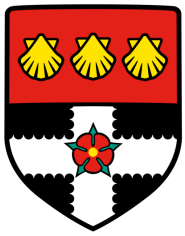

University of Reading

Department of Computer Science

Survival Game in Virtual Reality

Student: Giorgos Philippou

Supervisor: Dr Ashraf Mahmud

A report submitted in partial fulfilment of the requirements of
the University of Reading for the degree of
Bachelor of Science in *Computer Science*

April 22, 2022

Declaration

I, Giorgos Philippou, of the Department of Computer Science, University of Reading, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of UoR and public with interest in teaching, learning and research.

Giorgos Philippou
April 22, 2022

Abstract

Over the last years, the population has been developed rapidly in computational matters for research, health, entertainment, and many more reasons. Virtual Reality is one of the most exciting technologies that has been lately developed, and it still keeps developers' attention. How can you" live" in a world combining reality with imagination? What Virtual Reality and what can game development offer?

This project represents a 3D indie game in Virtual Reality, implemented in Unity. Working on such a project can intensify essential skills for a programmer, like creativity and algorithmic thinking. The coding part helps to get into more detail on a chosen language and even learn how to use a valuable and powerful engine. Combining game development with additional aspects can unlock different doors for the future. In this project, basic features of Artificial Intelligence are included.

The implementation of the game follows an Agile development methodology, as the reevaluation of the development is required to optimize the program's performance and cover the essential aspects of game development. When using VR, some techniques must be considered to compute a good result; an example could be to avoid motion sickness. Synchronisation, user experience, and interactivity are some of the most critical aspects that must be analyzed before creating a game. Focusing on the details forces developers to enhance their skills and find answers to any problem. Game development might be a complicated and time-consuming procedure, but it turned out to be a key to unlocking many of those skills.

Keywords: Virtual Reality, Game Development, C#, Unity, Creativity

Acknowledgements

Throughout the development of this project I have received a great deal of support and assistance.

I would like to acknowledge my supervisor Ashraf Mahmud, and Dr Varun Ojha for helping me out with the procedure to find the facility to start my project and more generally for providing some comments regarding the FYP. Also, I want to thank the department of Computer Science from University of Reading for providing me the essential hardware (Oculus rift), for the implementation of the Virtual Reality game.

Contents

List of Figures

List of Tables

List of Abbreviations

SMPCS	School of Mathematical, Physical and Computational Sciences

Chapter 1

Introduction

1.1 Background

Since 1958, when the first video game was introduced to the public (Chodos, A., 2019), gaming started to take an important role in people's life (Anon, 2012). This project focuses on developing a 3D survival game in Virtual Reality using Unity. The scripts are coded in C#, and the first goals for the game's outcome are to provide interactivity, immersion, and believability to the user. The ability of the player to interact with the world was analysed in detail to avoid motion sickness and other relevant problems. The interactivity was also improved by trial and error for changing the speed of movements, whether the player will be able to jump or fly. Moreover, using Virtual Reality to play a game can limit/make difficult some abilities like grabbing an object from the ground. In that instance, an algorithm was implemented where objects that can be grabbed are targeted by a line informing and allowing the user to grab them from a distance.

The design of the world space, as long as the design and animation of the characters, was somehow an easy but time-consuming part of the project. To design the world, some helpful Unity tools were used with some additional assets to provide a more meaningful and realistic environment. The character avatars were taken and assigned from some Unity assets, and their animations were also ready. The animators were changed to fit the game's expectations by creating new animators and calling them inside the scripts. Furthermore, the use of the particle system provided by Unity was exploited to implement animations like an explosion, resulting in a realistic and exciting outcome.

As the development of the game was from scratch, it required algorithmic thinking and research on different strategies to come out with an organized code and a sound output. The first step was to implement the fundamental functionalities of the player, which include the ability to move and turn around the world space, to grab objects either directly or from a distance, and more generally the way to interact with some objects. Then, the most important feature of such a game was the creation of enemies. Different kinds of enemies were developed to give a better user experience, changing each enemy's size, speed, health, and damage. The component "NavMesh", of Unity, was used to provide some basic Artificial Intelligence capabilities to the enemies. The idea of the algorithm follows an Object-Oriented design, and it was developed by creating a "Game Manager" script that holds an instance of each kind of enemy and handles the speed and position for the generation of a new enemy during

the gameplay. This information is taken from the "Enemy" abstract class, which is the base class for the implementation of any enemy. "Enemy" is also referencing the "Player" class for the use of dealing damage, earning coins, and position recognition to set to the "NavMesh agents" (enemies) target destination.

Additionally, the subject of the project offers no limitation to a programmer, and that is where the importance of creativity comes. The main goal is to provide a unique solution for people's entertainment. This requires the software to be user-friendly, explaining to the user how to interact and inform him/her at each stage or changing event. To achieve a good outcome, the difficulty of the game must be balanced; a too difficult game would be frustrating, and a too easy would be boring (Eva Kraaijenbrink, 2009). This can be set by adding more features and interactions to the player and setting the winning or losing requirements. To avoid having a boring game, the player must have a target and not just play, which on this occasion is to gather points and get a new high score, this will make the game more challenging for more than one user. The player can survive inside the game world using a gun and gather points/coins by killing enemies. A feature that was added to make the game more challenging and combine the collection of points for a score was to have a reloading system with limitations on magazine capacity. Getting into more detail, the player has a certain amount of bullets in each magazine and a maximum number of magazines that he/she can hold. By killing different types of enemies, the player can earn money that can be used to buy magazines from an "ammo box". The challenge of this algorithm was the combination of an overall magazine limitation that the player can keep with the limitations on magazine capacity for each gun separately. The sum of the coins collected during the game will be calculated to perform the player's final score.

1.2 Problem statement

The creation of a video game on a PC or Console typically needs 3 to 5 years (Wikipedia Contributors, 2019). Around 90% of the games do not sell more than 10 000 copies which show that most of the games are not successful and though not profitable. The implementation of video games requires a lot of research to meet the gamers' expectations, and many positions are required, like programmers, designers, artists, testers, and sound engineers. Concentrating on virtual reality games, the research must be further expanded to result in a profitable game, as on the current timeline, people who can afford virtual reality equipment are more limited. Also, players of virtual reality games get tired easily on certain games due to the complexity of the world. Therefore, it is questionable whether games in a virtual environment can be effective for gamers in the current, challenging gaming industry.

Taking all of the above into consideration, a game nowadays, to be regarded as profitable, must at least offer a professional performance; a clear concept and purpose; a nice design and graphics (including animations, sound effects, and haptic effects); control on the complexity with a user-friendly environment; replayability, allowing the user to try again; synchronisation; interesting and challenging gameplay. Furthermore, different solutions must be analyzed when the game development comes to Virtual Reality matters. The player's movement becomes a more complex problem, while the developer must keep in mind the risk of motion sickness and the illusions of self-motion when searching for a suitable solution. The system for selecting or grabbing objects is another complex task, along with manipulating objects that are released.

Another problem to be considered is the interactivity of the user with UI during the game.

1.3 Aims and objectives

Aims: The initial aim of this project is to perform a successful indie game in a virtual environment by finding solutions to the problems mentioned above. The performance of the program must be considered for optimizations regarding the CPU and memory usage, this will also affect the speed/smoothness of the gameplay. The concept of the game should be intent on a certain theme and give the user a clear perspective on his/her goal. When the ideal concept is met, the design must be developed accordingly and realistic. The final result must be balanced in terms of complexity and overstimulation and provide a user-friendly solution at any stage. Furthermore, the player must have the opportunity to play the game again until he/she achieves the desired goal, and the gameplay must be both exciting and challenging to offer a positive hype to the user.

Objectives: A good 3D game in a virtual environment requires research on different strategies along with some examples of simple games developed. Also, Unity can be exploited, and features for artificial intelligence and game design can be used. Unity offers a profiling tool for the game's performance where memory usage, the speed/frames per second, and other vital aspects are analysed at different stages. Moreover, multi-threading can be applied to enhance the performance further. After some research on other games, the game concept will be decided, and the design will be developed by choosing relevant assets, sound effects, characters, and features. Different features will be developed regarding the solutions for a user-friendly environment, like a home page with instructions, animations and sounds, and haptic events. A pause menu can be implemented to give the player the chance to take a break, restart or quit the game. Activities and interactions will be developed in order to have an exciting result. This could be the creation of different enemies, guns, explosive objects, and many more.

1.4 Solution approach

1.4.1 Software Development

Methodology

The methodology for the project has been researched to find the best possible approach to start the implementation. The findings resulted in a clear decision to use Agile Methodology (Wrike, 2019). The game's development was broken down into more minor problems that were implemented, evaluated, and improved at different stages. For example, the first step was designing the virtual environment, then the functionalities of the player using the Oculus rift hardware, the creation of enemies followed, and additional features were added. At different stages of development, the expectations of the game were slightly changed; therefore, reevaluation of previous features was applied, and they were improved. An example is changing the design of the environment to fit the concept of the game by adding/removing objects to the scene. In addition, the Agile methodology allows the developers to receive information and feedback from the user and use it to make changes to the existing project and for further expansion to meet users' expectations.

Program Approach

The program implemented is in C# and uses the open-source .NET platform. "C# is a general-purpose programming language, mainly designed for creating programs that execute on the .NET framework" (M, L., 2021). The chosen language is most commonly used in unity projects along with the .NET framework, and they can offer a lot of to the user, like scripting backends, managed code stripping, and garbage collection (Technologies, U., 2021). Most importantly, the .NET platform contains a lot of helpful libraries for building applications, including games. Also, C# is an easy-to-use language that can be implemented with an object-oriented approach (Microsoft, 2022). This approach offers some notable advantages, for example, providing a clear structure of the program and keeping the code DRY, is easier to execute and run, and helps to create a reusable application that will be easy to modify and debug (www.w3schools.com, 2019).

1.4.2 Game Approach

The game is built to be played with simple tasks, and present most of the features as quickly as possible, because it is an academic project.

User-Friendly Environment

When developing a video game, a crucial aspect is to keep track of informing the user on each changing state and provide the required information to let him/her know how to play. This is done by starting the game with a home page that will provide information on the game's important features and the buttons that can be used to play the game. Moreover, the player can pause the game at any time, given the option to resume, restart or quit the game. During the gameplay, the users are informed about their health, when an enemy is approaching them, when they hit an enemy when they have been hit, their current amount of money, their current ammo available, and many more. All these are done by one or more of the following techniques using sound effects and animations; displaying information using UI objects; adding or removing relevant game objects (for example the available number of magazines); using haptic effects.

User-Experience

Talking about user experience, the selection of features to be added must be carefully chosen to balance the game's complexity and the excitement that the feature can offer. To avoid computing a too complex game, the user has limited interactions options. The user can only move and turn around the world using a continuous movement, he/she can grab, shoot and reload with one or two pistols, and buy/collect ammo from the "ammo box". To enhance the enthusiasm of the player, some features were added, like explosion objects with animations, four types of enemies, scoring and healing systems, sound and haptic effects, and keeping everything realistic. The game is also set to a certain amount of difficulty to give the user the ability to improve and play enough time the game, but it is not too easy, so the user will not get bored. Furthermore, the profiling tool Unity was used to improve the game's performance by changing some functions or techniques that required a lot of CPU usage.

Game Solution

The features were implemented using Unity's assets, by scripting original codes in C# and findings various techniques after some research. The game can be played using different Virtual

Reality controls, but it was developed using the "Oculus Rift". For the movement of the player and some interactions, some scripts that Unity provides were applied and edited, for any other extra functionalities, the script "Player" has been implemented. Inheritance and polymorphism are used to create enemies, initializing the fundamental functionalities of every enemy in the base class "Enemy" and implementing their different attributes and methods in the derived classes. A gun design is added from the unity asset store along with its script, which was edited to comply with the game's expectations. The reloading system is implemented within the gun's script to handle the ammo of the individual gun, in the "BulletMananger" script to handle the overall magazines that the player can hold, and in the "Ammo" script to handle the ability of the player to buy and refill his/her ammo. Most of the animations were gathered from the unity asset store and the characters. These animations were edited and applied to the enemies to provide a more realistic environment. Some extra animations were created originally, for example, to open/close the "AmmoBox" and the explosions. The difficulty level was set by changing how often a monster will spawn, using a timer that increments the level every 2.5 minutes. The setting of the animations and sound effects were changed, and "Time.timeScale" was used along with "FixedUpdate()" to pause, resume and keep track of everything so that the user can be provided with a synchronised experience.

1.5 Summary of contributions and achievements

The overall result can be considered a successful 3D video game in a virtual environment, as it complies with the basic expectations. The virtual world is designed around the concept of surviving from monsters in an open area. The sound effects and the characters are chosen accordingly, therefore a realistic result, and combined with the activities of the player, a good experience is provided to the user. One of the most significant achievements of this project is enhancing skills for implementing unique algorithms with organized code. The game's development improved skills for coding, creativity, research, getting deeper into a particular language, and using a powerful tool like Unity. Also, some basic skills in designing, graphics, animations, testing, and sound engineering were developed.

1.6 Organization of the report

This report is organised into seven chapters. Chapter 2 provides a detailed explanation of some of the most important fields of research, for example, the selection of software and hardware to be used, what to consider when working with Virtual Reality and game development approaches. Chapter 3 describes the requirements specifications of the game with an overview of the idea and how it is implemented. It also focuses on several techniques, systems, and algorithms regarding both the approach and the coding aspects. This chapter reports the tools that were exploited and the game's design. The next chapter (Chapter 4), labels clearly the testing and validation process. Chapter 5, describes the process of optimising the performance and states the results and key findings of the project. Moreover, some limitations and their implications are reported there. Chapter 6, gives a conclusion of the overall project along with some suggestions for future work. The final chapter (Chapter 7), is a reflection of the experience of developing this project.

Chapter 2

Literature Review

This chapter will analyze different research areas, providing a literature review. Each of the following sections represents one of the most critical areas of research and includes major theories around the subject, existing work and systems related to the field, and an analysis of the correlation between the review and the current work.

2.1 Software & Hardware

Software

The gaming manufacturing has been enlarged over the last decades, concluding to a multi-billion industry. The opportunities have been widely expanded, with many aids on the game development part. These aids are mainly provided through some gaming engines helping developers with both designing and programming aspects. Some of the most important features that game engines offer are, rendering graphics, basic physics calculations, detection of collisions, and applying animations and sound effects. It is significant for game developers to exploit such software and expertise their skills; thus, the selection must be researched in detail.

There are many game engine options, although only the best recommendations, for this year, will be analyzed. The most powerful game development platforms, according to Schardon, L. (2020), are:

- Unity
- Unreal Engine
- Godot
- Phaser
- GameMaker Studio 2

The project represents a 3D video game, therefore "Phaser" and "GameMaker Studio 2" are the first options that are removed from the selection list, these engines have many helpful tools and features, but they focus on the development of 2D games. Godot is one of the most known and helpful 2D and 3D game development engines. It is suitable for developers who want to use their programming skills and implement unique algorithms. Also, it supports C++

and C#, which are commonly used for such projects, although it offers a unique programming language called GDScript (MUO, 2019). There are many more features that Godot can provide to its users, but it is not suitable for this project as it does not include any Virtual Reality interfaces by default. The two softwares that are the most suitable are Unity and Unreal Engine. Both of them can be used to create a 3D video game in a virtual environment and using the engines' features, the implementation will be much easier and still result in a professional outcome. Although, Unity excels in VR and AR because the plugins are very versatile and integrate into the overall XR infrastructure. Both engines have good graphical capabilities and a marketplace where free assets are available. Unreal Engine can be best for high-end graphics, huge team projects for AAA games, and a very good choice for VR. On the other hand, this engine is not recommended as the best choice for solo projects, and it requires very powerful computers due to its graphics. Also, Unreal Engine gives the user a choice to use blueprints, which can be a replacement for the coding part. The two engines seem to be suitable for this project; although Unity is selected due to it is VR and AR components, it is better for smaller/solo projects and is a better choice for a project like this, which focuses on the improvement of coding skills (Buckley, D, 2020) and not on realism and graphics. Therefore, there will be a more detailed research regarding Unity.

Unity was founded by Nicholas Francis, Joachim Ante, and David Helgason, in 2004. The company was recently valued at around 6 billion dollars and is now considered one of the most popular game engines (TechCrunch, 2019). Unity natively supports the C#, and uses the open-source .NET platform (Unity Technologies, 2019). Some examples of video games implemented in Unity are, "Cuphead", "Pillars Of Eternity", "Fe" and "Escape Plan". Concerning Bosnjak, M. and Orehovacki, T. (2018), Unity's main element is a scene; a scene can be considered as a level or a menu. Everything that a level requires can be implemented in a scene by adding game objects and components. Game objects can represent characters, objects, buttons, or even the camera. A game object must contain components in order to have certain functionality. The "transform" is one of the most important components, and it sets the position of the object. Other components can be used to add physics, to allow collision, for animation and many more. Scripts are also added as components to the selected game objects.

This conference cites a 2D game developed in Unity. There is some basic explanation of the framework's functionality, and some C# scripts are presented. Also, there is a questionnaire that 96 people have responded to evaluate the overall game. The project can be useful for initial ideas on how to start implementing a video game in Unity. There are some functions presented for the movement of the player, although there are assets that can be used for such tasks from Unity's asset store.

Hardware

The hardware that has been used was accessible through the University of Reading and included a computer along with the Oculus Rift. According to TechRadar (2019), and N.P. last (2021), the Oculus Rift was one of the first VR headsets that were launched in 2016. Oculus was ready to introduce an enjoyable Virtual Reality experience using their first headset. The headset can be used with a wired connection of 2 USB3.0 to a PC, and it can drive a 1080 x 1200 image resolution to each of the two lenses. The minimum requirements for the computer are an Intel

i3-6100 or equivalent processor, 8GB of RAM, and an Nvidia GTX 960 graphics card. The Oculus Rift package contains the primary headset along with two wireless controllers. The headset's and the controllers' position can be tracked and used in a 3D virtual environment. This is done by setting the two Oculus Sensors that are also included in the package. Even though the headset requires a wired connection, the wires are long enough to give to the user the possibility of playing while sitting or standing. Also, the sensors can track all of the movements even if the user makes a 180 degrees rotation. It is paramount to fit the headset to the correct position on the head using the straps; otherwise, the image will look blurry. Moreover, the headset contains two ear pads responsible for outputting the sound, the quality of the sound is good enough to let the user hear all the in-game sound effects, while the system offers a 3D surround sound. The controllers' buttons can recognise both when they are pressed or touched, giving broader possibilities to the developers.

2.2 Virtual Reality

Virtual Reality was implemented based on ideas from the 1800s, and the first VR headset was created in 1968 (Virtual Reality Society, 2015). Until now, this concept has been rapidly improved and turned out to be very helpful for many workplaces like medicine, psychology, education, and many more. Virtual Reality is a computer-generated domain that is used to create a simulated environment to immerse the user (Iberdrola, 2019). Virtual Reality technology is currently considered the technology with the highest growth potential. As reported by Liang, Z., Zhou, K. and Gao, K. (2019), the use of such a system, especially when combined with gaming, can offer critical advantages in education for safety matters. Some of the advantages stated are that a VR education game can provide an immersive virtual environment with physics feedback, allowing the users to be trained in a safe and realistic habitat. Also, the game design can attract the trainees to keep completing their tasks with an entertaining experience. It is significant to mention the ability to record the training at any stage and analyze the details in the future. Moreover, the use of VR for training purposes can be a cheaper solution instead of using full-scale physical training scenarios. The article suggests that immediate physics feedback, interactivity, and immersion are some of the essential aspects of a user's engagement with the game.

The journal presents a 3D educational game in Virtual Reality and targets evaluating the effectiveness of a game for interactive safety training specific to underground mines. There is a good overview of Virtual Reality aspects and how they can be combined with real-world scenarios. The importance of using this technology with gaming is clearly stated with examples of specific advantages. This can be a valuable introduction to Virtual Reality and to some features that must be considered when developing a game.

2.3 Game Development

The Importance of Game Development

The topic of video games turned out to be very important in people's life these days. Whether a person is a gamer or a developer, video games can offer numerous advantages. Playing a video game can positively impact memory; for example, using different buttons for completing tasks exercises the hippocampus. This is when the brain converts short-term memory to long-term memory, and even controls spatial memory. Likewise, many games can improve

perception and decision-making skills. Complex environments and action games can enhance the ability to create perceptual templates. Playing fast-paced games can result on making accurate choices quickly, even in real life. It is obvious, that video games can also entertain and changing the mood of the user positively (Record Head, 2020). At a developer level, getting into game development helps to develop problem-solving skills; tasks like performance optimization, physics, mathematics, and security problems, can be crucial to sharpen such skills. Moreover, it can improve the ability to create new/unique things, as long as improving imagination and designing skills. Game development can combine the ability of self-learning with teamwork skills. A developer has the opportunity of turning his/her ideas into a game and creating a project alone, focusing on different aspects such as programming, game designing, sound engineering, and many more. Although, when working on a bigger project, a developer must focus on one of these aspects and even combine his/her skills with other developers on that certain aspect. This can offer a massive improvement in teamwork skills (GeeksforGeeks, 2021).

Game Development in Virtual Reality

When game development comes to a combination with VR, some additional problems must be considered and different techniques to be applied. In consonance with Steed, A., Takala, T.M., Archer, D., Lages, W. and Lindeman, R.W. (2021), a technique that must be analyzed for implementation is object selection and grabbing. The implementation starts by recognising a selection from a short distance (under 2 meters), by moving the controller or the headset that has been attached with a capsule, facing through the object. This way, using a collision, the object will be selected, and by pressing the trigger button, the object can be grabbed. Although the hand-pointing or hand-eye pointing did not thank all users, suggesting to use the vector between hand-eye and hand-pointing. Another vital system that was explained is the "Tomato Presence". The Tomato Presence disappears the hand when an object is grabbed, therefore the user can only see the object. This system solves a common problem, which is how the hand will look while it is grabbing different objects. Also, the "Snap Drop Zone" technique is suggested to solve the problem of manipulating the released objects. The two obvious ways of solving this problem are to just let the object fall to the ground using physics or remove the physics and let the object float. Despite this, the Snap Drop Zone solution suggests placing the object in its initial position and rotation. Another problem that must be considered is mapping the player's movement within an infinite virtual environment. A locomotion technique attempted to address this problem, although there are several locomotion solutions to be analyzed. A solution called "Point-and-Click Walking" gives the user the option to choose a location destination using a curved selection arc, and then the player will automatically start to walk towards that point. The automatic walking increases the risk of motion sickness as it is a virtual movement. Another interesting solution is the "Interaction Scale Adjustment", which advises changing the scale of the player or the virtual world. Increasing the player's scale gives the user the possibility to traverse through large virtual worlds using natural locomotion. Furthermore, a locomotion technique that can be used is the "Task-Oriented Teleportation". This technique lets the player use a teleportation system with the restriction on the teleportation destinations to specific predefined points.

The techniques presented in the journal for the implementation of a video game in a virtual environment were at a high level of significance. Different techniques can be useful for different kinds of games to help avoid risks like motion sickness. These systems created a consideration

on how to implement such complex tasks and get into more detailed research to fit the game's specifications. The different solutions are helpful and exciting, but there are some additional problems when implementing a video game in VR, and they must be carefully solved with appropriate solutions.

Game Development Approach

Game development combines complex tasks from different domains, like software engineering, graphics designing, sound engineering, and many more. Therefore, planning and managing the development of a game requires clear strategies and continuous evaluation. In regards to Zelia B. and Dorian G. (2017), the first step for implementing a video game is to find a story/concept to keep the users interested. This can further motivate the player by creating a transition scene between the levels of the game. Then the virtual world is designed around the selected concept, and the first enemies are created. Artificial Intelligence controls these enemies, and the player's task is to protect himself/herself by shooting them with different kinds of guns. In the beginning, the player has some time to explore the world and get used to it, although in the following stages, the enemies start to fight him/her, and at the end, a monolith appears in the scene with an unknown location and the player must follow its sound signals to find it. At that point of the development, changes to the current specifications are applied to include new ideas that can improve the game experience. This can be including new features, like power-ups, changes to the scene, or even setting the number of possible enemies spawns in the scene.

The article provides an example of a 3D survival/action game developed in Unity 2017.3.1. The development approach was explained, providing an idea of the steps that must be taken on the road to creating such a game. The game story and its correlation with the design were clearly presented. Also, the use of Artificial Intelligence for gaming was applied.

2.4 Critique of the review

Research is one of the most vital procedures that must be taken to complete a professional project. There are many considerations that arise and several techniques to be used in any field of work. Different examples of related work can always be helpful, and they can widen ideas and improve the way of thinking. Some meaningful systems were analysed in this section with reference to journals, articles, conferences, and websites related to this project. Some of the main findings from those sources are:

1. The selection of a suitable gaming engine along with the hardware that can be supported. This resulted in Unity, with the use of Oculus Rift and a PC that fits to the minimum requirments (see section 2.1)

2. The benefits of using Virtual Reality and what problems must be highlighted. Virtual Reality, especially when combined with gaming, can be an entertaining and safe solution for training people to work on dangerous environments. Also, it can be helpful in topics like medicine, education and gaming. Although, the implementation must provide immersion to the users, with realistic design, physics feedback and interactivities.

3. Systems to be used for game development solutions in a virtual environment, and possible approaches that can be used. Solutions for selecting and grabbing objects were

gathered as long as different locomotion systems. The most important technique that has been kept for usage is the "Tomato Presence". The approach of the development suggest to find the concept of the game, design the game around this idea providing several tasks to the user, and proceed to reevaluation to apply new/different solutions that can improve the game experience.

2.5 Summary

The literature review is a critical chapter that must be studied to develop a project. It can provide different ideas and solutions for the developer. This can be regarding the methodology that will be applied for the project, the problems that may be faced during the implementation, as well as systems and techniques to solve those problems. The procedure of the research can be timely most of the times, in order to find suitable sources, but the results are vital. For this project, there was profound research on aspects such as game development, use of Virtual Reality, use of Unity, combining artificial intelligence with gaming, enhancing the performance of an indie game, and many more. However, the major aspects were briefly described stating some of the helpful findings.

Chapter 3

Methodology

3.1 Requirements specifications

3.1.1 Game Specifications

A great game requires a concept/story to keep the users' interest. "A game concept, in its simplest form, is the easy-to-understand vision you have for your game" (www.pluralsight.com), 2020). The concept and purpose of this video game were selected to provide an entertaining experience to its users as well as to allow the implementation of infinite features/ideas that can relate to the gameplay. It is a survival game with the purpose of surviving from different kinds of monsters while exploring the virtual world. The story represents "The Last Man Standing", the last man who survived from the monsters that came to ravage the population. A short story is provided at the beginning of the game to prepare the player and help him/her immerse himself/herself in the idea of the purpose of surviving. During the gameplay, the player must achieve his/her goal by killing the enemies using a gun and gathering points to achieve a high score. The game was implemented to a manageable difficulty level to let the user explore the virtual world. Also, the interactions, the features, and the tasks were kept at a low level of complexity but offered an exciting and enjoyable experience. The game ends when the player dies, displaying a "GameOver" canvas. The environment's design was developed around the game's concept, and as the project does not focus on the graphics aspects, the design was at a good and realistic standard.

3.1.2 Scenario and Task Description

The basic idea for the development of the game was implemented. The player was able to move around the world using the thumbstick of the left Oculus controller, turn using the thumbstick of the right controller, look around the world just by turning his/her head in the physical environment, grab objects using the grip button, shoot when a gun is grabbed using the trigger button and pause the game using the "start" button. The four different types of enemies were implemented with some AI functionalities, spawning at random positions in the scene, and the player could gather points by killing them. The character of each enemy was chosen, taking into account its adaptability and the story of the game. The virtual world was designed as an open area with trees, grass, and mountains to provide a realistic environment. Moreover, some containers, cover objects, and damaged buildings were added to fit the concept's expectations. The design of each of the aforementioned features was at the same level of realism and graphics.

After the basic implementation and evaluation, the specifications were slightly changed with features added to improve the user experience. The home page was developed to appear on the PC screen and let the user read the information without the headset. The objects that could be grabbed were limited only to the guns to keep the player concentrated to the purpose of the game, and the playing area was restricted using interactive walls. Also, the option of jumping was removed to minimise the risk of motion sickness. The enemies were set with appropriate properties, found by testing, like health, movement speed, attack speed, spawn speed, damage per attack, and amount of points earned on death. A feature that was added was a reloading system, which limits the number of bullets that a gun with a full-filled magazine can shoot, and the player can use the "X" or "A" buttons to reload. In addition, the player could keep up to a maximum number of magazines in him/her possession to reload the gun. This system required further implementation considering that the player has used all of his/her magazines. This was solved by adding some "AmmoBoxes" over the scene and letting the user refill the maximum number of magazines. It is essential to mention that if the player is holding guns, they will also be refilled to the maximum. The points earned from enemies have been switched to money, and the process of buying ammo could be completed using the "Y" or "B" button (the above button for each controller) to buy the ammo and then the "X" or "A" button to collect it. Another significant development was the creation of levels, and it was done by setting the spawn speed of each enemy. The first level is easy enough to let the user get used to the virtual world; each level lasts for 2.5 minutes, then the next level will be reached, and the spawn time of enemies will be reduced. The gun must be efficiently used due to the limitation of the capacity in order to earn enough money to refill the ammo. This task becomes difficult enough on higher levels. Therefore a new and interesting way of killing the enemies was added. This feature includes an explosive object where it can be shoot and explode, dealing high damage to all of the nearby enemies. The range of the explosion could be approximately calculated by the user observing the explosion animation. 3D sound effects were also added to help the user recognise upcoming enemies' positions, an empty magazine, the procedure to the next level, and others. Additional sound effects, haptic effects, and animations like when enemies walk, run or attack, when shooting, on explosion occasions, and many more, are applied to provide a realistic outcome. The final result recommends that the user plays by sitting on a turning chair to receive the best experience.

3.2 Analysis

3.2.1 Development Approach

The development of this project follows an Agile methodology, which is an iterative approach to project management. This approach suggests developing the software incrementally by breaking up the overall work into smaller tasks. These tasks are known as iterations or sprints, and they are developed according to their importance. The first task for the development of the game was done by research to find a suitable concept and purpose. Then an initial design around the theme was completed, the design of the virtual environment was considered to be implemented as an open area with trees or a more town-like world with more buildings. The idea of the open area was selected to provide the immersion of surviving from monsters and being alone. Furthermore, this environment offers a wider option for other implementations as more features can be added without changing the concept. The next step was the implementation of the player's functionalities using the VR hardware. A lot of research was done, and many examples were analysed to choose the techniques that can suit to the game. The systems were implemented but were changed at later stages to avoid problems and provide a

better user experience. Then, the enemies were implemented, and a way of fighting them was considered. This was concluded by the idea of using guns, and the first one was a handgun that was developed to kill the enemies by shooting them. At that stage, all of the critical tasks were completed to provide a video game with a particular concept and purpose; hence evaluation and testing were applied to improve the gameplay by changing/adding properties, designs, and features. The addition of the reloading system provided a more challenging experience and a balanced complexity to the survival game. The different kinds of enemies provided a more exciting play mode, so different guns were also added. This was tested, concluding with unwanted results of a frustrating experience, as the game was too complex. During the overall process, there were many times for moving back and changing the features to balance the gameplay.

After the story's definition, the Unity approach started designing the virtual environment by adding the first game object, a terrain, to form the ground. Using terrain tools, the height of the terrain was edited to create mountains and generally a more realistic ground design. Textures were then used to give an initial design to the virtual world, such as grass and rock textures for the mountains. Trees, grass, and flowers were also added to enhance the believability of the world during the gameplay. To visualise the virtual world to the user and give the possibility of playing and interacting, there must be at least one camera to the scene (information about what is a scene, a game object, and a component in Unity can be found in section 2.1). The player was developed by creating a new game object called a "VR rig", having a camera offset as child game objects and an actual camera as a child of the camera offset, and the camera was initialised in a script to track and be synchronised with a VR headset. Also, the camera offset contains two child game objects for displaying the hands of the player according to the VR controllers and two ray interactors to implement the ray selecting and grabbing systems. Some components were added for the functionality of game objects, like ready scripts from Unity, colliders to detect collisions, a rigid body to add physics, original scripts, and many more.

3.2.2 Techniques, Systems and Properties

Several techniques and systems were developed for the functionality of the game. Considering that the game is in Virtual Reality there was a detailed research on what techniques should be used to offer the best user-experience around the concept of the game.

Home Page and Pause Menu

The game starts with a home page introducing the game to the user, it provides information about the buttons that could be used, when the "Manual" button is pressed, and the option to start the game by clicking the "Start" button. During the gameplay, the user can click the "start" button from the Oculus controller to pause the game, at this stage, everything that runs in the virtual environment pauses, along with the animations and sound effects, and a canvas appears to give to the player the option to resume, restart or quit the game. In the beginning, the interaction of the user with the UI was done using a ray cast system with the Oculus controllers, but at a later stage, this turned out to be disturbing; therefore, it was changed to appear only on the PC screen and the interaction to be done using the mouse. With this system, the user can pause the game and take a break without the headset.

Player

Initially, the movement and turn of the player follows the standard continuation locomotion system provided by Unity, the properties (e.g. speed of movement) of the system were edited and tested, and the speed of the player is set to 3. The selecting and grabbing of guns was implemented to provide both a direct and a ray cast grabbing system, and the technique of Tomato Presence was applied (see section 2.3). The camera movements (i.e. position and rotation) is controlled by the tracking system of Unity using the functionalities of the Oculus headset and Oculus sensors. The health of the player is set to 6 and as he/she loses health a "blood overlay" image appears to the borders of the headset screen, in order to figure out an approximation of the health. This is an original technique, where the transparency value of the image is inversely proportional to the health, therefore the equation is as shown in Eq. (3.1),

$$x = \frac{1}{h}. \tag{3.1}$$

where, x is the transparency value and h the health of the player. When the health is at 6 and at 1 the image's transparency is set at the minimum of 0 and maximum value of 1, respectively.

Enemies

The enemies were implemented using "Inheritance" technique (www.w3schools.com, 2022) provided by C#. The 4 types of enemies provide a more interesting experience to the user and increase the skill difficulty that can be improved to achieve a new high score. All of the enemies use the "NavMesh" component of Unity for to get some basic functionalities of Artificial Intelligence. Each enemy have an initial spawn speed, and on each level the time is reduced by 2 seconds until their minimum spawn speed is reached. The characters of the enemies were imported from Unity asset store, along with their animations. The animations were used to create a new animator according to their functionality and game expectations. Animations and sound effects were applied for each enemy's walking state, running state, attacking state and spawn. The type of enemies are:

1. Regular Enemy - This is a zombie looking, and is the most common enemy with a balanced level of difficulty, it can follow and attack the player. Properties:
    - Health = 2
    - Attack Speed = 1.8s
    - Movement Speed = 2
    - Damage per Attack = 1
    - Initial Spawn Speed = 16s
    - Minimum Spawn Speed = 6s
    - Coins Earned on Death = 15
2. Giant Enemy - A bigger version of the regular enemy, the high amount of health increases the difficulty for killing this rock made giant. Properties:
    - Health = 6
    - Attack Speed = 3s
    - Movement Speed = 1.4

- Damage per Attack = 2
- Initial Spawn Speed = 28s
- Minimum Spawn Speed = 16s
- Coins Earned on Death = 40

3. Running Enemy - A high speed monster that chase the player and explode dealing high damage, but if the player is conscious it can be easily killed. Properties:
   - Health = 1
   - Movement Speed = 4
   - Damage on Explosion = 3
   - Initial Spawn Speed = 54s
   - Minimum Spawn Speed = 20s
   - Coins Earned on Death = 10
4. Patroling Enemy - The enemy's functionalities are separated into 3 stages, when the player is away it randomly move around the virtual world, if the player is close enough to his sight range it chase the player and when it gets close to the player it attacks. This guard is ready with its armor to kill the player. Properties:
   - Health = 3
   - Attack Speed = 1.4s
   - Movement Speed = 1.8
   - Damage per Attack = 1
   - Initial Spawn Speed = 40s
   - Minimum Spawn Speed = 30s
   - Coins Earned on Death = 20

Pistol and Ammo

The pistol was imported from Unity asset store including the design, the animations and animator, and the script for shooting bullets. All these were combined together and the script was edited to take input from Oculus controllers and to implement the reloading system algorithm. Another original system that was implemented, is that the damage that the bullet will deal to the enemies will be measured according to the distance. The distance could be measured with time (seconds), therefore a stopwatch was developed to start when the bullet was shoot and calculate the damage when it collided with an enemy. The maximum damage by a bullet is set to 1, and the maximum damage can be dealt to enemies up to 2 seconds away. In Eq. (3.2), the damage x is calculated,

$$x = \frac{D_{max}}{t}. \tag{3.2}$$

where D_{max} is a constant of the maximum distance in seconds, that does not affect the maximum damage value of 1, and t is the time in seconds, until the bullet hit an enemy. The reloading system, includes 12 bullets for each magazine and let the player have up to 4 magazines on his/her possession. The player can reload the pistol every 2.5 seconds and he/she needs at least 1 extra magazine. To buy extra ammo the user need to get near an

ammo box, and a canvas will appear with different instructions for each state. The cost of the ammo is at 85 coins and the player's extra magazines must be less than 4. The ammo box opens when the ammo is bought and closes when is collected. There are 3 ammo boxes all over the game world. Different sound effect were applied when the player, shoot; reload; shoot but the magazine is empty.

Explosive Objects and Healing System

There are 9 gases in the virtual environment in total. Each gas can be explode by shooting it twice. When it explodes an animation that was implemented with Unity's particle system, along with a sound effect, starts and it deals a damage of value 3 to all of the nearby enemies, and the gas will be destroyed. The healing system starts 2 seconds after the player collides with a healing item and it heals the player with a value of 1 every 1.5 seconds, until he/she being attacked or the health have reached the maximum. There are 8 healing items randomly spawn in the world and each time the player collides with one a new will be spawned in a different position.

Use of Artificial Intelligence

The Unity offers the "NavMesh" component that helps on the development of objects with AI functionalities. This component was used to the enemies, setting the different height, movement speed, the step length and the maximum slope that they can walk. All these were taken into consideration along with the obstacles in the virtual environment, to bake and calculate the possible paths that can be used for each agent.

Synchronisation

The Unity offers a good system to help the synchronisation of the gameplay by editing the frame rates and using "FixedUpdate" and "Update" methods, etc. The synchronisation is requisite to meet the realistic result that is expected from the game's requirements, therefore the updates were set to run according to real time, and the animations and sound effects were edited to be synchronised with the functionalities of the gameplay. For example the hand of the enemy is moving to hit the player the time that he receives the damage (according to the enemy's attack speed) the punch sound effect plays the same time of the hit. Moreover, this is the reason for the use of timers on different tasks like measuring distance, healing, and many more.

3.3 Implementations

3.3.1 Tools

The most important tool that was used for the development of this project was Unity 2020.3.26f1, which is a powerful game development engine, famous for both 2D and 3D games, and especially for mobile applications (more information about Unity can be found in section 2.1). Unity was exploited using different tools for the designing and the implementation of the game. Some examples are, a terrain with designing tools, lighting and shadow systems, UI elements, virtual reality components (i.e. locomotion system, grabbing system, etc.), artificial intelligence system ("NavMesh"), particle effects, assets imported from the Unity asset store, and many more.In addition the documentation of the work was written in LaTeXusing the "OverLeaf" tool. This is a very helpful tool for big projects, as it helps you in organising

sections with different files, there are several commands for internal and external references, for mathematical formulas and code listings, and generally it has a functionality of automating the content and design. The tool "Draw.io" was also used for the creation of UML diagrams. The assets that were used in the project are listed below.

- Terrain Sample Asset Pack - This asset provides tools for the designing of the terrain (assetstore.unity.com, 2021*d*).
- Conifers [BOTD] - Trees that were used for the designing of the virtual environment (assetstore.unity.com, 2021*a*).
- Grass Flowers Pack Free - Grass and flower textures for the designing of a realistic environment (assetstore.unity.com, 2019*a*).
- Realistic Sandbags - Cover objects to design an environment around the concept (assetstore.unity.com, 2017*c*).
- Nature Materials vol.2 - Designing material to add realistic texture on the terrain (assetstore.unity.com, 2019*b*).
- Realistic Rock - Rocks for further designing of the virtual world (assetstore.unity.com, 2016*a*).
- HQ Shipping Container - Containers for designing a more interesting virtual world (assetstore.unity.com, 2021*b*).
- Indian Gas cylinders PBR - Gasses design used for the implementation of explosive objects (assetstore.unity.com, 2021*c*).
- Altar Ruins Free - Objects used for providing a realistic environment around the current conept (assetstore.unity.com, 2018).
- Wooden Barricades - Cover used for designing the world around the concept (assetstore.unity.com, 2020*c*).
- Modern Guns: Handgun - A handgun with animations and the "SimpleShoot" script, used for killing enemies (assetstore.unity.com, 2020*b*).
- Ammo Box - Wooden box design that was used in the implementation of buying ammo (assetstore.unity.com, 2017*a*).
- Oculus Integration - provide scripts that were used for the development of script using Oculus hardware (assetstore.unity.com, 2022).
- Fantastic Creature #1 - The character of the running enemy, provided with animations that were used to create the animator (assetstore.unity.com, 2017*b*).
- Zombie - The character of the regular enemy, provided with animations that were used to create the animator (assetstore.unity.com, 2019*c*).
- Toon RTS Units - The character of the patroling enemy, provided with animations that were used to create the animator (assetstore.unity.com, 2016*b*).
- Mini Legion Rock Golem PBR HP Polyart - The character of the giant enemy, provided with animations that were used to create the animator (assetstore.unity.com, 2020*a*).

3.3.2 OOP Design

"Object-oriented programming (OOP) is nothing but that which allows the writing of programs with the help of certain classes and real-time objects" (GeeksforGeeks, 2018). The principles of Object-oriented programming are explained in section 1.4.1, and a class diagram explaining the design of the code can be seen in figure 3.1, please zoom in to observe the class diagram.

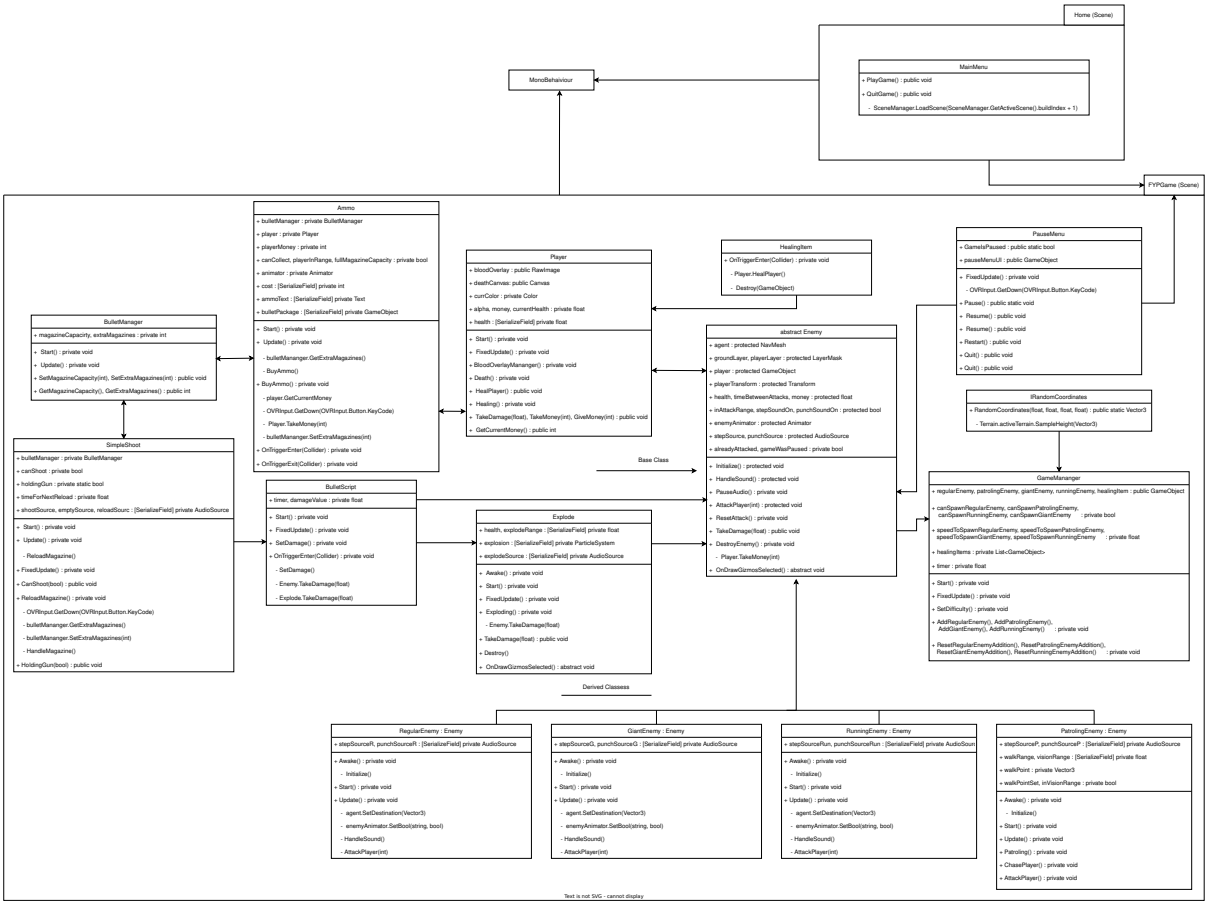

Figure 3.1: Class Diagram for OOP Design.

Unity's C# scripts derive from the base class called "MonoBehaviour" to use the engine's functionality by calling several methods. Therefore, most of the components (scripts) inside the scene inherit this class. At the beginning of the game, a home page is displayed, giving the option to the user to read the manual, quit, or play the game. The "MainMenu" script inside the "Home" scene is responsible for loading the "FYPGame" scene to start the game. This was done by calling the following scene from Unity's Scene Manager, using its index. When the "FYPGame" scene opens, the "Awake" method from each script has been called, and then the "Start" method is called before the first frame. After that, the "Update" is called once per frame, and the "FixedUpdate" is often called even more frequently. Inside the "FYPGame" scene, the "GameManager" is rensponsible for spawning objects like the enemies during the gameplay. For the generation of a random position for those objects, the "IRandomCoordinates" script has been used. The "Enemy" abstract class is the base class of all the enemies; therefore, it has been used to interact with any enemy and another object. This class takes information from the "PauseMenu" script for the recognition of pausing the

game and interacts with the "Player" script for occasions like receiving or dealing damage. The "Player" class is additionally referenced from "HealingItem" script when a player collects a healing item during the gameplay, to start the healing process. The script "BulletManager" keeps information for the current magazines of the player and is used in the reloading system. For example, the "Ammo" script interacts with the "BulletManager" to handle the players' magazines' availability when he/she buys ammo. Furthermore, the "Ammo" class interacts with the "Player" to handle the transaction of buying ammo. The "SimpleShoot" class is rensponsible for the shooting system of the gun, and it also interacts with "BulletManager" to check for magazines to reload. When the bullet is shoot, a new game object is created with the "BulletScript" class, and when it hits an explosive object or an enemy, the class reference the "Explode" and "Enemy" classes, respectively, to apply damage. The "Explode" script also references the "Enemy" when an explosion occurs.

3.3.3 Algorithms and Features

In this sections some of the important features and algorithms that were discussed in 3.1, will be explained in terms of coding and algorithmic thinking, and a more detailed overview of the significant scripts that were mentioned in section 3.3.2 will be presented.

The healing system of the player starts when the player collide with a healing item. The healing item was attached with the "HealingItem" script, therefore the "OnTriggerEnter" method of that script will be called. This method takes the collider of the objects that collides as parameter, this parameter was used to identify if the object is the player, to call the "HealPlayer" method from the "Player" script. The "HealPlayer" method assigns the helath of the player to a new variable called "currentHealth" and uses "InvokeRepeating" to call the "Heal" method every 1.5 seconds. The "Heal" method then adds 1 to the health of the player and to the "currentHealth". In the "FixedUpdate" method the health of the player is checked to stop the healing process when it reaches the maximum or took damage, as shown in Code Listing 3.1

```
1  /* Called approximately twice than the frame rate
2   */
3  void FixedUpdate()
4  {
5      // if the health comes to the max or the player is being attacked
6      if (health >= 6f || health < currentHealth)
7          CancelInvoke(nameof(Healing)); // stops the healing process
8  }
9
10 /* Sets the speed of healing
11  * Called when a healing item is gathered
12  */
13 public void HealPlayer()
14 {
15     currentHealth = health;
16     InvokeRepeating(nameof(Healing), 2f, 1.5f); // starts in 2 seconds
                                                   // and heals every 1.5
17 }
18
19 /* Heals the player
20  */
21 private void Healing()
22 {
23     health += 1f;
```

```
24      currentHealth += 1f;
25  }
```

Listing 3.1: Healing Process

The enemies are implemented using "inheritance" with the "Enemy" script being the "Base Class". In this script, there is a reference to "NavMeshAgent" (controls AI functionalities), the "Ground" and "Body" layer masks that are used to recognize the ground and the player, the "Player" script, the position of the player, and the animator of each enemy. These are initialised in the "Initialize" method that is been called by the "Awake" method of each enemy's script when it is created. The "HandleSound" protected method handles the sound effects of each enemy, and the private "PauseSound" pauses all of the sound effects for occasions like pausing the game or game over. Also, the protected method "AttackPLayer" uses the reference of the "Player" script and the attack speed of the current enemy to apply damage to the player. The function takes as a parameter the amount of damage to be dealt. The "TakeDamage" public method is responsible for animating and applying damage to the enemy that has been hit and checking for the case of death. The "RegularEnemy", "GiantEnemy", "RunningEnemy", and "PatrolingEnemy" scripts are the "Derived Classes", and they use some protected methods and variables from the "Enemy" script as long as some extra to implement their functionalities. For example, "RegularEnemy" calls the "Initialize" method and assigns the unique sound effects in its "Awake" method. In the "Start" method, its properties are assigned (i.e., health, attack speed, and coins earned on death). The "Update" method is used to check if the player is in the enemy's attack range if it is the enemy's attack range is slightly increased to avoid some bugs, and the destination of the "NavMeshAgent" is set to the current position of the enemy. Therefore the enemy stops moving. The regular enemy's attack animation will then start, along with the corresponding sound effect. The sound effect starts by setting the one that needs to be played to false and the others to true and call the "HandleSound" protected method from "Enemy" abstract class. Finally, the "AttackPlayer" method is called passing the damage per attack value. If the player is not in the attack range, then the attack range is set back to normal, the animation and the sound effects are set to the current state, and the destination of the AI agent is set to the player's position.

The "GameManager" script controls the current level (spawn speed of enemies), and creates new objects to the scene. The game objects were assigned to variables as prefabs and the addition of a new object into the scene was done using the "Instantiate" method. The parameters that were passed to this methods are, the prefab instance of the object, the location to be spawned and the rotation. The location of the object is randomly generated using the "RandomCoordinates" function from the "IRandomCoordinates" script. This is a static function and takes as parameters the maximum and minimum value for the x-position and z-position. A number in the range is randomly generated and then the terrain height at that position is checked and assigned to the y-position. The "GameMananger" updates the level using a stopwatch, synchronised with real time (see Code Listing 3.2), and every 5 minutes the stopwatch is reset, the according sound effect plays and the spawn speed of the enemies is reduced by 2 seconds.

```
1 /* Start is called before the first frame update
2  */
```

```
3 void Start()
4 {
5     // Sychronise virtual with physical enviroment by setting the frame
      rate
6     Time.timeScale = 1f;
7     Time.fixedDeltaTime *= Time.timeScale;
8 }
```

Listing 3.2: Synchronise Real Time with Virtual Environment

The script "BulletManager" holds information on the total magazines that are in possession of the player, and it is also used to fill the magazine of the pistol that the player holds when he/she buys ammo. The "SimpleShoot" script was provided by the asset of the gun, imported from the Unity asset store, and contains code for the functionality of shooting bullets and animation of the gun. Although, it was edited to be used by the Oculus controller to shoot or reload. The "CanShoot" method takes as an argument a boolean value that states if the player holds a gun and is called when the trigger button is pressed. When the function is called, if there is enough magazine capacity and the player holds a gun, the "canShoot" variable is set to true else, if there is not enough capacity, it is set to false, and the according to sound effect plays. In the "Update" method, the "canShoot" variable is checked, and if it is true, the shooting sound effect and animation plays, a bullet is fired, and the magazine capacity is reduced by 1. When the bullet is fired, the "BulletScript" starts a stopwatch to measure the distance traveled until it hits an object and calculate the damage value (as explained in section 3.2.2). When the bullet collides with an object, the "OnTriggerEnter" function has been called, taking as a parameter the collider of the colliding object, and the damage value is calculated. Using the collider, the layer mask of that object is checked; if the object is an enemy or an explosive object, their scripts are accessed, again through the collider, and damage is applied by calling their "TakeDamage" function, with the amount of the calculated damage value. This process is presented in the Code Listing3.3 .

```
1  private float damageValue = 2f;
2
3  /* Calculates the damage that bullet will dealt,
4   * according to the distance travelled
5   */
6  private void SetDamage()
7  {
8      damageValue /= timer;
9      // max damage can be 1
10     if (damageValue > 1f)
11         damageValue = 1f;
12 }
13
14 /* Called when collision occures
15  *
16  * @param coll - Collider of the colliding object
17  */
18 void OnTriggerEnter(Collider coll)
19 {
20     SetDamage();
21     // check if it collided with an enemy
22     if (coll.GetComponent<Collider>().gameObject.layer == LayerMask.
       NameToLayer("Enemy"))
23     {
```

```
24          // apply damage and destroy the bullet
25          coll.gameObject.GetComponent<Enemy>().TakeDamage(damageValue);
26          Destroy(gameObject);
27      }
28      // check if it collided with an explosive object
29      else if (coll.GetComponent<Collider>().gameObject.layer == LayerMask.
        NameToLayer("Explosive"))
30      {
31          // apply damage and destroy the bullet
32          coll.gameObject.GetComponent<Explode>().TakeDamage(damageValue);
33          Destroy(gameObject);
34      }
35  }
```

Listing 3.3: "BulletScript" code snippet

The reloading of a gun is checked in the "ReloadMagaine" method of the "SimpleShoot" script. It first check whether the user have a gun, and then if, the "A" (gun in right hand) or "X" (gun in left hand) button is pressed, 2.5 seconds from the previous reload passed and there are 1 or more extra magazines available. The input of the buttons is checked using the command "OVRInput.GetDown(OVRInput.Button.One)" (developer.oculus.com, n.d.), the time between the reloads is measured with the stopwatch system that was earlier explained, and the extra magazines available are checked by calling a getter method for "BulletManager" script. The "Ammo" script, is assigned to the "AmmoBoxes" to handle the process of buying ammo. The "BuyAmmo" method is called on each frame (in "Update"), and it first check if the player is in the colliding range using "OnTriggerEnter" as in previous examples. When the player is inside that range the "playerInRange" variable is set to true and some instruction text is visible above the box, according to the current state. The different states are when, the extra magazine capacity is full; the player has not enough money; the player can buy ammo; the player bought ammo and can collect it. If the player can buy ammo, and the "Y" or "B" button is pressed an animation that open the box starts, the player loses money according to the ammo cost, by calling the "TakeMoney" method from the "Player" script, and the collect state is set to true. When the player can collect ammo and the "X" or "A" button is pressed, the "SetExtraMagazine" method and the "SetMagazineCapacity" from "BulletManager" script are called to full fill the extra magazines and the pistol's magazine capacities.

3.4 Design

The design of the idea targets on solving several problems and aims, with an originality on the development and critical thinking. Firstly, the video game was implemented with the idea of giving infinite possibilities for the addition of features. The survival game story was decided, as it has a certain purpose for staying alive, but can be combined with other tasks to either provide a more interesting experience or extend the goal of the game. The overall experience required to be user-friendly in order to achieve the outcome of a 3D video game that can be considered as successful. This was done using various techniques for different cases. Some of this techniques are, the way to inform the player that is loosing health; to pause the game, along with the UI appearing on PC screen, giving the opportunity to the user to take a break from using the VR headset (i.e. home page, pause menu and game over screen); the selecting and grabbing system with a ray cast; recognising the position of the enemies using 3D sound effects; the transition to the next level without interrupting the gameplay; the UI above an "AmmoBox", giving information/instructions to the user. Moreover, the design of the virtual

environment and the characters, required to be around the concept with balanced graphics. Therefore, the assets that were used for the development, were at the same level of good graphics, and always considering the story of the game. The animations, sound and haptic effects, particle system and the overall design, resulted in an outcome providing immersion and believability to the user.

3.5 Summary

This chapter represents the steps for the development of the 3D survival game in VR. The game's requirements are stated, along with the overall idea. There are instructions on how a user can play the game, how different techniques and systems were used to meet the aims and objectives, and the implementation process for those systems. The development approach is clearly explained and a detailed description on the properties and functionalities of each feature is reported. The use of Object-oriented programming with C# is analyzed and the scripts are described in terms of coding and algorithmic thinking, with some of the important algorithms to be highlighted. The tools that has been used for the development of the game are stated and an overview of the game's design is given.

Chapter 4

Testing and Validation

4.1 Testing Process

The development of the project followed an Agile methodology as explained in section 3.2.1, hence several test methods were applied during the implementation process to recognise problems and edit the results to meet the requirements specification. The four phases of the testing process are presented as follows, unit testing, integration testing, system testing and acceptance testing. The figure 4.1 shows an example for the process of such testing.

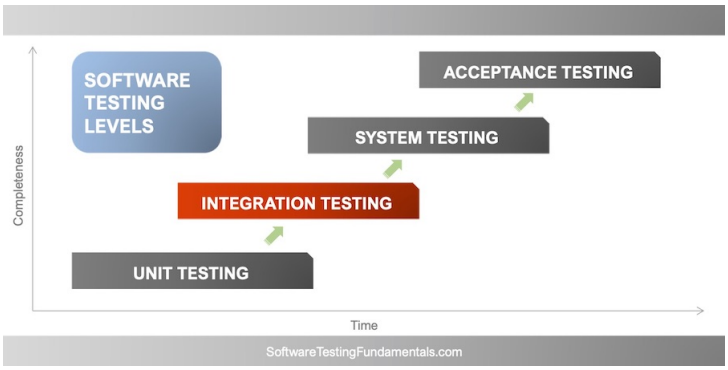

Figure 4.1: Testing Process

4.1.1 Unit Testing

In this phase, the smaller testable parts of the game, units, are individually tested. This testing method was very commonly used to ensure correct implementation of a feature before moving to the bigger picture. Some of the most significant examples will be explained to show different approaches used for the unit test. The initial test regards the player's functionality, where the properties were edited and tested using the Oculus headset to ensure the avoidance of motion sickness and illusions of self-motion. This testing process was done after the basic design of the virtual environment by trying different movement techniques and changing properties like walking/turning speed. For each different state, the technique was tested by moving around in the virtual world for 2 minutes. Another example that must be stated is the creation of enemies. The first step was to create one enemy and then implement a suitable base class for a more safe and easy solution. When a new enemy was implemented, it was tested alone in the environment by removing all of the other kinds of enemies; in this way, the movement and functionality of that enemy could be observed in detail. To set the properties of a new kind of enemy, the game was played with multiple enemies of that kind, examining the changing

properties like health, movement speed, attack speed, and many more. These were finally set by trial and error, according to the desired difficulty level. Moreover, the range of the enemies was visualised using the Unity's method "OnGizmosDraw", giving the possibility of drawing a circle of the range, changing the color of the circle (attack range was set to red and sight range to yellow), and changing the size of the range even during the gameplay. This technique was used for any feature that contained a range for its functionality. An interesting and helpful way of testing was to set the damage of the bullet according to the distance traveled. The distance was measured in seconds from the time that the bullet was shoot until it hits an object. Therefore the first test was to evaluate a constant and synchronised iteration of the stopwatch. This was done by starting a stopwatch at the start of the game, using "Debug.Log" to print the results on the console and compare them with a real-time stopwatch. Then, the "Debug.Log" was used again to evaluate the calculation of the bullet's damage by printing to the console the calculated damage value and how many seconds passed until the bullet hit an object. The general technique and the use of "Debug.Log" were used in other cases related to the timing and some distant test cases. In addition, the player's interactivity should provide a user-friendly system. Thus the interaction between the player and objects/UI was tested. The grabbing system turned out to be brutal and inappropriate for using ray casting with long-distance objects; therefore, the system was limited to short distances only. The interaction with UI was undesired with the VR headset, so the UI was decided to be displayed on the PC and the interaction to be completed with the mouse.

4.1.2 Integration Testing

This phase of software testing combines several individual modules/units to test them as a group to comply with a system with specified functional requirements. The objective of integration testing is to ensure the gameplay's correct functionality, detect any irregularity between the units, and satisfy the demands for creating a successful game. The test case that can be considered with the highest level of importance is the system of setting the difficulty of the game experience. The project's requirements suggest balanced game difficulty. To evaluate this, the difficulty of each kind of enemy, combined with the spawn speed and the time that the user needs to buy ammo, were considered. The spawn speed was calculated in seconds; thus, the same technique with a stopwatch was used, as mentioned in the previous section. Then, the gameplay was tested by trial and error to calculate the final difficulty with appropriate spawn speed values for each enemy. Furthermore, the addition of levels made a notable contribution to concluding with a balanced difficulty, so the game was tested again, trying different values regarding the initial spawn speed, and the minimum spawn speed; the duration of each level; how the spawn speed will be affected on each level. The table 4.1, shows the enemy's contribution to the difficulty of the game, combined with the spawn speed.

Table 4.1: Enemy contribution to difficulty

	Enemy		
Type	Difficulty	Initial - Minimum Spawn Speed(s)	Final Difficulty
Regular	easy	16 - 6	balanced
Giant	balanced to hard	28 - 16	balanced
Running	balanced	54 - 20	balanced to hard
Patrolling	easy	40 - 30	easy

The animation and sound effects of the enemies were tested to be synchronised with their functionality, in order to provide immersion, and a realistic gameplay experience. The animations were first synchronised with the functionality, i.e. when the player receive enemy the animation of the enemy must hit him the same time, and then the sound effects were added and synchronised too. This was tested by trial and error again, changing the speed of the animations and the speed/duration of the sound effects. The reloading system was another feature, combined with buying ammo, that affected the difficulty of the game. Hence, the capacity of the magazine, the maximum number of magazines and the complexity for the process of buying ammo were evaluated together to provide an appropriate result.

4.1.3 System Testing

System testing takes as input the components of integration testing to detect defects within both the integrated units and the whole system. This testing method evaluates the whole system in terms of requirement specifications and functional requirement specifications. Respecting this projects the system testing method was used to test the overall game. For example, the properties of the enemies were tested again combined with the reloading/buying ammo system and explosive object's properties to balance the complexity of the game and enhance the hype. Changes were applied on the aforementioned properties, along with the money that the user gains by killing an enemy. The designing of the virtual environment was also considered on this phase of the testing process to ensure the, animations, sound effects, characters, purpose of the game etc, are concentrated around the selected concept.

4.1.4 Acceptance Testing

This is the final phase of the testing process, and it takes the users' feedback and evaluation. For the acceptance testing, a questioner was created and given to 5 people that played the game and their answers were reported and visualized. The following questions were included in the questioner:

1. Rate the story/concept of the game (1-10).
2. Rate the designing of the game in terms of animations, sound and haptic effects (1-10).
3. Rate the design of the game in terms of graphics (1-10).
4. Rate the difficulty of the interactions, selecting and grabbing guns, interaction with UI and shooting enemies (1 very difficult - 10 very easy)
5. Rate the difficulty of the gameplay (1 very easy - 10 very hard).
6. Rate the performance of the game (1-10).
7. Rate the overall experience of the game (1 weak - 10 excellent).
8. Write an overview of your experience, what you like and what you do not.
9. Write some recommendations that in your opinion, can improve the gameplay experience.

The results of the questioner were mostly good, the users suggested better graphics and an extension of the story. Although, their experience was good, stating that the gameplay was interesting and the ratings showed a success on believability and immersion. Furthermore,

the answers were concluded and visualised to 4 sections, design, interactivity, user-friendly environment and difficulty. This can be observed in figure 4.2

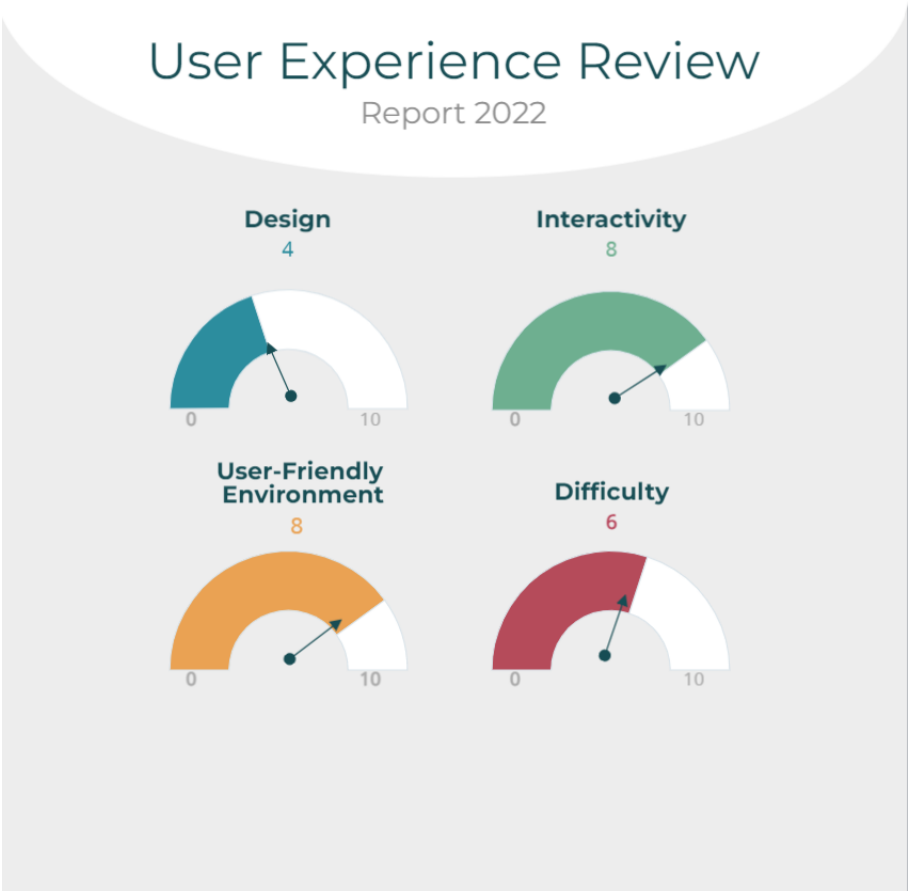

Figure 4.2: Ratings Visualisation

4.2 Summary

The testing and validation process is significant for software engineering projects. In this section, the test phases were described, and different test techniques and test cases were reported. The testing process help in editing the game to outcome with the desired aim and objectives described at section 1.3. It should be noted that the evaluation concluded that almost all kinds of enemies have a slower speed than the player's to give the user enough time for the buying ammo process.

Chapter 5

Results and Discussion

5.1 Performance

The performance of the game was observed using the stats bar and the profiler tool, that Unity provide. The initial changes to settings were made to improve the performance without loosing quality. The first optimizations were applied following this site. At first, the lighting settings were edited, and the most important changes were to set the mode to "baked", the intensity of the light from 1 to 0.8 and the shadows type from soft to hard, resulting to an improvement of the fps from 11 to 22 and a reduction on the shadow casters from almost 650 to nearly 100. Then all of the stationary objects were marked as static, reducing the number of batches from almost 700 to around 400, this helps both CPU and GPU performance as Unity renders all the static objects together, at this stage the graphics were rendered to up to 30 fps. Moreover, the graphics were changed from the highest quality to a balanced/high quality, and some additional edits were applied concluding to the optimized result of almost 40 fps, as shown in figures 5.1 and 5.2. These changes helps the GPU reduce the workload, and enhance the performance of the system.

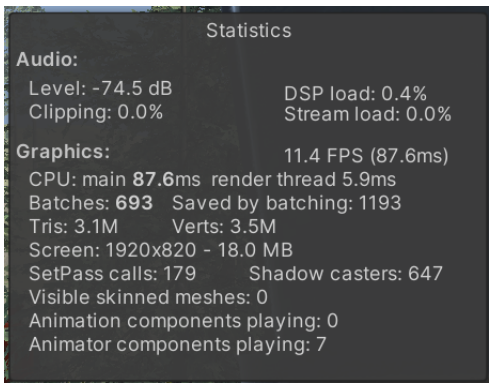

Figure 5.1: Initial Statistics

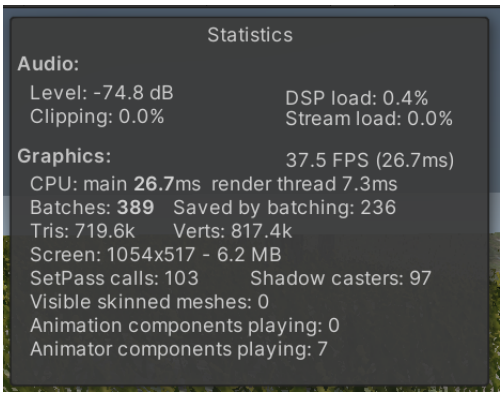

Figure 5.2: Optimized statistics

5.2 Significance of the findings

The results of the project draw to a close of the development of a successful 3D game in Virtual Reality. The goals of providing interactivity, immersion and believability to the user, were addressed with the implementation of a user-friendly environment, an interesting gameplay with balanced difficulty and complexity, and a realistic design around the concept.

The different types of enemies and the reloading system resulted to an original solution for the enhancement of an exciting experience. The testing process helped to set the most suitable properties to the implemented features and balance the complexity of the game. Regarding the considerations for the development of VR systems, the background research gave a vital advantage with techniques that were used and/or analysed to minimise the risk of motion sickness and implement some user-friendly systems. The pause menu is another feature that issued the game with a user-friendly environment and repeatability. For the enhancement of the user-experience the animations, the sound effects, the explosive objects and the realistic design hook an important role. Also, the organisation and approach of the scripting, with object-oriented programing, concluded to a smaller code that can be easily analysed, edited and used for future work. The performance of the game was at a good level, it is significant to mentioned that the performance was tested with a laptop (Dell Inspiron 7380) with the characteristics of a processor Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz, 8 GB RAM, 64-bit operating system and a GPU Intel(R) UHD Graphics 620. The performance of the system is predicted to be higher enough on more powerful PCs.

5.3 Limitations

The development of the game was mostly implemented with the Dell Inspiron 7380 laptop, that was mentioned in the previous section. This caused a significant limitation on the performance and testing process, as the Oculus rift require more performant specifications. This led to the solution of using the PC and VR hardware provided by the University of Reading between 8pm to 4pm on weekdays. A more powerful PC could provide more time for focusing on the details of the game, the enhancement of the performance and the addition of further features. The time that was limited and the restriction for the ability to test the game concluded to the implication of focusing to the details and to testing problems, having an extremely small timeframe to implement a multiplayer edition.

5.4 Summary

This chapter analyses the results and the key findings of the project. An overview for the performance of the game, and an explanation for the significance of the findings is reported. The limitations state some problems that resulted to the disability for improvements of the findings.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Taking everything into consideration, the problem that this project has addressed was the development of a successful 3D video game in Virtual Reality. The aims and objectives that were set for solving such a problem were implementing a game that can provide interactivity, immersion, believability, a user-friendly environment, and balanced complexity/difficulty. The project followed a specific approach for the development of different techniques and systems that addressed each problem. This was the selection of an exciting concept, a realistic design around the selected field, a clear purpose of the game, the enhancement of the performance to provide a smooth game flow, and a balance between complexity and overstimulation. The results that have been vital to the game's development are the player's locomotion system, selecting and grabbing technique, implementation of different enemies that use AI functionalities, animations, sound effects, and the reloading system. In addition, the project targeted academic results. The selection of the concept and implementation of several features helped enhance coding skills, algorithmic thinking, and creativity. The use of VR components improved skills for research and finding original solutions for unfamiliar problems. Game development can force a developer to get familiar with designing, animations, sound engineering, and testing skills. The experience of the game proved how powerful software could be when the physical environment is combined with a virtual environment.

6.2 Future work

Before the start of the project, the initial aim was the development of a survival single-player and multiplayer 3D game in Virtual Reality. During the background research, it turned out that developing a game, especially in Virtual Reality, can be a very complex and timely task. It requires focusing on many aspects to conclude with the desired result and spending much time to find techniques that can solve several problems and focus on details. The limitations that have been faced concluded to the implication of a smaller time frame, and thus the multiplayer edition was researched but not developed. The game was implemented in C# with an Object-oriented programming approach. This resulted in an organised code that can be easily analysed for editing or additions. There are infinite possibilities for the future with the extension of the developed game. The first addition that can take place is the enhancement of the performance and the use of multi-threading to achieve the best outcome. Furthermore, adding a database can be substantial to create a leader board and save each user's high score. This can be further extended by analysing and visualising the data to

allow the users to observe their progression. The virtual environment could be used/copied to quickly implement a multiplayer option or make other game modes by editing the enemies' functionalities and design. Game development can be combined with more advanced artificial intelligence components and provide extraordinary results.

Chapter 7

Reflection

The experience of the development of this project was unprecedented, as this was the first time working on such a big project, implementing own/unique ideas. One of the challenges faced is the time management and organisation of the approach. The project was implemented using Agile methodology, breaking down the overall problem to smaller tasks; this kind of project requires to meet the deadlines of those tasks strictly. The research part turned out to be one of the most critical aspects. Over the academic years, some good research skills were developed, and the initial research was done regarding some basic techniques with a minimal number of examples for the implementation of the desired features. During the development of the literature review (Chapter 2) much time was spent researching relevant projects or systems, and at that point the significance of that more profound research was identified. Game development is a very interesting subject that has a lot of possibilities, as it can be combined with several other subjects to compute a unique/new result. Also, game development can offer a lot in terms of learning. Some of the best skills that were developed are creativity, algorithmic thinking, and the ability to solve unexpected tasks with original techniques. The creation of games can further offer the ability to map a developer's thoughts/ideas with no limitations. Working with Virtual Reality introduces more complex problems that can enhance a developer's skills rapidly. It is vital to mention the specialisation of skills regarding Unity and coding development. The initial aim of this project was to create a single-player and a multiplayer edition of the survival game in VR, focusing on networking aspects. Although, in the development process of the single-player game, some new aspects came up that could be analysed and implemented, such as the performance, complexity, interactivity, and design of the game. Hence, the work was focused on those aspects, and combined with some limitations that were faced; the multiplayer edition could not be implemented. The overall result was good and covered all of the considered aspects. During the development, there were many ideas to implement the system further. In conclusion, when working on a similar project in the future, a deep research on more specific and relevant fields will be applied, time management will be set to more realistic requirements, and the work will be more efficient because of the familiarity with the subject.

References

Anon (2012), 'The good things about video games'. (accessed March 20, 2022).
URL: *https://mediasmarts.ca/video-games/good-things-about-video-games*

assetstore.unity.com (2016*a*), 'Realistic rock'. (accessed January 4, 2022).
URL: *https://assetstore.unity.com/packages/3d/realistic-rock-51251*

assetstore.unity.com (2016*b*), 'Toon rts units'. (accessed February 20, 2022).
URL: *https://assetstore.unity.com/packages/3d/characters/toon-rts-units-demo-69687*

assetstore.unity.com (2017*a*), 'Ammo box'. (accessed February 18, 2022).
URL: *https://assetstore.unity.com/packages/3d/props/weapons/ammo-box-7701*

assetstore.unity.com (2017*b*), 'Fantastic creature 1'. (accessed February 20, 2022).
URL: *https://assetstore.unity.com/packages/3d/characters/creatures/fantastic-creature-1-103074*

assetstore.unity.com (2017*c*), 'Realistic sandbags'. (accessed January 4, 2022).
URL: *https://assetstore.unity.com/packages/3d/props/exterior/realistic-sandbags-95964*

assetstore.unity.com (2018), 'Altar ruins free'. (accessed January 4, 2022).
URL: *https://assetstore.unity.com/packages/3d/environments/fantasy/altar-ruins-free-109065*

assetstore.unity.com (2019*a*), 'Grass flowers pack free'. (accessed January 1, 2022).
URL: *https://assetstore.unity.com/packages/2d/textures-materials/nature/grass-flowers-pack-free-138810*

assetstore.unity.com (2019*b*), 'Nature materials vol.2'. (accessed January 4, 2022).
URL: *https://assetstore.unity.com/packages/2d/textures-materials/nature/nature-materials-vol-2-32020*

assetstore.unity.com (2019*c*), 'Zombie'. (accessed February 20, 2022).
URL: *https://assetstore.unity.com/packages/3d/characters/humanoids/zombie-30232*

assetstore.unity.com (2020*a*), 'Mini legion rock golem pbr hp polyart'. (accessed February 21, 2022).
URL: *https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/mini-legion-rock-golem-pbr-hp-polyart-94707*

assetstore.unity.com (2020*b*), 'Modern guns: Handgun'. (accessed January 31, 2022).
URL: *https://assetstore.unity.com/packages/3d/props/guns/modern-guns-handgun-129821*

assetstore.unity.com (2020*c*), 'Wooden barricades'. (accessed January 26, 2022).
URL: *https://assetstore.unity.com/packages/3d/props/exterior/wooden-barricades-111243*

assetstore.unity.com (2021*a*), 'Conifers [botd]'. (accessed January 1, 2022).
URL: *https://assetstore.unity.com/packages/3d/vegetation/trees/conifers-botd-142076*

assetstore.unity.com (2021*b*), 'Hq shipping container'. (accessed January 4, 2022).
URL: *https://assetstore.unity.com/packages/3d/props/industrial/hq-shipping-container-modular-203201*

assetstore.unity.com (2021*c*), 'Indian gas cylinders pbr'. (accessed January 4, 2022).
URL: *https://assetstore.unity.com/packages/3d/props/industrial/indian-gas-cylinders-pbr-194471*

assetstore.unity.com (2021*d*), 'Terrain sample asset pack'. (accessed December 27, 2021).
URL: *https://assetstore.unity.com/packages/3d/environments/landscapes/terrain-sample-asset-pack-145808*

assetstore.unity.com (2022), 'Oculus integration'. (accessed February 19, 2022).
URL: *https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022*

Bosnjak, M. and Orehovacki, T. (2018), 'Measuring quality of an indie game developed using unity framework'. (accessed March 20, 2022).
URL: *https://ieeexplore.ieee.org/document/8400283*

Buckley, D (2020), 'Unity vs. unreal – choosing a game engine'. (accessed March 20, 2022).
URL: *https://gamedevacademy.org/unity-vs-unreal/*

Chodos, A. (2019), 'Physicist invents first video game'. (accessed March 20, 2022).
URL: *https://www.aps.org/publications/apsnews/200810/physicshistory.cfm*

developer.oculus.com (n.d.), 'Ovrinput'. (accessed March 20, 2022).
URL: *https://developer.oculus.com/documentation/unity/unity-ovrinput/*

Eva Kraaijenbrink (2009), 'Balancing skills to optimize fun in interactive board games'. (accessed March 20, 2022).
URL: *https://www.researchgate.net/figure/The-Flow-Channel-is-the-right-balance-between-a-challenge-being-too-easy-leads-to*$_{fig1_221054198}$

GeeksforGeeks (2018), 'Oops — object oriented design'. (accessed March 20, 2022).
URL: *https://www.geeksforgeeks.org/oops-object-oriented-design/*

GeeksforGeeks (2021), 'Top 5 reasons to learn game development'. (accessed March 20, 2022).
URL: *https://www.geeksforgeeks.org/top-5-reasons-to-learn-game-development/*

Iberdrola (2019), 'Virtual reality, the technology of the future'. (accessed March 20, 2022).
URL: *https://www.iberdrola.com/innovation/virtual-reality*

Liang, Z., Zhou, K. and Gao, K. (2019), 'Development of virtual reality serious game for underground rock-related hazards safety training'. (accessed March 20, 2022).
URL: *https://ieeexplore.ieee.org/document/8795446*

M, L. (2021), 'C vs c++ comparison: Find out the difference between c and c++'. (accessed March 20, 2022).
URL: *https://www.bitdegree.org/tutorials/c-sharp-vs-c-plus-plus/*

Microsoft (2022), '.net game development on windows, linux, or macos'. (accessed March 20, 2022).
URL: *https://dotnet.microsoft.com/en-us/apps/games*

MUO (2019), '10 reasons to use godot engine for developing your next game'. (accessed March 20, 2022).
URL: *https://www.makeuseof.com/tag/reasons-godot-engine-game-development/*

N.P. last (2021), 'Oculus rift review'. (accessed March 20, 2022).
URL: *https://www.techradar.com/reviews/gaming/gaming-accessories/oculus-rift-1123963/review/2*

Record Head (2020), '10 reasons why playing video games is good for your brain'. (accessed March 20, 2022).
URL: *https://recordhead.biz/10-reasons-video-games-is-good-for-you/*

Schardon, L. (2020), 'Best game engines of 2020'. (accessed March 20, 2022).
URL: *https://gamedevacademy.org/best-game-engines/*

Steed, A., Takala, T.M., Archer, D., Lages, W. and Lindeman, R.W. (2021), 'Directions for 3d user interface research from consumer vr games'. (accessed March 20, 2022).
URL: *https://ieeexplore.ieee.org/document/9523846*

TechCrunch (2019), 'How unity built the world's most popular game engine'. (accessed March 20, 2022).
URL: *https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/*

Technologies, U. (2021), 'Unity - manual: Overview of .net in unity'. (accessed March 20, 2022).
URL: *https://docs.unity3d.com/Manual/overview-of-dot-net-in-unity.html*

TechRadar (2019), 'Oculus rift'. (accessed March 20, 2022).
URL: *https://www.techradar.com/reviews/gaming/gaming-accessories/oculus-rift-1123963/review*

Unity Technologies (2019), 'Unity - manual: Creating and using scripts'. (accessed March 20, 2022).
URL: *https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html*

Virtual Reality Society (2015), 'History of virtual reality - virtual reality society'. (accessed March 20, 2022).
URL: *https://www.vrs.org.uk/virtual-reality/history.html*

Wikipedia Contributors (2019), 'Video game development'. (accessed March 20, 2022).
URL: *https://en.wikipedia.org/wiki/Video*$_{g}ame_{d}evelopment$

Wrike (2019), 'What is agile methodology in project management?'. (accessed March 20, 2022).
URL: *https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/*

www.pluralsight.com) (2020), 'Coming up with an idea for a game concept'. (accessed March 20, 2022).
URL: *http://rochi.utcluj.ro/articole/6/RoCHI2018-Blaga.pdf*

www.w3schools.com (2019), 'C oop (object-oriented programming)'. (accessed March 20, 2022).
URL: *https://www.w3schools.com/cs/cs*$_{o}op.php$

www.w3schools.com (2022), 'C inheritance'. (accessed March 20, 2022).
URL: *https://www.w3schools.com/cs/cs*$_{i}nheritance.php$

Zelia B. and Dorian G. (2017), 'Game development methodology mapped on the evoglimpse video game experiment'. (accessed March 20, 2022).
URL: *http://rochi.utcluj.ro/articole/6/RoCHI2018-Blaga.pdf*

Appendix A

Appendix

CSGitLab link: click here