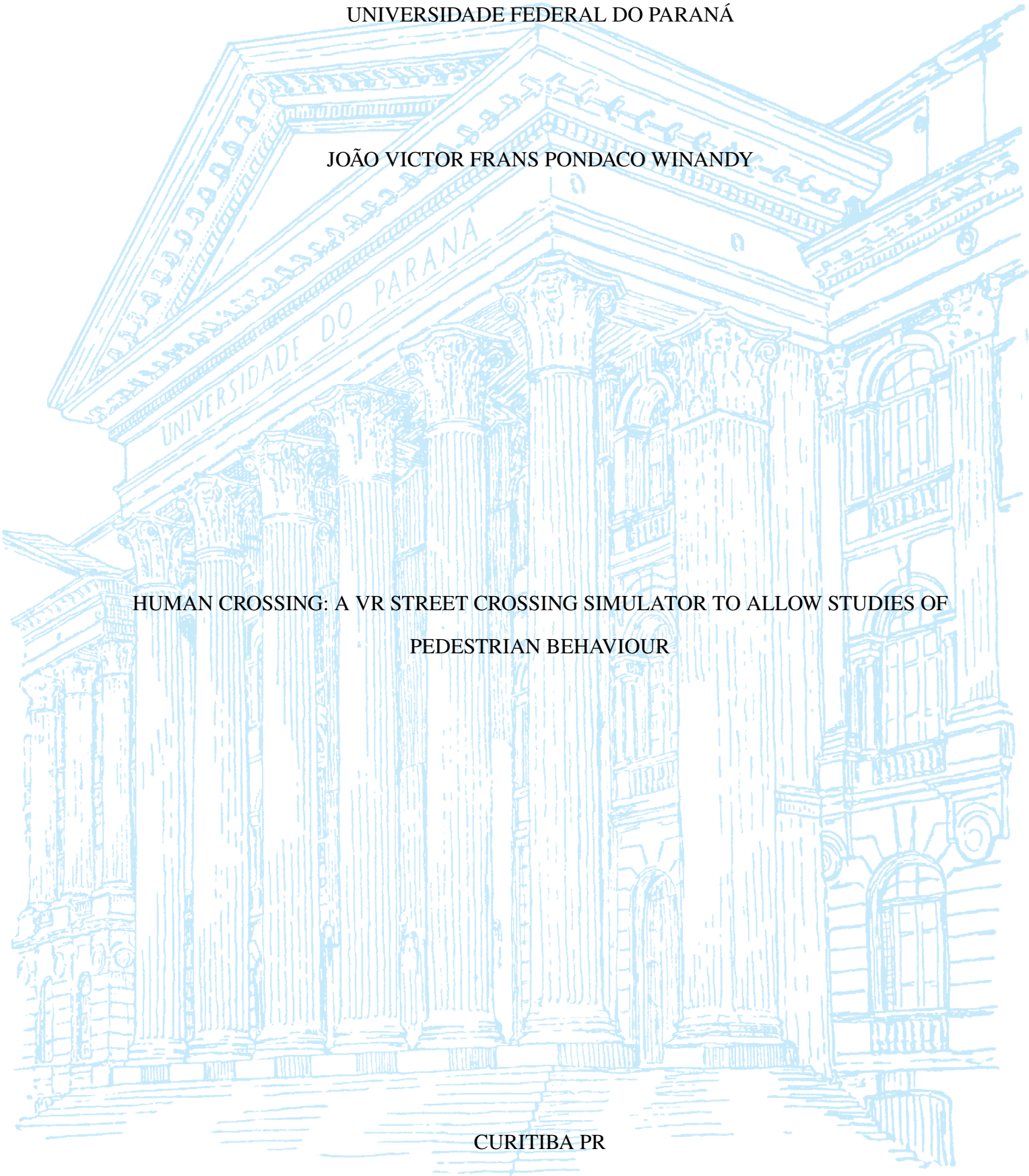
UNIVERSIDADE FEDERAL DO PARANÁ

JOÃO VICTOR FRANS PONDACO WINANDY



HUMAN CROSSING: A VR STREET CROSSING SIMULATOR TO ALLOW STUDIES OF PEDESTRIAN BEHAVIOUR

CURITIBA PR

2023

JOÃO VICTOR FRANS PONDACO WINANDY

HUMAN CROSSING: A VR STREET CROSSING SIMULATOR TO ALLOW STUDIES OF PEDESTRIAN BEHAVIOUR

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Eduardo Todt.

CURITIBA PR

2023

Ficha catalográfica

Substituir o arquivo `0-iniciais/catalografica.pdf` pela ficha catalográfica fornecida pela Biblioteca da UFPR (PDF em formato A4).

Instruções para obter a ficha catalográfica e fazer o depósito legal da tese/dissertação (contribuição de André Hochuli, abril 2019. Links atualizados Wellton Costa, Nov 2022):

1. Estas instruções se aplicam a dissertações de mestrado e teses de doutorado. Trabalhos de conclusão de curso de graduação e textos de qualificação não precisam segui-las.
2. Verificar se está usando a versão mais recente do modelo do PPGInf e atualizar, se for necessário (`https://gitlab.c3sl.ufpr.br/maziero/tese`).
3. conferir o *checklist* de formato do Sistema de Bibliotecas da UFPR, em `https://bibliotecas.ufpr.br/servicos/normalizacao/`
4. Enviar e-mail para "`referencia.bct@ufpr.br`" com o arquivo PDF da dissertação/tese, solicitando a respectiva ficha catalográfica.
5. Ao receber a ficha, inseri-la em seu documento (substituir o arquivo `0-iniciais/catalografica.pdf` do diretório do modelo).
6. Emitir a Certidão Negativa (CND) de débito junto a biblioteca, em `https://bibliotecas.ufpr.br/servicos/certidao-negativa/`
7. Avisar a secretaria do PPGInf que você está pronto para o depósito. Eles irão mudar sua titulação no SIGA, o que irá liberar uma opção no SIGA pra você fazer o depósito legal.
8. Acesse o SIGA (`http://www.prppg.ufpr.br/siga`) e preencha com cuidado os dados solicitados para o depósito da tese.
9. Aguarde a confirmação da Biblioteca.
10. Após a aprovação do pedido, informe a secretaria do PPGInf que a dissertação/tese foi depositada pela biblioteca. Será então liberado no SIGA um link para a confirmação dos dados para a emissão do diploma.

Ficha de aprovação

Substituir o arquivo 0-iniciais/aprovacao.pdf pela ficha de aprovação fornecida pela secretaria do programa, em formato PDF A4.

RESUMO

De acordo com uma estimativa da OMS feita em 2018, um pedestre morre no trânsito a cada 106 segundos. Um número tão elevado que os países membros da Organização das Nações Unidas definiram como objetivo reduzir pela metade esse número até o final da década, porém, para reduzir este número, é necessário primeiro compreendermos os fatores que causam essas mortes.

Com o avanço da tecnologia de realidade virtual nas últimas décadas, óculos de realidade virtual vem se tornando menores, mais capazes e mais baratos a cada ano. Por serem capazes de proporcionar um ambiente seguro e controlado, são um excelente dispositivo para estudar o comportamento dos pedestres. Neste trabalho, desenvolvemos uma plataforma para executar simulações de travessia de pedestres, com a possibilidade de pedestres andarem tanto fisicamente no mundo real ou atráves de controles. Em destaque, nossa plataforma permite a manipulação de parâmetros em quatro cenários distintos, facilitando a análise de diferente cenários. Em complemento, uma simulação de tráfego complexa acrescenta uma camada de realismo crucial para a compreensão da interação entre pedestres e veículos.

No centro de nosso sistema, está um sistema de coleta de dados, que fornece aos pesquisadores extensos logs e um sistema de replay para uma análise aprofundada dos pedestres após a simulação. O resultado final é um simulador, criado para estudar vários cenários de forma abrangente. A plataforma desenvolvida tem condições de avançar as análises do comportamento dos pedestres, oferencendo a pesquisadores uma ferramenta sofisticada e versátil para a realização de experimentos.

Palavras-chave: Simulação de pedestres. Realidade virtual. Simulação de trânsito. Comportamento de pedestres.

ABSTRACT

According to a WHO estimation made in 2018, every 106 seconds a pedestrian dies on the road. A number so large that the United Nations set the goal of halving it by the end of the decade, but to be able to reduce this number, we need to first understand what causes those deaths.

With the advance of Virtual Reality in the past decades, head-mounted displays have been growing smaller, more capable, and cheaper with each year. By being able to provide a safe and controlled environment, they are an excellent device for studying pedestrian behavior. In this work, we developed a platform to run crossing simulations, with the possibility for a pedestrian to walk physically in the real world or with motion controllers. Notably, our platform allows for the manipulation of parameters across four distinct scenes, facilitating the examination of diverse scenarios. Complementing this, a complex traffic simulation adds a layer of realism crucial for understanding the interplay between pedestrians and vehicles.

At the core of our system is a data collection system, providing researchers with extensive logs and replays for in-depth analysis of pedestrian behaviour post-simulation. The end result is a simulator crafted to study multiple scenarios comprehensively. The developed platform stands poised to advance pedestrian behaviour analyses, offering researchers a sophisticated and versatile tool to run experiments on.

Keywords: Pedestrian simulator. Virtual reality. Traffic simulation. Pedestrian behavior.

LIST OF FIGURES

LIST OF ACRONYMS

3D	Three dimensional
3DoF	Three Degrees of Freedom
6DoF	Six Degrees of Freedom
AI	Artificial Inteligence
API	Application Programming Interface
AR	Augmented Reality
AV	Automated Vehicles
CRT	Cathode-ray tube
FOV	Field of View
HMD	Head-mounted Display
IR	Infrared
JSON	JavaScript Object Notation
UFPR	Universidade Federal do Paraná
VR	Virtual reality

CONTENTS

1 INTRODUCTION

According to a WHO estimation, every 106 seconds a pedestrian dies on the road [2]. A number so large that halving the number of deaths and injuries from road crashes by 2020 was defined as one of the targets in the Sustainable Development Goals, adopted by all United Nations Member States [1, goal 3.6]. But to be able to reduce those numbers we have to first be able to understand the causes.

The motivations driving a pedestrian towards a crosswalk before traversing a road form a complex interplay of variables. Questions arise regarding the impact of vehicle density, vehicle speeds, distance to the crosswalk, and the time constraints individuals face. While efforts have been made over the decades to analyze pedestrian behavior in real-world settings, these methods are often impractical, costly, and may compromise subject safety[3]. Analyzing video camera footage, though common, fails to capture the nuanced reasons behind road accidents [3], as each pedestrian's situation is unique.

To address these challenges, simulators have emerged as a valuable tool. Those devices simulate the environment but use the user input to see how a person reacts to the created world. Historically, simulators have been employed in the aviation community for almost a century[32, c. 9.4.1]. They have been used extensively not only to facilitate learning but also to enable the replication of challenging scenarios, aiding in understanding human reactions, and leading to improvements in the safety of the crew and passengers.

The quest for realism in simulations has led to the rise of Virtual Reality (VR) devices, offering an immersive experience superior to traditional setups, which before compromised of multiple screen setups. Those devices consist generally of two screens placed very close to the face of the user, like an enclosed glass, making the virtual world the only thing the user can see. They may also contain several sensors to detect the user's movement, being able to track all movements that the user can do in the real world, having 6-degrees of freedom.

Those VR devices have been getting increasingly more powerful and cheaper, being currently cheaper than building a multiple-screen setup. Using a VR device also has increased user immersion compared to the alternatives [6]. In this work, we leverage the advancements in VR technology to create a simulated environment that closely mimics real-world street crossings.

Beyond hardware considerations, the realism of the simulated world is paramount. This work delves into the simulation of realistic streets, complete with authentic traffic scenarios, providing a nuanced platform for understanding how pedestrians interact with and navigate through dynamic environments. Furthermore, the developed program collects and stores the crossing data, for later analysis. The platform developed aims to empower traffic researchers with a powerful resource for in-depth analysis of pedestrian behaviour.

1.1 OBJECTIVES

This research aims to develop a VR application that simulates the traffic in a street realistically. The primary goal of this simulation is to understand the intricacies of pedestrian behaviour during street crossings. The overarching aim is to provide a platform that can identify conditions contributing to an elevated risk for pedestrians, paving the way for subsequent research aimed at mitigating these risks effectively.

For that, we will develop a realistic and dynamic traffic simulation, to create an authentic environment to facilitate the study of the interplay between pedestrians and vehicles. Those

vehicles should also be capable of interacting with one another and responding to environmental cues, such as wielding to a pedestrian waiting to cross through a crosswalk.

The developed simulator should also have a robust data collection system, providing researchers with extensive data to be able to discern critical factors influencing pedestrian risk.

1.1.1 Specific goals

As a set of goals to achieve our objective, the developed simulator should:

- Investigate strategies to enhance the affordability and accessibility of the simulator, ensuring it caters to a broad user demographic.
- Explore techniques to maximize user immersion within the virtual world, enhancing the realism of the simulation experience.
- Experiment with customization options within the simulator, allowing researchers to tailor the simulated experience based on their specific requirements.
- Investigate methods to increase the complexity of traffic scenarios within the simulator, providing a more nuanced understanding of pedestrian interactions in varied environments.
- Experiment with randomization techniques to diversify simulator scenarios, increasing the scenarios the simulator can analyze.
- Research and incorporate alternative test scenes, expanding the diversity of situations the simulator can replicate and analyze
- Develop a streamlined mechanism for running multiples scenarios in quick succession, facilitating experiments.
- Investigate metrics and methodologies for researchers to study pedestrian behavior post-crossing, considering variations between studies.

1.2 STUDY OUTLINE

This study is structured in the following way:

Chapter 2 - Background provides a brief overview of the definition of virtual reality. It explores the different types of virtual ecosystems and it gives a brief look at the evolution of virtual reality. This chapter also presents the hardware that will be used for the proposed system and provides insights into the chosen Game Engine that is used for the simulator implementation.

Chapter 3 - Related Works this chapter reviews past approaches to studying pedestrian behaviour within virtual systems. By analyzing prior works, it seeks to contextualize the proposed system, so we can improve on the current simulators.

Chapter 4 - Proposal describes our proposed system. It gives insights about the environment created, describes how the vehicles move and the methods we employ for controlling them, as well as how the user can interact with the world. Additionally, it describes the log that is outputted from a crossing, and what goes into it.

Chapter 5 - Results and validation in this chapter we will be talking about the end result of this work, and validating the functionalities described in chapter 4.

Chapter 6 - Conclusion summarises the study's proposal and achievements and offers an analysis of what was created within the defined objectives. Furthermore, it suggests opportunities for future research and possible enhancements to expand the simulator's capabilities.

2 BACKGROUND

In this chapter, we will explore the definition of virtual reality and its differences from augmented reality. We will also be taking a look at the past and current virtual reality devices, as well as their strengths and drawbacks. Finally, we will be providing insight into the engine we choose for the implementation of our simulator.

2.1 VIRTUALITY CONTINUUM

To investigate pedestrian behaviour while crossing a street, a way to avoid subjecting individuals to the inherent risks of real-world testing is needed. For some time, those kinds of simulations have been made using a multiple-screen setup to allow the users to see in multiple directions. However, this approach has dwindled in popularity in the past few years. The main reason is advancements in Virtual Reality devices, which currently allow for a more immersive experience, while also being possibly cheaper.

But first, it's important to define the concepts of virtual reality to better understand this work. Paul Milgram and Fumio Kishino introduced the concept of "virtuality continuum" in their work[21]. There, they defined this term as the continuum space that goes from reality to a fully virtual environment, with Mixed Reality being everything in the middle of this spectrum, as illustrated in the figure 2.1.

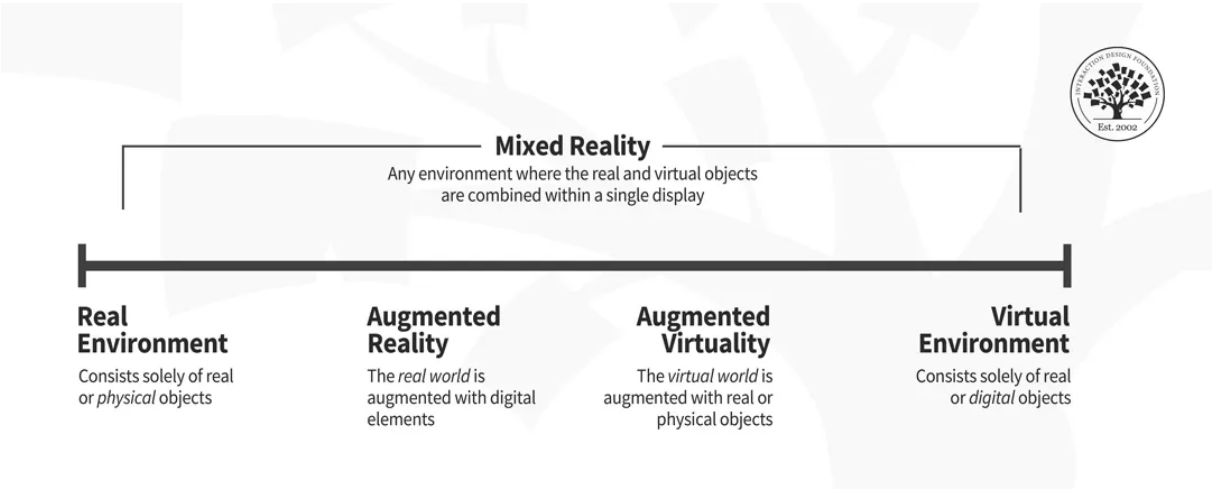


Figure 2.1: Representation of the virtuality continuum. Source: Laya Tremosa and the Interaction Design Foundation [15].

As we walk through the continuum, we first encounter **Augmented Reality (AR)**, which consists of augmenting the real world with digital elements[15]. A noteworthy example is the widely popular game Pokemon GO, developed by Niantic for iOS and Android, where players could see "Pokemons" in the real world using their smartphone cameras. According to Forbes, Pokemon GO was very popular, engaging 147 million users in May 2018 [26].

Beyond AR lies **Augmented Virtuality (AV)**, wherein the virtual world incorporates real or physical objects [15]. A good example of this would be being able to visualize a user's hand within a fully virtual world.

The **Virtual Reality (VR)** is then defined as a world consisting of only digital elements. This is the spectrum of the continuum that this work will focus on. While other approaches have

explored using AR in the past to study pedestrian behaviour [19], we decided to use a fully virtual world. This allows us to simulate the full crossing sequence, by having the user walk through a virtual and controlled street, in a way that the generated metrics can be translated into the real crossing behaviour [3, 7, 19].

2.2 VIRTUAL REALITY SYSTEMS

The first use of VR hardware traces back to 1968 when Ivan Sutherland created a head-mounted display (HMD) with miniature CRTs called "The Sword of Damocles" [25] (Figure 2.2). This device allowed users to perceive three-dimensional spaces, by presenting two images at slightly different angles and thus giving the perception of perspective. Even though stereo images are important for a three-dimensional illusion, an image that changes in a natural way as the observer moves his head is even more so [25]. So, Sutherland's HMD also contained head position sensors to be able to translate and rotate the objects accordingly. Sutherland's HMD, while rudimentary by today's standards, laid the foundation for contemporary VR systems.

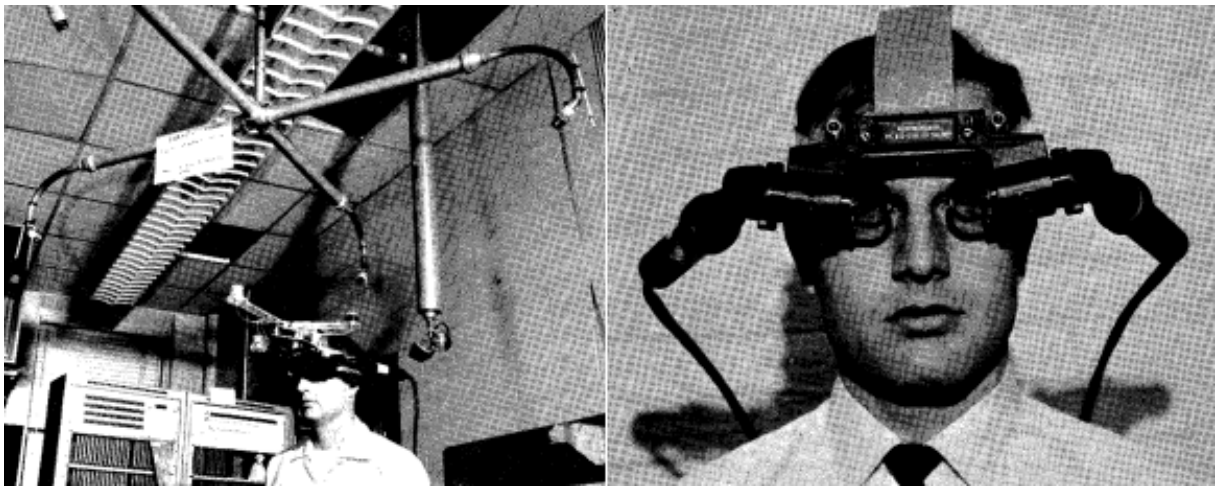


Figure 2.2: Sutherland's Sword of Damocles HMD [25].

The fundamental structure of modern HMDs remains rooted in Sutherland's HMD. Comprising two (or sometimes one split) screens, lenses to focus and zoom the images, so it can take more of the user's field of view (FOV) and sensors for tracking the head's translation and rotation. Head-mounted displays are currently the main hardware used for VR, with large companies like Sony, Microsoft, Valve, Meta, and Apple having their own devices.

In recent years, HMDs have become accessible to the general public and their usage has been growing steadily. In the 2010s the advancement of the technology made that the first devices fully capable of achieving 6 degrees of freedom[1] and a visual fidelity never seen before. They also introduced motion controllers, which the user can move freely, being used as the user's hands in the virtual world. Unfortunately, those high-fidelity devices required a powerful computer to be able to render the games at the resolution and frame rate needed for a good experience, which on top of the prices of those devices, made the public that could afford them quite limited. They also made use of tracking stations to be able to properly track player movement, requiring a room with properly placed sensors. Figure 2.3 shows how the tracking of those devices operates.

[1]Degrees of freedom here refers to the number of ways an object can move through the 3D space. Three of them correspond to the rotational movement around the x (pitch), y (yaw), and z (roll) axes. Devices that can only track the head through those three have 3DoF. Devices that can also track the translational motion along those axes, have 6DoF, making all of the user's movement to be reflected in the virtual world.

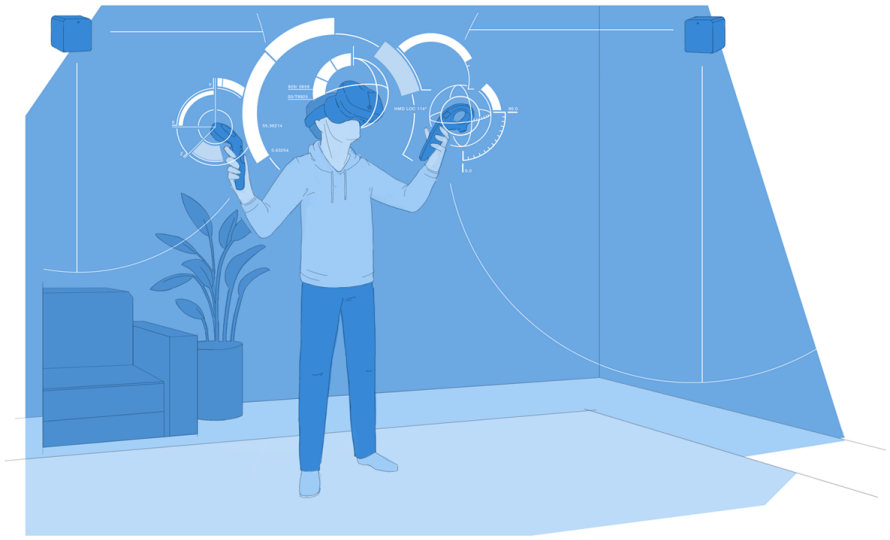


Figure 2.3: SteamVR™ Tracking bases, and a person with a HMD and motion controllers [30].

In June 2014, Google created a new platform for VR, giving a simpler experience, with lower resolution and only giving 3 degrees of freedom, with their Cardboard platform (Figure 2.4). The Google Cardboard was an accessory that transformed normal smartphones into viable VR devices, by placing two lenses on the smartphone screen, making use of the device's Gyroscope and Accelerometer, to track rotation.

Figure 2.4: Google Cardboard [13].

This allowed for the general public to get a taste of VR technology. However, the low resolution of those devices, caused by the splitting of the normal smartphone's screen, as well as the limited refresh rates, and the not-so-precise sensors, caused the experience to be not ideal. The lack of proper tracking of the head's translation, influenced strongly to cases of motion sickness among users [16]. Although there have been studies to enable 6DoF using the Google cardboard [23], it was still not fully achieved. While the Google Cardboard was simpler than

some of the HMDs being shipped, those were a lot more affordable and accessible, making it one of the first VR experiences for many people in the world.

In May 2019, the company Oculus released a powerful device that shaped the current trend of VR, with the release of a 6DoF, standalone, high-resolution device, the first Oculus Quest. This device made use of a powerful smartphone chip, which allowed more demanding apps to be run at a resolution of 1440x1600 per eye and 72 frames per second [23]. To be able to achieve 6DoF, the Quest makes use of 4 IR cameras to track the user's movement by mixing the data from the cameras using computer vision and its sensors.

Figure 2.5: Meta Quest 2 and its motion controllers [20].

Shortly after, in October 2020, a new version of this standalone device was released, the Oculus Quest2, later renamed to Meta Quest 2 (Figure 2.5). Making use this time of a modified smartphone chip, made with VR in mind, being able to run at a resolution of 1832 x 1920 per eye, and a frame rate of 90 frames up to 120 frames per second. The device was also priced cheaper than its predecessor at U$300.00, making it more affordable to the general public. The Quest 2 can also connect wirelessly to a PC, being able to make use of more powerful hardware if the user has access to it. On February 2023 Verge reported that the, now called, Meta had sold nearly 20 million Quest headsets [14].

Because of its portability, power, and accessibility, we will be making use of a Quest 2 in standalone mode for the experiments on this work. However, the system developed works in any modern VR Headset that supports the OpenXR standard.

2.3 GAME ENGINE

A game engine is a framework for creating virtual worlds, offering a suite of tools to facilitate user interaction. They usually come with a world editor, where one can place 3D models and associate them with scripts. Beyond visual elements, game engines incorporate a physics framework, sparing developers from reinventing common physics calculations like gravity, traction, friction, and drag. Additionally, engines also abstract various graphics APIs, enhancing cross-platform compatibility and reducing hardware dependencies.

For the development of our simulator, three engines were considered: Unity 5, Unreal Engine 4, and Godot 3. To choose between those, the following key aspects were considered:

VR support and documentation - While VR device usage has been on the rise in the past decade, it still consists of a small portion of all the games released yearly. According

to the user tags on Steam, only 12% of the games released from 2015 to 2022 have VR support [24]. While all three engines support VR game creation, Unity and Unreal Engine were favored due to more established official support from hardware manufacturers. Godot, lacking an official Oculus SDK, required additional effort for input method compatibility.

Asset library - Given the primary focus of this work is to create a platform to study the behaviour of pedestrians, having a robust asset package facilitates development by providing steadily usable 3D models, letting us focus our efforts on other parts of the system besides 3D modeling. Unity boasts the most extensive asset library, having over 50 thousand entries in the 3D category [29], followed by Unreal Engine with close to 40 thousand entries in all categories [10], and Godot having only around 2 thousand items [27].

A large community - A big community of developers creating programs on the same platform increases the chance of someone already having found some of the same problems as ours, and documenting the possible solutions. This is even more important when developing to a relatively new device, with a heavy performance requirement, since it has to render a world at close to 4k resolution while having only a mobile chip and form factor. According to the Unity CEO, in 2018, more than 50 percent of mobile games, and more than 60% of AR/VR content were built in Unity [8].

Considering those factors, we ended up choosing Unity 5 for the development of our simulator. Its robust VR support, extensive asset library, and large and active community align with the project's objectives, ensuring a solid foundation for our simulator.

3 RELATED WORKS

As the usage of VR devices grows, so does the use of them in research. In the past few years, there have been a significant number of attempts to study pedestrian behaviour, in the safe environment that the virtual world provides. In special, we have been seeing a growth in the number of works that study the difference in pedestrian behaviour between automated vehicles (AV) and vehicles with a human driver.

Deb et al. [7] made a study in 2017 about the efficacy of virtual reality in pedestrian safety research. They developed a simulator in Unity, using an HTC Vive HMD, with the intent to test pedestrian behaviour when crossing through a four-way intersection with pedestrian crosswalks and traffic and pedestrian signals. In their simulator, the vehicles would first go through a green signal, after some time, the traffic signal would turn red and the pedestrian green, then, a last vehicle would come that could either stop in one of the lanes or go through the red signal. Their simulator, calculated data on the minimum gap between the pedestrian and vehicles, the number of collisions, walking speed, and crossing time, as well as gathered the subjective experience of the participants through three questionnaires.

Deb et al. concluded that the participants walked realistically, by having an average walking speed that matches the real-world data. Their questionnaire also concluded that the users found the virtual environment to be a realistic pedestrian simulator. The study confirms overall the effectiveness of the usage of Virtual Reality in the research of pedestrian behaviour.

Tran et al. [28] realized a review in 2021 of virtual reality studies on autonomous vehicle-pedestrian interaction. The authors reviewed 31 studies made from 2010 to 2020. They performed a systematic analysis to identify the current coverage and assessed the evaluation measured. With their findings, they presented a set of recommendations for implementing VR pedestrian simulators and proposed directions for future research.

The review by Tran et al. identified study gaps in the current literature in relation to the following points:

Scalability : the majority of the current studies focused only on the interaction between a single pedestrian with a single vehicle, suggesting that new simulators should handle complex traffic situations, where multiple vehicles and pedestrians cross paths.

Mixed traffic : mixed traffic consisting of vehicles with different levels of automation.

Environmental conditions : the majority of studies have been conducted during daytime hours. Other conditions of weather and time should be tested in scenarios where vehicle movements are difficult to observe.

Vehicle behaviour in VR : more studies should consider pedestrian behaviour and have the vehicles adapt to the environment by making use of, e.g. ray casting.

Tran et al. also suggest that vehicles should simulate human-like driving behaviour, develop traffic with appropriate complexity by including more moving traffic, studies should consider familiar elements and traffic cultures which may have differences where an unsignalized crosswalk can be a safe or unsafe situation, create a social atmosphere and utilize of noise to increase the user's presence. He also summarized the diverse factors which influences on pedestrian behaviour within a simulator, as shown in figure 3.1.

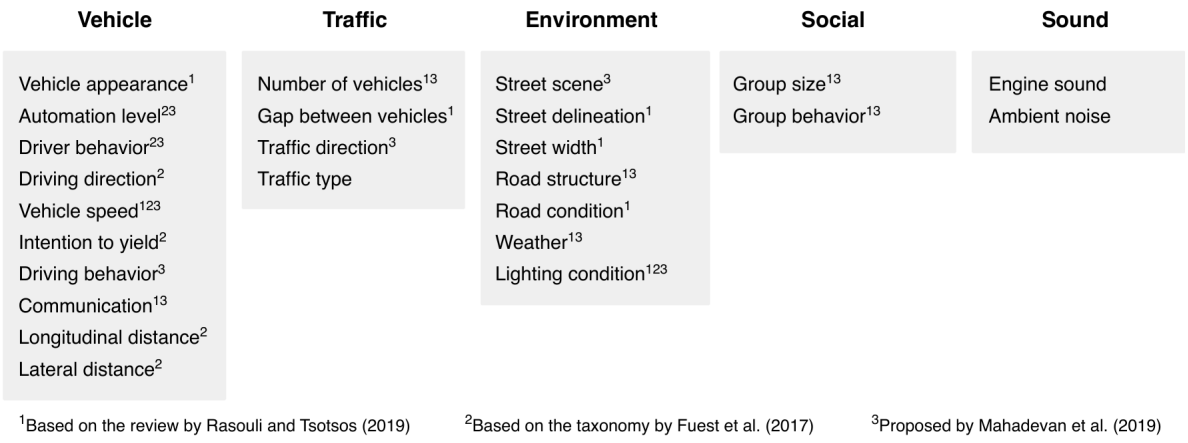


Figure 3.1: Summary of the different factors influencing pedestrian experience and behavior. Source: Tran et al. [28]

Angulo et al. [3] made a study in 2023 using a virtual reality simulation as a tool for understanding and evaluating pedestrian safety and perception at midblock crossing. In their work, they compared the real-world data of a midblock crossing and a replica made with a VR simulator. On their simulator, the vehicles traveled at a constant 40 km/h with spawns based on the real-world arrival distribution, acquired through video data. They used the gap between two vehicles to verify what was considered an acceptable distance between two vehicles for a safe crossing.

The work of Angulo et al. concluded that the gap acceptance distribution between the real world and the VR simulator was statistically similar. Similar to the work of Deb et al. [7], they also found the crossing speeds to match those of the real world. Angulo et al. survey results also reported high levels of immersion and realism. Those results indicate that the users of the VR simulator made consistent and realistic decisions while crossing the street.

4 PROPOSAL

In this chapter, we will explore how the simulator we made work and behave, the considerations and decisions we had to make, and how we chose to log the data from a crossing. In section **4.1 - Environment**, we will explore the virtual world created to immerse the user. While on section **4.2 - Running Experiments**, we will explore how researchers can set up the simulator to run a batch of tests on examinees to study their behaviours in different situations, adapting the simulator to their needs. Following this, in the section **4.3 - Car Simulation**, we will explore how we increased the complexity and realism of the traffic on the simulator, and how the vehicles behave in the world. At the end, in section **4.4 - Logging**, we will explore how we made the data of a crossing available for researchers to work on, and the implementation of a replay system to the simulator.

4.1 ENVIRONMENT

In this section we will define and explain how the street environment is created, discussing the models utilized and performance considerations (4.1.1); what is the crossing goal of the pedestrian and where the crossing start (4.1.2); which scenes we made available (4.1.3); what type of crosswalk we used in our simulator (4.1.4); the multiple vehicle models and their different speed profiles (4.1.5); how the vehicles are instantiated on the street (4.1.6); how the player can move to cross the street (4.1.7), and the audio created to increase the user immersion (4.1.8).

4.1.1 World

An important aspect of every VR simulator is to create a realistic world for the users to feel immersed in. The street should look and feel real. For this, the aspect rates of the objects are quite important, since the view height is the same as the one the user is used to in the real world. It's also important that some elements that we are used to seeing in the real world be present, even if they feel not very important at first, like power cables.

The art style of the models should also be consistent, so some objects don't seem out of place. To achieve the points listed, we had to choose carefully which asset pack to use. While being realistic is important, we need to remember that our target hardware is still a mobile device, so it's not powerful enough for ultra-realistic graphics at a high resolution, and decreasing the resolution may cause discomfort and motion sickness for some users. For that reason, we should have a simple but consistent art style, with a modular design that allowed us to place the environment in the way we wanted. The width of a street lane in the model we chose is, approximately, 4.5 meters. Figure 4.1 gives a quick look at the street environment we created.

Another important aspect of the world is its illumination. While dynamic light sources that generate dynamic shadows and reflexes look good, it is quite performance-intensive. So instead of using dynamic light sources, all the lights and shadows of our world are baked into the scene models. This causes the unfortunate fact that the vehicles don't cast shadows and their headlights also don't illuminate the street on the night scene. This was a trade-off that we had to make to allow the game to run at the target frame rate of the Quest 2 at native resolution, instead of being dependent on a powerful computer, which would increase significantly the cost to run our simulator.

Figure 4.1: A view at the street level of the one way straight street taken from the unity editor.

4.1.2 Goal and Start Positions

In the context of simulating pedestrian behavior during street crossings, our approach involves creating a designated endpoint for the pedestrian journey rather than merely reaching the opposite side of the road, since we usually have a goal location when crossing a street. To achieve this, we selected a recognizable and compact destination—a bus station. The choice of a bus station not only ensures ease of identification for the player but also allows for it to be moved through the virtual scene, thanks to its compact size. The crossing is considered successfully completed when the player reaches a predefined location within the bus station.

To facilitate the goal detection, we implemented a box detector surrounding the bus station, shown in Figure 4.2. When the player enters this goal box, an event is triggered, signaling the successful achievement of the crossing goal. The dimensions of the goal box are set at 3 meters in width, 3 meters in height, and 4 meters in length.

Since we wanted to allow for experiments to be run at room scale, with the players walking in real life to move through the virtual world, we chose to put the default position of the goal straight ahead of the start position of the pedestrian. In scenes that have a raised crosswalk, they are also put close enough to both, the start position and the goal, that the player could possibly walk to in a limited space. However, as we discuss in the section 4.2, those values can be easily changed by the examiners. The default start position and goal of those scenes can be seen in Figure 4.3.

4.1.3 Scenes

To increase the variety of studies that can be done on our simulator, we designed four scenes to allow the studies of crossing at different conditions.

The first scene is a unidirectional straight street, identified by the name 'OneWayStraight-Street', on the middle of each we have a raised crosswalk (section 4.1.4). The second scene, identified by 'OneWayStraightStreetNight', is a copy of the first one but at night, to measure the

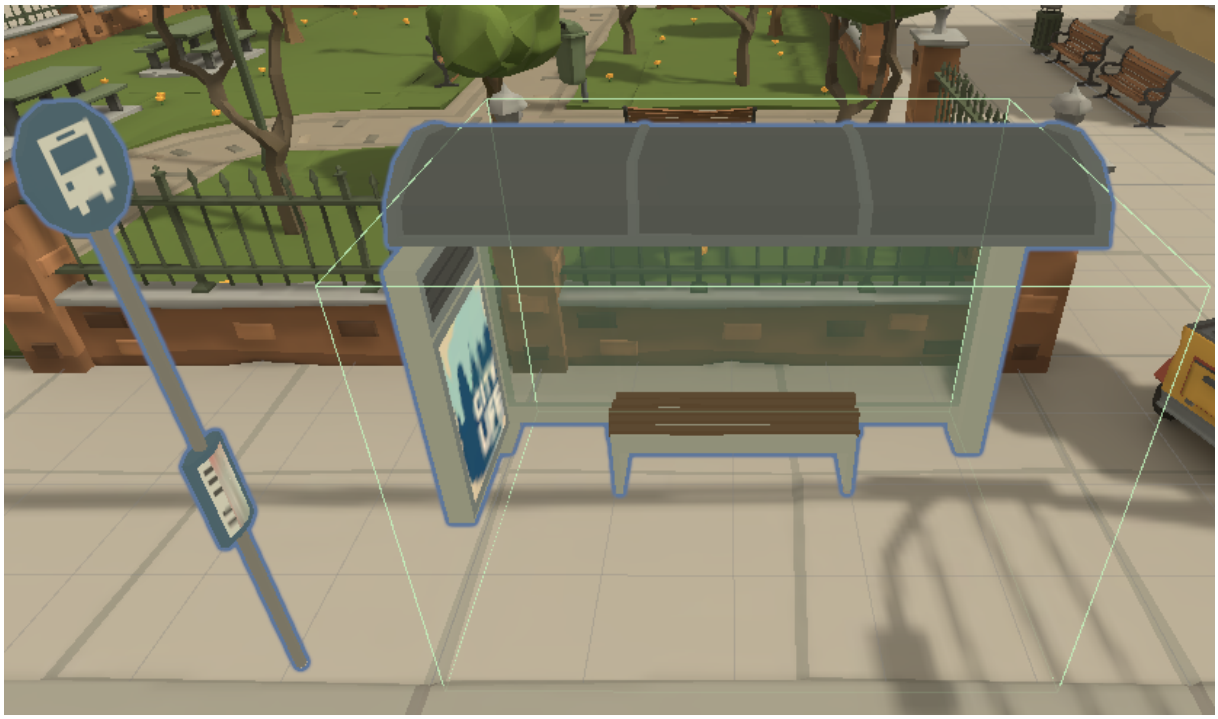


Figure 4.2: Bus stop model and goal detector box.

impact between different light conditions. In the third scene, identified by 'TwoWayStreet2', we have a bi-directional, with two lanes, straight street, so here the pedestrian needs to carefully look at the traffic coming from both sides, it also has a raised crosswalk in the middle. The last scene, identified as 'OneWayTurn' has a 90-degree turn, which, if the start position or goal is placed close to the turn, may limit the time of visibility of coming vehicles. The Figure 4.4 shows the turn and night scenes environment.

Vehicles were placed in advance in all scenes, so right at the beginning of a crossing there will be already vehicles passing through the player's location.

4.1.4 Raised Crosswalk

A common case of study is testing what makes a pedestrian walk to a crosswalk and cross with more safety, or cross directly where he is, getting to their objective faster, but increasing their risks. Unfortunately, in Brazil, the number of vehicles that wield to a crosswalk in certain regions is very low. A research made in the São Paulo city showed that only 28,3% of vehicles wielded to the pedestrian [12]. In Brasília, those numbers are better, but still far from ideal. A study made in Brazil's capital indicates a wielding rate of 57,48% [22].

Those numbers are generally better on a raised crosswalk since the vehicles need to slow down before crossing it. For this reason, and to also give some variety to the vehicle's speed profiles, we decided to make all the crosswalks of our simulator raised. When a vehicle comes close to a raised crosswalk, they will reduce their speeds to around 25km/h, to not hit the bottom of the vehicle on the street, accelerating right after going through the crosswalk.

Those crosswalks have a detector box on them, which detects when a pedestrian is waiting to cross. When a pedestrian is waiting, the coming cars will stop, wielding the right of way to the pedestrian, making them then able to safely cross to the other side.

Because of this, the pedestrian start position should not be placed directly ahead of them, but rather at a small distance, creating a case where the pedestrian needs to walk more, taking a

Figure 4.3: Default player start position (blue circle), goal and raised crosswalk.

Figure 4.4: Night scene on the left, 90 degree turn scene on the right.

little more time, to cross safely. This also gives a good study case, by placing the start position and goal closer or further from the crosswalk, we can study what is a distance that the majority of pedestrians would be willing to walk to it, giving important data about crosswalk placement.

4.1.5 Vehicle Types

As discussed in the study review made by Tran et al. [28], a missing aspect of the current studies is to have different models and colors of the vehicles, to study the impact of those elements on the crossing of pedestrians. So, to improve on those points, our simulator can spawn between four different vehicle models, and it has 12 different colors for each model.

Two of those vehicles' models, also have different behavior, one going up to 50% faster than the delimited street speed, and the other moving at a speed 25% slower than the one defined. This increases the variety of the vehicles, and increases the transit complexity. With this, we have the following vehicle types and models:

0 - Normal : its target speed is the same defined as the path nodes (section 4.3.1). The possible models are a compact vehicle or a SUV (Figure 4.5).

1 - Fast : its target speed is 50% faster than the one defined by the path node. The model of this vehicle is a muscle car (Figure 4.6).

2 - Slow : its target speed is 25% slower than the one defined by the path node. The model of this vehicle is a Van (Figure 4.6).

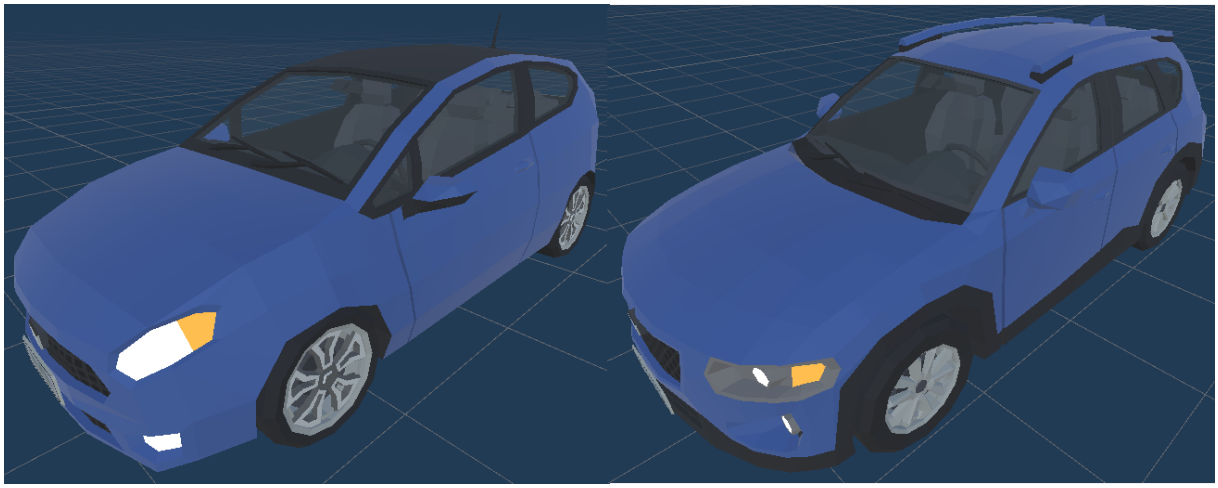


Figure 4.5: Compact car model on the left, SUV model on the right.

The approximate dimensions of the vehicles are the following:

Compact car : 1.76m wide, 1.4m high, 4.07m long.

SUV : 1.8m wide, 1.6m high, 4.6m long

Muscle car : 2m wide, 1.35m high, 5.3m long.

Van : 2.4m wide, 1.9m high, 4.85m long.

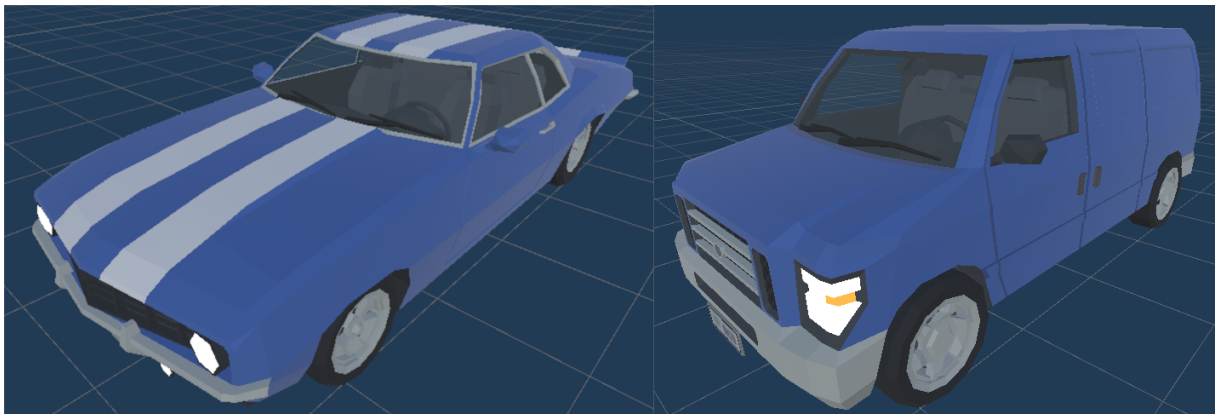


Figure 4.6: Muscle car model on the left, Van model on the right.

4.1.6 Vehicle Spawn

To introduce variability and enhance realism across scenes, vehicles are dynamically spawned at random intervals at the start of each lane, provided this space is unoccupied. Each lane operates independently, allowing for simultaneous spawns in different lanes. The default spawn interval ranges from 1 to 5 seconds.

To ensure consistent comparisons between different test instances within the same scene, a fixed seed is utilized for the random spawning algorithm. This deliberate choice aids in maintaining reproducibility and facilitating a reliable assessment of pedestrian behavior across different trials. Notably, while vehicle types and materials adhere to a predetermined set, the non-deterministic nature of the Unity physics engine may result in slight variations in spawn times, especially if a vehicle occupies a space for an extended period. This seed may also be changed when setting up a test environment, as detailed in Section 4.2.

In addition to temporal variability, the vehicle type is also subject to randomization. By default, there is a 10% chance of spawning a vehicle with a speed exceeding the default, and a separate 10% chance of spawning a slower vehicle. These default values offer a balanced mix of vehicle speeds, creating a dynamic traffic environment. As part of the experiment's adaptability, researchers can modify these probabilities, tailoring the traffic conditions to specific study requirements (Section 4.2).

4.1.7 Player Movement

To maximize the realism of our simulator, we designed it with the capability for users to navigate by physically moving in the real world. Leveraging the Quest 2's translation tracking, the virtual camera aligns seamlessly with the user's movements, ensuring a lifelike crossing experience.

With that said, a space of close to 10m is needed to do the crossing, which we know is not a space that all laboratories can afford to have. So, we also added support to moving through the Quest 2 analog controllers. In this mode, the user will move at a constant speed of 1.5m/s, which is close to the mean speed of a pedestrian while crossing an unidirectional street [5]. While the direction of the movement will follow the player's face.

4.1.8 Audio

An important factor for immersion is to make use of other human senses to give a better feeling of presence. With that in mind, we added a sound to the vehicle engine, which is added to each vehicle in the scene [17]. Those audio sources make use of Unity's built-in Doppler effect, giving

the sensation that the cars are coming close to the user and then passing him. This also helps the pedestrian to know that a vehicle is coming closer without having to look directly at it.

Those engine sounds were also slightly modified to each vehicle, by increasing or decreasing volume and pitch, giving each vehicle model a unique sound profile. The Van model also has a unique sound to represent its different motor characteristics.

4.2 CREATING EXPERIMENTS

In this section, we will explain how our simulator defines a sequence of experiments to be run, and how it behaves while running them.

4.2.1 Defining Experiments

In the context of conducting experiments with a group of participants, a standardized test routine, where an experiment can be run after another, is often essential. For that reason, and to allow the simulated environment to be customized according to the researchers' needs, the simulator is able to read a sequence of scenes, each containing parameters to modify the virtual environment. The definition of those scenes and parameters is done through a JSON file, called 'runner.json'.

This file contains a list of 'scenes' dictionaries, which have the following format and can contain specifications for the following items:

sceneName : a string containing one of the four scenes identifiers described in section 4.1.3.

maximumSpeed : an integer representing the target speed that the nodes will define.

goalPosition : a dictionary representing the center point of the goal on the world, it contains three floats described by the letters: x, y and z.

goalRotation : a dictionary representing the rotation of the goal, it contains three floats described by the letters: x, y and z.

playerPostion : a dictionary representing the center point of the pedestrian's spawn on the world, it contains three floats described by the letters: x, y and z.

playerRotation : a dictionary representing the rotation of the pedestrian's spawn, it contains three floats described by the letters: x, y and z.

spawnMin : a float representing the minimum time between vehicle's spawn (section 4.1.6)

spawnMax : a float representing the minimum time between vehicle's spawn (section 4.1.6)

fastVehicleSpawnChance : an integer between 0 and 100 representing the chance of a fast car being spawn (section 4.1.6).

slowVehicleSpawnChance : an integer between 0 and 100 representing the chance of a slow car being spawn (section 4.1.6).

randomSeedLeft : an integer representing the seed of the vehicle instantiator on the left lane.

randomSeedRight : an integer representing the seed of the vehicle instantiator on the right lane.

Below, you can see an example of a 'runner.json' file:

```
{
   "scenes": [
      {
         "sceneName": "OneWayStraightStreet",
         "maximumSpeed": 50,
         "goalPosition": {
            "x": 2.53, "y": 0, "z": 107.89
         },
         "goalRotation": {
            "x": 0, "y": 0, "z": 0
         },
         "playerPosition": {
            "x": -12.84, "y": -0.826, "z": 107.46
         },
         "playerRotation": {
            "x": 0, "y": 0, "z": 0
         },
         "spawnMin": 1.0,
         "spawnMax": 5.0
         "fastVehicleSpawnChance": 10,
         "slowVehicleSpawnChance": 10,
         "randomSeedLeft": 33,
         "randomSeedRight": 3
      },
      {...},
      ...]
}
```

4.2.2 Running Experiments

After defining the scenes sequence and their parameters, the simulator app can be opened the experiments can be executed. Right when starting the app, the simulator will read the first scene and its parameters and start it. Then, when a player finishes the crossing of this scene, be it by getting to the goal or by being run by a vehicle, a menu (Figure 4.7) will pop in front of him and the physics simulation will stop, while also logging the information about this run. This menu will show him the result of the crossing (success or crash), the time it took, and two options: go to the next scene, or watch the replay of his crossing (section 4.5).

Those steps will repeat until there are no more scenes defined on the runner file. At this point, after the end of the last crossing, a message will congratulate the user for ending the exam, and prompt a replay of the whole exam run, which would normally be run by a different person.

4.3 CAR SIMULATION

In order to get realistic human behavior to the environment described, it's important for the cars to behave naturally [28]. For that, the cars of this simulator are subject to the physics simulation present in the Unity engine and have a non-constant velocity. For accelerating or decelerating a vehicle, torque or brake forces are applied directly to the car's wheels. The vehicle is also subject to drag and friction. Different vehicle models also have different weights and wheel sizes, changing their acceleration and deceleration profiles.

This physics simulation is great for increasing the realism, but it adds to the complexity of our system, since now vehicles behave differently from one another, so they need to react to one another to not crash. Since they are also subject to friction, some braking and acceleration

Figure 4.7: Crossing success menu.

may cause variations to their traction, causing slight variations in their forward vectors, which if not handled would accumulate during the simulation and make some cars leave their delimited street path.

To handle all this, we created a system which handles the vehicles' control logic, consisting primarily of two elements. The first one is a path navigation approach that uses checkpoints to be able to keep the cars on a predefined path, while the second is a fuzzy logic system making use of Unity's ray cast for a car to be able to notice if it's going to hit another one and handle acceleration changes accordingly.

4.3.1 Path Navigation System

The path navigation system is a simple node-based path that the cars consume as checkpoints. The cars trace a straight vector to the node location, adjusting its wheels to turn towards the target as needed. The nodes also contain the vehicle's maximum speed information, which is the maximum speed the cars should be from the last node until this new one. When a car passes through a node, it consumes it and then starts seeing the maximum speed and position of the next one in route. The node path of the straight one way street can be seen in Figure 4.8.

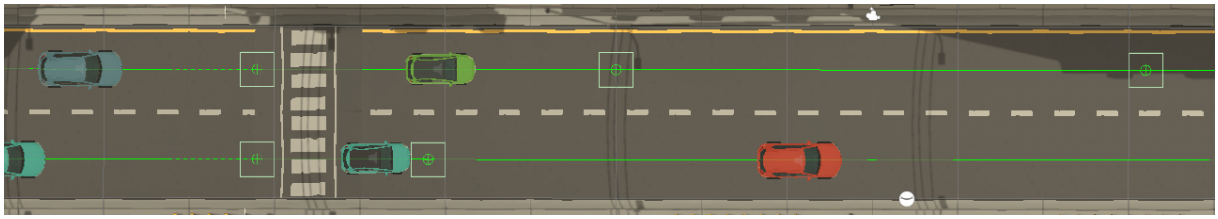


Figure 4.8: Nodes path. The line represents the full path the cars should follow, while the cubes are the nodes itself.

To be able to keep the vehicle in route and do turns as needed, we have to control the wheel's angle. Making also sure that small traction losses don't make the cars leave the desired path during a long street. For this, we create a vector from the vehicle position to the node and then use the normalized X point to multiply the maximum angle the car's wheel can turn. The maximum angle a wheel can turn was set to 30 degrees, since this is the maximum value for most passenger vehicles. A visualization of the steer vector is shown in Figure 4.9

$$turnAngle = 30 * \frac{steerVector.x}{|steerVector|} \tag{4.1}$$

where:

$turnAngle$ the turn angle to be applied to each front wheel.

$steerVector$ the vector between the car position and the target.

(4.2)

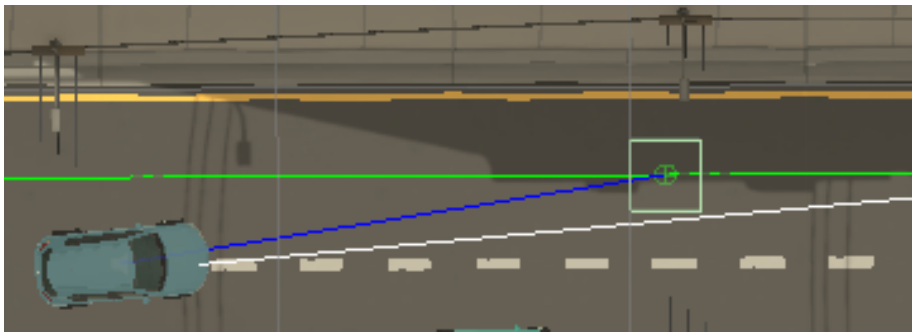


Figure 4.9: A car intentionally moved out of its path. The blue line represents the 'steerVector'. The white line is its ray cast, so the actual front vector in this frame.

4.3.2 Ray Cast System

While the path navigation system makes sure the vehicles stay on the road, it doesn't do much to keep one from hitting another. Since it has different target velocities, the node changes are actually a critical point, where a car can suddenly brake faster than the other, causing the car on the back to hit it.

This second system comes to fix that. The approach we used is inspired by the fuzzy logic described by Walotek, J. et al [31]. In his work, he described how to make an Artificial Intelligence (AI) system that controls a vehicle through a circuit, using fuzzy logic and seven-ray casts. In our case though, since our path is always fixed and thus controlled better with the node described in the Path Navigation System, we can reduce the number of ray casts to only one. This allows us to detect the distance between a car and the one in front of it, as well as the velocity and acceleration differences between those two. A visualization of the vehicles' ray cast can be seen in Figure 4.10.

Since our system is designed to work with a wide range of velocities, the distance of the ray cast is not fixed. While setting it to detect objects 3m ahead could be useful at 30km/h, it may be too late at 80km/h, so instead of trying to get a good value, we used as a basis the "secure distance" between two vehicles suggested by Brazil's defensive driving guide, which is of two seconds [4]. This means that the faster a vehicle moves, the bigger the distance it will try to keep from the in front of it, as shown in Figure 4.11

Figure 4.10: Vehicles with their "keep out" ray casts. White means that no vehicle is in range, red that they are too close from a vehicle, and green that they are seeing a crosswalk.

To calculate the distance that the car will move in 2s, we used a simplified physics model to calculate the rectilinear motion of the vehicle, with the distance of the ray being defined as the multiplication of the vehicle's velocity multiplied by the desired time (2s). This means that we are ignoring the acceleration of this calculation. This was done because adding it to the formula caused some unrealistic behavior of the vehicle changing between braking and accelerating too many times because it knew it would get too close in the future. The velocity of the vehicle is also not updated in every physics frame, but instead every 5 frames, this is done to ignore some extreme variations that may happen because of the interpolation of the vehicle in short frames.

Figure 4.11: The faster a car is moving, the bigger the ray it casts.

When a vehicle detects that there is another ahead of it, it uses the front vehicle speed as its target speed, instead of the usual path node-defined value. This change to maximum velocity

is enough to not hit the vehicle, and as the one ahead of it accelerates, the distance between the vehicles will naturally increase - since we don't take into account accelerations in this formula until the ray cast doesn't detect it anymore. Then, the vehicle goes back to its usual behaviour of being at a speed close to the maximum defined by the node, repeating this process along its path. The vehicles' speed control process is explained in more detail at the section 4.3.4

It's important to note here that the system's desired objective, is not to have a car that can drive perfectly in a very mechanical way. Instead, what we want in this simulation are cars that drive the closest to a human as possible. It's not common for drivers to give lots of small pressing of brakes and gas, but instead, they generally start pressing with a little force and increase until they get the desired effect.

The simplifications done in the distance calculations end up helping us to get the desired behaviour, since the moment a vehicle detects a vehicle ahead of it, it is a little too late to be able to keep a perfect 2s distance. So it will start braking slowly but will have to increase the force as it gets closer, giving us a behaviour similar to a human changing speeds.

4.3.3 Vehicle States

In order to control the vehicles' speed, we define the vehicles' move state as being in one of 5 states. This allows us to easily adjust which forces should be applied to the current vehicle in a frame. Those states are:

- **0 - Inertia**: No forces are applied by the controller, but the vehicle is still subject to external forces like drag and friction.
- **1 - Accelerating**: We are applying torque to the front wheels in order to increase the vehicle's speed.
- **2 - Breaking**: We are applying brake at all four wheels of the vehicle.
- **3 - Stopping**: We are breaking if the intent of zeroing the vehicle's speed.
- **4 - Stopped**: The vehicle speed can be considered as zero.

The state of a vehicle is defined by the following rules:

1. The target speed is equal[1] to 0 and the vehicle speed is equal[1] to 0: MoveState <- **Stopped**
2. The target speed is equal[1] to 0 and the vehicle speed is not equal[1] to 0: MoveState <- **Stopping**
3. The vehicle's speed is bigger than its target velocity plus the speed margin: MoveState <- **Breaking**
4. The vehicle's speed is bigger than its target velocity less the speed margin, but lesser than the target velocity plus the speed margin: MoveState <- **Inertia**
5. Otherwise: MoveState <- **Accelerating**

[1]Approximately equal, since we're handling with floating point values.

4.3.4 Controlling the vehicles speed

Now that we have defined the vehicle's target speed and its state, we need to be able to break at the correct time in order to not hit the one ahead of it. For that, we make use of the kinematic equations for linear motion, adjusting the forces as we get closer or farther away from the goal. The objective of this algorithm is to move the vehicle in such a way that it hits a *target speed* close to a *target position*. The target position of a vehicle is defined as the next path node, if there isn't any car in its ray cast (section 4.3.2), or otherwise as the front car's position.

A human driver only breaks quickly in emergencies, since it's uncomfortable for the passengers of the vehicle. Also, when a driver stops their car it's generally done by reducing the speed gradually to stop in the intended spot, e.g. close to the crosswalk line. In the same way, our autonomous vehicles should behave similarly. So we need to gradually increase or decrease the braking force until we stop at the desired spot. For this, we will first calculate the difference between the time to get to the spot with our current speed and acceleration, and the time to get to the desired speed. After this, we will update the breaking force accordingly to how close we are to getting the speed at the exact spot, that is, how close the difference between those two calculated times is to zero.

For the calculation of the time that we are expected to get to the target with our current velocity and acceleration, we make use of the following kinematic equation of motion:

$$d = v_i * t + \frac{1}{2} * a * t^2 \tag{4.3}$$

where:

d is the distance between the vehicle and its target.
v_i is the current vehicle's velocity.
t is the time to the target position.
a is the current vehicle's acceleration.

(4.4)

isolating the time t:

$$t = \frac{-v_i \pm \sqrt{v_i^2 + 2 * a * d}}{a} \tag{4.5}$$

since we are handling with time, we can only focus on the positive root, so the final form of our equation is:

$$t = \frac{-v_i + \sqrt{v_i^2 + 2 * a * d}}{a} \tag{4.6}$$

With the time to the target position in hand, we then calculate the time that it will take to get to the target velocity. For this, we use the kinematic equation below:

$$v_f = v_i + a * t \tag{4.7}$$

where:

v_f is the target velocity

v_i is the current vehicle's velocity.

a is the current vehicle's acceleration.

t is the time to the target velocity.

(4.8)

isolating the time t:

$$t = \frac{v_i - v_f}{-a} \tag{4.9}$$

Now that we have both the time to the target position and the time to the target velocity, we use the difference of those two as a factor to increase the torque, or break force, until the difference is close to 0. This is done using the equation below:

$$force < -max(min(force + (force/10) * timeDelta, baseForce * 4), 1) \tag{4.10}$$

where:

$force$ is the breaking force

$baseForce$ is a constant which define the minimum and maximum force applied

$timeDelta$ is the calculated time difference.

(4.11)

The max function in the above equation exists for when the vehicle stops before getting to the target position. In this case, the root of the equation 4.6 becomes negative and without a real solution. To circumvent this situation, when we do not get to the object we start reducing the break force by setting the time delta to -1. The maximum then guarantees that we will not start accelerating by putting a negative break force.

For the acceleration, we don't need to hit a speed at a specific spot, so here we just gradually increase our acceleration to a maximum until we get to the vehicle's desired speed. The equation for this is the following:

$$force < -min(force + baseForce/10, baseForce * 4) \tag{4.12}$$

Notice that it's intentionally very similar to the breaking one, with the exception of not using a calculated time delta. This is done because in both cases we are simulating the behaviour of a driver pressing the gas or brake pedals. With the difference that when accelerating he only wants to get to a determined speed to get to his destination as fast as possible. While when he is breaking he wants to stop at a determined spot our match another speed. A demonstration of the stopped vehicles is shown in Figure 4.12.

4.4 LOGGING

In order to study the behaviour of a pedestrian using the simulator, our system creates at the end of a scene two log files, containing the data about how the crossing happened. Those log files are separated into two types, a small JSON log file containing the results of the crossing, and a bigger one that is a replay of the frames created on the scene.

Figure 4.12: Vehicles stopped while waiting for a pedestrian to cross.

4.4.1 Results Log

This log file's main goal is to give a quick look at the results of a test scene. Here we output information about the duration of the crossing, whether it ended in a crash or not, the distance of the car that got the closest to the pedestrian, all the positions and state of the vehicles at the end, and the position of the pedestrian on the world.

The resulting json file has the following format and content:

date : a string containing the date of when this scene run ended, with the following format: yyyyMMddThh_mm_ss

scene : a string containing a descriptor of the scene asset name. Its value can be: "OneWayStraightStreet", "OneWayStraightStreetNight", "TwoWayStreet2", "OneWayTurn".

replay : a string containing the name of the replay file. (section 4.5)

endTime : a float which shows the time in seconds that the test run ended, starting from 0.

hasCrashed : a boolean indicating if the scene ended with a crash or not.

closestCarDistance : the distance, in meters, between the vehicle who got the closest to the pedestrian and the pedestrian.

cars : a list containing dictionaries with the information at the final moment about all the vehicles who were active when the scene ended, the information is given in the following format:

- **id** : an integer representing an unique identifier for this vehicle on this test run.
- **carType** : an integer representing the enumerator of the type of this vehicle. 0 being a normal vehicle, 1 being a faster vehicles, and 2 being a slower vehicle.
- **moveState** : an integer representing the enumerator of the final state of the vehicle (section 4.3.3).
- **speed** : a float representing the vehicle speed.
- **acceleration** : a float representing the vehicle acceleration.

motorTorque : a float representing the torque being applied to the vehicle's wheels.

breakForce : a float representing the force being applied on the vehicle's wheels in order to reduce its speed.

position : a dictionary representing the center point of the vehicle on the world, it contains three floats described by the letters: x, y and z.

player : a dictionary containing information about the pedestrian's position and to where its looking. The format is the following:

position : a dictionary representing the center point of the pedestrian on the world, it contains three floats described by the letters: x, y and z.

rotation : a dictionary representing the camera rotation of the pedestrian, that is, where the pedestrian is facing. It contains three floats described by the letters: x, y and z.

Below you can see an example of a log outputted by the simulator.


```
{
   "date":"2023-11-25T17_59_13",
   "scene":"OneWayStraightStreetNight",
   "replay":"replay_log_2023-11-25T17_59_13.json",
   "endTime":3.425032138824463,
   "hasCrashed":true,
   "closestCarDistance":0.0,
   "cars": [
      {"id":99,
      "carPrefabId":1,
      "carMaterialId":0,
      "carType":0,
      "moveState":2,
      "speed":7.650308609008789,
      "acceleration":-2.5024702548980715,
      "motorTorque":0.0,
      "breakForce":278.7091369628906,
      "position":{"x":-2.449937105178833,"y":0.003030717372894287,
         "z":-13.877348899841309}
      },
      {...},
      ...],
   "player":{
         "position":{"x":-7.763314247131348,"y":-5.960464477539063e-8,
            "z":101.0302963256836},
         "rotation":{"x":0.0,"y":0.7071068286895752,"z":0.0}}
}
```


4.5 REPLAY SYSTEM

While the log described in the last section is useful for getting a quick look at what caused the end of the crossing, it lacks further details and metrics that may be needed for some researchers, like vehicles' gap sizes, in seconds, and crossing speed [3], or gap between vehicle and participants, in meters [7]. Unfortunately, there is currently no standard on the metrics used to study the behaviour of pedestrians [28]. So, since our main goal is to provide a platform that allows researches to study pedestrian behaviour as they want, instead of trying to predict the metrics that the researches will need or want to use, we opted for giving all the information about the positions

of the vehicles and the pedestrian along the time of the crossing. For this, we implemented a replay system in our simulator.

A replay system allows both researchers and examinees to analyze a crossing in detail. But first, let's discuss what is a replay system and the two approaches we could take to implement it.

A replay system is primarily a system that records a scene sequence and then allows it to be replayed similarly to recording a video and then playing it later on [9]. Though here, we use the game engine to replay it. This allows the scene to be seen in various angles and positions.

To implement a replay system in unity, we can follow one of the two main approaches: Input-Based or State-based [9].

In an input-based system, we capture the initial state of the objects and then the inputs by the players [9], in our case that would mean recording only the player position, and then the world would react to those inputs in the same way it did at first, relieving then the scene that the user experienced. This approach has a low memory footprint, it has some problems though. Because we are only recording the player input, this approach needs to have a deterministic game or engine, for the world to have the exactly same behaviour every time [9]. Our simulator makes use of the Unity default physics system, which is, unfortunately not deterministic, and the behaviour of the vehicles is also frame-time dependent in our world. That is, if a frame ends up being late, vehicles can start breaking a little later than in the last run, which will cause differences that accumulate over time. This approach also has another significant drawback for us, it wouldn't allow the data we are recording to be analyzed outside of the simulator, in bulk.

So the approach we implemented for our simulator is a State-based system. Here, we record the position and rotation of all moving objects on the scene along the time and then replay it in the exactly same order. This approach has the disadvantage of using more memory, though this can be reduced by storing the data at a lower frame rate than the one played. [9], but in our case, having all the positions being recorded is a big advantage. By storing all the vehicles' positions and the pedestrian's position along the time, a researcher can plot those points the way he wants, being able to see them in another software, possibly reducing it to only two dimensions for ease of view. This also allows any metric to be calculated, since we are outputting all the movements that occurred during the crossing. If a researcher wants to, e.g. calculate the gap size between the vehicles, he can use the kinematic equation to calculate the time between the vehicles, using the current values for their acceleration and speed.

The replay system we implemented, records a frame at each 0.05s (20 frames/second), to reduce memory usage. Considering this frame time, a vehicle moving at the maximum default speed of 50km/h, may move at most 70cm between each frame.

In order to store the data to be analyzed by researchers, we use a simple JSON file. This JSON file consists of two lists, one containing the frame information, and the other information needed to recreate the scene at the engine, like the prefab and material the vehicle uses. The format of each frame element is the following:

player : a dictionary containing the player position and rotation, in the following way:

- **position** : a dictionary representing the center point of the pedestrian on the world, it contains three floats described by the letters: x, y and z.
- **rotation** : a dictionary representing the camera rotation of the pedestrian, that is, where the pedestrian is facing. It contains four floats described by the letters: x, y, z and w.

cars : a list containing dictionary with information about the vehicles active on the current frame. Each car entry has the following format:

position : a dictionary representing the center point of the vehicle on the world, it contains three floats described by the letters: x, y and z.

rotation : a dictionary representing the rotation of the vehicle. It contains four floats described by the letters: x, y, z and w.

id : an integer representing an unique identifier for this vehicle on this test run.

speed : a float representing the vehicle current speed.

acceleration : a float representing the vehicle current acceleration.

frameDuration : A float describing how long is the time of the between the end of the last recorded frame and the start of the recording of this one.

The list containing the information to recreate the scene at the scene has the following format:

id : an integer representing an unique identifier of a vehicle on this test run.

details : a dictionary containing the information needed to instantiate the vehicle on the simulator. It contains the following information:

carPrefabId : an integer that points that a vehicle model prefab. If this is a vehicle with a default speed profile, then 0 represents a SUV, and 1 represents the small vehicle.

carMaterialId : an integer that point to a vehicle material, which is used to give their colors variance.

carType : an integer representing the enumerator of the type of this vehicle. 0 being a normal vehicle, 1 being a faster vehicles, and 2 being a slower vehicle.

Below, you can see an example of a replay file outputted by the simulator:


```
{
   "frames":[
      {"player":
         {"position":{"x":-12.84000015258789,"y":-5.960464477539063e-8,
            "z":107.45999908447266},
         "rotation":{"x":-0.1271669864654541,"y":0.695626974105835,
            "z":-0.016566216945648195,"w":0.7068644762039185}},
      "cars":[
         {
            "position":{"x":-2.3114371299743654,"y":-0.005701422691345215,
               "z":31.846420288085939},
            "rotation":{"x":-0.00003174791345372796,"y":0.0017363657243549 8,
               "z":-0.00001980801607714966,"w":0.9999985694885254},
            "id":86,
            "speed":0.028882600367069246,
            "acceleration":0.28882601857185366
         },
         {...},
         ...],
      "frameDuration":0.053286951035261155
      },
      {"player": {...}, "cars": [...], "frameDuration":0.05541764944791794},
      ...],
   "info":[
```

```
25          {"id":86,"details":{"carPrefabId":0,"carMaterialId":0,"carType":1}},
26          {"id":87,"details":{...}},
27          {...},
28          ...]
29  }
```

To replay the scene on the simulator's engine, we go through each frame of the JSON file, instantiating any new vehicle or removing the older ones. Then, to make the viewing experience of the replay on the simulator less prone to motion sickness caused by the low frames per second and to make the viewing experience more enjoyable, we interpolate the movement of the objects during the duration of the frame. This way, the vehicles move at the usual target frames per second of our simulator, instead of being limited by the replay file frames, making it seem like they're moving by the physics simulation used during the actual crossing. The player's movement is represented through a simple human model, that moves and turns accordingly to the stored information.

5 RESULTS

In the last chapter, we presented the simulator we developed as a platform to allow further studies on the behaviour of pedestrians. Although the main function of the simulator is the study of psychological and behavioural patterns, our focus in this work was the development of the platform, with the main result being the platform developed and presented in the last chapter. With that in mind, experiments to gather and analyze those behavioural patterns were not done at this work, being a future work topic.

In this chapter, we will then be validating the functionalities described in Chapter 4, making sure they work as intended and as described.

5.1 VALIDATION

5.1.1 Experiments runner

An important aspect of the simulator is its high customization and ease of setting up experiments. To validate those aspects, we modified the value of a custom "runner.json" file, and ran the simulator on the Unity editor, to validate if the described parameters were being set as described in the file.

After making sure the values were correctly changed, and the positions of objects also moved in the world correctly, we created a sequence of eight scenes and ran it on the Quest 2 two times, validating that the scene order was the same as the one defined and that the experiment sequence was able to run without any hiccups.

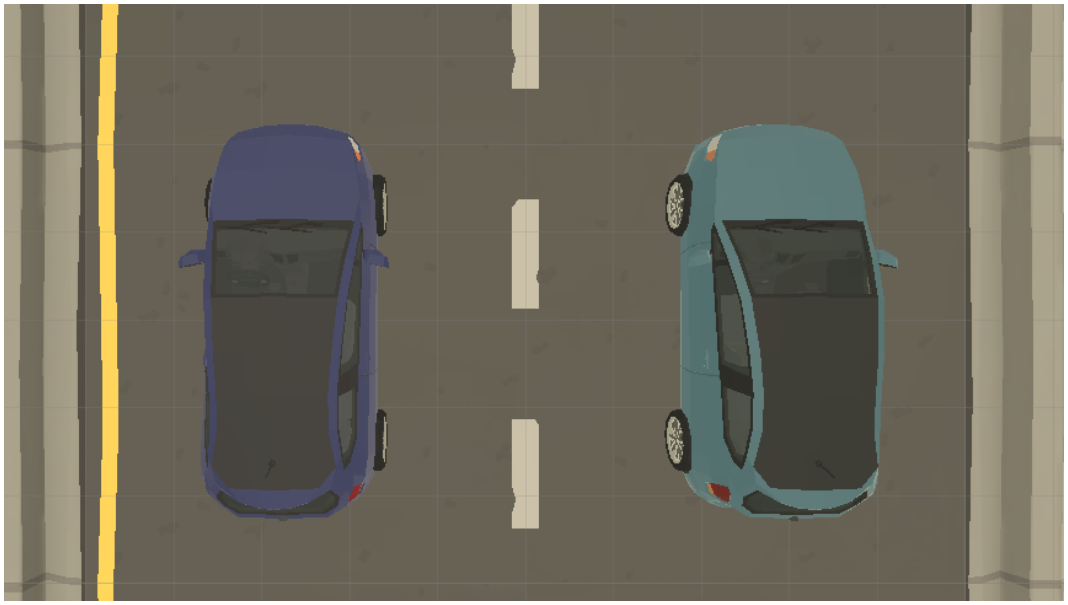


Figure 5.1: Vehicles instants after beings spawned, with spawnMin and spawnMax set to 20s.

By setting the spawn rates of the fast vehicles and slow to 100% and 50%, we could verify that only those vehicles' types were being spawned, validating the parameter change, and the spawn of the vehicles. Changing the spawn time also gave the correct result, setting the minimum and maximum value to 20 seconds, we were able to verify that the vehicles of both lanes spawned at the exactly same time, with 20s between spawns (Figure 5.1).

Altering the maximumSpeed also works as intended, with a speed of 0km/h making the vehicles not move.

5.1.2 End States

A scene in our simulator can end in two ways, getting to the goal position, or being hit by a car.

To validate the first one, we walked until the goal, verified visually that the next scene menu did open, and also verified the log states, which listed correctly that we didn't get hit, the closest car distance was 1.92m, and that our end position is on the goal box defined by the runner. With the goal center position defined as x: 2.53 and z: 107.89, the logged position of the player at the end of the scene was x: 0.89 and z: 107.46, which is within the goal boundaries (section 4.1.2). By running the replay we also can see that the pedestrian model is within the goal.

Figure 5.2: The end of a replay were the pedestrian was hit by a vehicle.

For the second one, we did a similar thing but now intentionally being hit by a car (Figure 5.2). We could validate then that the menu did open telling that we were hit, the log "hasCrashed" variable was true, and that the closest car distance was 0. By looking at the player position of -7.36 and 107.46 and the car positions, we can get that the car that hit us was the one with identifier 20 and the compact model (section 4.1.5), and as expected hit us through the front (z-axis), having a final position of -7.51 and 105.13. When we calculate the boundaries, we can see that the car boundary extends to 107.18, while the player boundary goes to 107.26, making the log position only 8cm off the real hit.

5.1.3 Vehicle AI

The vehicle AI we created is composed of two main components, its path navigation system, and the ray cast velocity control.

The first system makes the car target a certain speed, which we could verify was working as intended by looking at the vehicles' speed and state when going through a node. With this, we could see that it did indeed increase its speed, by being on the "Accelerating" state, when its speed was at least 5km/h lower than the target, keeping the speed when within the margins, and decreasing when a lower speed node was in front of it. With relation to targeting the position, we validated it by moving it out of its path, as seen in figure 4.9.

Finally, for the validation of the second system, we could first validate that it aims to keep the 2s out zone by having the ray cast with a red color when close to another vehicle, with their distance increasing with time until the ray is green again. We could also quantitatively test its breaking precision, by having a pedestrian waiting, we logged the vehicle's target stop point at the point where it actually stopped. The positions x and z of the target were -7.5, 87.83, and the vehicle stopped at the position: -7.49, 87.34, giving a distance of 49cm. We repeated this test 10 times to consider different vehicle types and small changes in their speeds, getting an average distance of 75cm between the target and the point we stopped.

6 CONCLUSION

This chapter concludes our study by summarizing and analyzing what we achieved. It also looks at the possible applications of the work here developed. Finally, it will present future researches that may be done to expand the work on pedestrian behaviour on the streets.

6.1 SUMMARY

In this work, we explored the current and past state of VR technology, looking at their strengths and limitations. With this information as a basis, we developed a platform to study and analyze pedestrian's behaviour while crossing a street. After looking at past studies, we analyzed the limitations of the pedestrian simulators developed in the past. Taking those into mind, we developed a simulator that focused improved on those limitations, in order to study the behaviour in multiple scenarios, with high fidelity.

The end result was a highly customizable simulator, which allows different researchers to study specific topics as they need, by modifying the parameters of the scenes. The simulator is able to run a suite of experiments defined by a researcher with no intervention, all while allowing the real-time monitoring of those by making use of the Quest 2 built-in stream functionality. The scenario was made diverse by the use of multiple vehicles models, with different colors and randomization. The simulator, makes use of current design techniques, like light baking, to improve the fidelity of the virtual environment without the need for a more powerful and expensive device. The traffic created by the vehicles' AI allows a complex world to be analyzed, all while having human-like behaviour. By making use of a replay system, we were also able to cater to all past and future metrics which may be used to analyze pedestrian behaviour, while still giving a quick output for those who need simpler metrics.

With all this in mind, we conclude that the simulator developed in this work was able to meet all of our original goals, with the prospect of being a powerful device for the study of pedestrian behaviour.

6.2 POSSIBLE APPLICATIONS

A clear application, which was the focus of this work, is the use of the simulator here developed on diverse experiments with the focus on analyzing pedestrian behaviour on the streets. With the simulator here developed, new scenarios which were not possible to be studied before without extensive development work, by making use of this platform, researchers can focus more on the behaviour and less on the implementation of a new, complex system.

6.3 FUTURE WORKS

The topic of developing a virtual reality simulator to study the behaviour of pedestrians is something that we expect will only keep getting more and more relevant with the advance of automated vehicles. For instance, future researchers may want to test how humans will behave in a complex traffic environment composed of vehicles that use AV techniques like machine learning to drive, without focusing on simulated human driver behaviour.

Another important area of study is making the simulator with the focus, not on data collection and analysis, but instead, as a teaching research, studying the impact of those on their behaviours at the transit.

More study can also be done on improving the capability of the vehicles, having them able to do more scenarios, like changing lanes, overtaking other cars, avoiding obstacles, and reducing the speed for a pedestrian that started the crossing to safely finish when it's possible to react on time.

Another point of study is improving the interaction between the human pedestrian and the automated vehicles present on the simulator. Khartik et al. had some good results in their work by giving visual cues to pedestrians about when the vehicle acknowledged and the pedestrian intended to wield [18]. Though, carefulness must be taken in this area to not have vehicles that can detect pedestrians in situations where it would not be true for a real driver, teaching the wrong behaviour to the users.

With the advance of VR technology, we also expect simulators with higher visual fidelity to be developed, improving further the sense of presence of the user in the virtual world. Finally, while we tried to make our simulator as accessible as possible, we understand that for some researchers the cost of a device like the Quest 2 may still be too expensive at this time. As such, research can be done on adapting the simulator here developed to be able to run on smartphones, using something like the Google cardboard platform, or even transforming it into an AR device using the smartphone cameras and sensors.

REFERENCES

[1] (2015). Transforming our world: the 2030 agenda for sustainable development. Technical report, United Nations.

[2] (2018). Global status report on road safety 2018. Technical Report ISBN 978-92-4-156568-4, World Health Organization, Geneva.

[3] Angulo, A. V., Robartes, E., Guo, X., Chen, T. D., Heydarian, A., and Smith, B. L. (2023). Demonstration of virtual reality simulation as a tool for understanding and evaluating pedestrian safety and perception at midblock crossings. *Transportation Research Interdisciplinary Perspectives*, 20:100844.

[4] Chagas, F. C. (205). Direção defensiva - Trânsito seguro é um direito de todos. Apostila.

[5] Chandra, S. and Bharti, A. K. (2013). Speed distribution curves for pedestrians during walking and crossing. *Procedia - Social and Behavioral Sciences*, 104:660–667. 2nd Conference of Transportation Research Group of India (2nd CTRG).

[6] Coulter, R., Saland, L., Caudell, T., Goldsmith, T. E., and Alverson, D. (2007). The effect of degree of immersion upon learning performance in virtual reality simulations for medical education. *Medicine Meets Virtual Reality*, 15:155.

[7] Deb, S., Carruth, D. W., Sween, R., Strawderman, L., and Garrison, T. M. (2017). Efficacy of virtual reality in pedestrian safety research. *Applied Ergonomics*, 65:449–460.

[8] Dillet, R. (2018). Unity ceo says half of all games are built on unity. `https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/`. Publised by TechCrunch. Accessed on 2023-11-18.

[9] Engel, T. (2020). Creating a replay system in unity. `https://www.kodeco.com/7728186-creating-a-replay-system-in-unity/`. Publised by Kodeco. Accessed on 2023-11-18. Archived at https://web.archive.org/web/20231110080308/https://www.kodeco.com/7728186-creating-a-replay-system-in-unity.

[10] Epic Games (2023). Unreal Engine Marketplace. `https://www.unrealengine.com/marketplace/en-US/assets`. Accessed on 2023-11-18. Archived at https://web.archive.org/web/20231115205255/https://www.unrealengine.com/marketplace/en-US/assets.

[11] Fazzolari, R. and Jaber, A. A., editors (2021). *Proceedings of the International Conference of Yearly Reports on Informatics, Mathematics and Engineering*, number 3118 in CEUR Workshop Proceedings, Aachen.

[12] Freire, L. R. (2021). Respeito à travessia do pedestre na cidade de são paulo: quantos e quem são os que a estão respeitando? In *Revista UniCET*.

[13] Google (2023). Google Cardboard. https://arvr.google.com/cardboard/. Accessed on 2023-11-02. Archived at https://web.archive.org/web/20231031215145/https://arvr.google.com/cardboard/.

[14] Heath, A. (2023). This is Meta's AR / VR hardware roadmap through 2027. https://www.theverge.com/2023/2/28/23619730/meta-vr-oculus-ar-glasses-smartwatch-plans. Accessed on 2023-11-02. Archived at https://web.archive.org/web/20231020031043/https://www.theverge.com/2023/2/28/23619730/meta-vr-oculus-ar-glasses-smartwatch-plans.

[15] Interaction Design Foundation - IxDF (2022). What is virtuality continuum? https://www.interaction-design.org/literature/topics/virtuality-continuum. Accessed on 2023-10-17. Archived at https://web.archive.org/web/20231017000000/https://www.interaction-design.org/literature/topics/virtuality-continuum.

[16] Lim, C. H. and Lee, S. C. (2023). The effects of degrees of freedom and field of view on motion sickness in a virtual reality context. *International Journal of Human–Computer Interaction*, 0(0):1–13.

[17] Limbasiya, H. (2018). Sense simulation in virtual reality to increase: Immersion, presence, and interactions. *Master, University of Dublin, Dublin, Ireland*.

[18] Mahadevan, K., Sanoubari, E., Somanath, S., Young, J., and Sharlin, E. (2019). Av-pedestrian interaction design using a pedestrian mixed traffic simulator. pages 475–486.

[19] Maruhn, P., Dietrich, A., Prasch, L., and Schneider, S. (2020). Analyzing pedestrian behavior in augmented reality — proof of concept. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 313–321, Atlanta, GA, USA.

[20] Meta (2023). Meta Quest 2 and its motion controllers. https://www.meta.com/quest/products/quest-2/. Accessed on 2023-11-02.

[21] Milgram, P. and Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE Transactions on Information Systems*, E77-D(12).

[22] ONG Rodas da Paz and ONG Andar a Pé and Coletivo MOB (Movimente e Ocupe o seu Bairro) (2018). Pesquisa de respeito à faixa de pedestre no df - relatório geral. Relatório.

[23] PATTI, A. (2020). Reducing motion sickness on mobile phone based vr applications using 6dof tracking.

[24] SteamDB (2023). Steam Game Release Summary. https://steamdb.info/stats/releases/. Accessed on 2023-11-18.

[25] Sutherland, I. E. (1968). A head-mounted three dimensional display. In *Fall Joint Computer Conference*, pages 757–764.

[26] Tassi, P. (2018). 'Pokémon GO' Is More Popular Than It's Been At Any Point Since Launch In 2016. https://www.forbes.com/sites/insertcoin/2018/06/27/pokemon-go-is-more-popular-than-its-been-at-any-

point-since-launch-in-2016/?sh=61080639cfd2. Accessed on 2023-10-17. Archived at https://web.archive.org/web/20231017000000/https://www.forbes.com/sites/insertcoin/2018/06/27/pokemon-go-is-more-popular-than-its-been-at-any-point-since-launch-in-2016/?sh=61080639cfd2.

[27] The Godot Engine community (2023). Godot Asset Library. https://godotengine.org/asset-library/asset. Accessed on 2023-11-18. Archived at https://web.archive.org/web/20231117105331/https://godotengine.org/asset-library/asset.

[28] Tran, T. T. M., Parker, C., and Tomitsch, M. (2021). A review of virtual reality studies on autonomous vehicle-pedestrian interaction. *Transportation Research Interdisciplinary Perspectives*, 51(6):641–652.

[29] Unity Technologies (2023). Unity Asset Store. https://assetstore.unity.com/. Accessed on 2023-11-18. Archived at https://web.archive.org/web/20231117021436/https://assetstore.unity.com/.

[30] Valve Corporation (2023). SteamVR™ Tracking. https://partner.steamgames.com/vrlicensing. Accessed on 2023-11-02. Archived at https://web.archive.org/web/20231009105355/https://partner.steamgames.com/vrlicensing.

[31] Walotek, J., Oleksiak, J., Cebula, P., Stanek, A., and Szczypinski, M. (2021). A fuzzy logic based autonomous car simulation in unity. In [11], pages 77–84.

[32] Wise, J., Hopkin, V., and Garland, D. (2010). *Handbook of Aviation Human Factors*. Human factors in transportation. CRC Press.