Virtual Reality Football Videogame

A Social Experience

Bruno Filipe Barata Rodrigues

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Pedro Alexandre Simões dos Santos

Examination Committee

Chairperson: Prof. Nuno João Neves Mamede
Supervisor: Prof. Pedro Alexandre Simões dos Santos
Member of the Committee: Prof. Rui Filipe Fernandes Prada

June 2019
Acknowledgments

I would like to thank my supervisor Pedro Santos for his help, guidance and patience during these long three years of work. He always showed great enthusiasm and confidence regarding this work, even at times when I did not dedicate as much time as I should have. I would also like to express my gratitude to everyone who directly or indirectly allowed me to have all the necessary equipment at my disposal, either Kinects, virtual reality headsets of powerful computers to work on, particularly to Diogo Rato and Ricardo Rodrigues.

Also, I would like to mention all my friends and family who were always there supporting me in the most difficult times. For all of them I express my thanks, but in particular: to my colleague Leandro, who helped me with his knowledge and ability to better understand the problems; to my girlfriend, who never let me give up, even when I wanted to; and finally to my mother, who gave me the possibility to have this academic education and without whom this would not have been possible.
Abstract

Technology is in constant evolution, and the gaming industry is not an exception. The latest developments have been in the Virtual Reality (VR) field, with powerful hardware being developed and released by some of the world's biggest companies, but big gaming studios have not been following the same path. VR games have the potential to take the gaming level one step forward by putting the player in the game's virtual world and creating endless possibilities to be explored by developers' creativity. However, in VR users can feel isolated from the people and the world around them, so shared VR experiences play an important role in the success of VR.

In order to better understand the inherent challenges of the creation of VR content, we propose the development of a VR game. This game should be played by two people, stressing the need of exploring VR as a way of interaction and communication between people, instead of delivering isolated experiences. It consists in a football game with the players facing each other. It is not supposed to be a game on which players run through the pitch; instead they are in their respective goal, kicking the ball in order to score on the opposite goal, and trying to defend the other player's kicks. By making this game a shared experience, we intend to create and strengthen ties between people through a method so propitious to this, which is simply having fun with another person.

Keywords

Virtual Reality; Head-Mounted Display; Football; Social Game
Resumo

A tecnologia está sempre em constante evolução, e a indústria dos jogos não é excepção. Os últimos desenvolvimentos têm sido na área da Realidade Virtual (RV), com algumas das maiores empresas do mundo a desenvolverem e lançarem hardware bastante capaz, mas os maiores estúdios de desenvolvimento de jogos não têm acompanhado esses passos. Os jogos de RV têm o potencial de elevar o nível dos jogos para um outro patamar, ao colocar o jogador no centro do mundo virtual do jogo, abrindo portas a uma infinidade de possibilidades a serem exploradas pela criatividade de desenvolvedores. No entanto, com a RV os utilizadores também se podem sentir isolados das pessoas e do mundo real à sua volta, ganhando as experiências de RV partilhadas um importante papel no sucesso da RV.

De forma a perceber melhor os desafios inerentes à criação de conteúdo em RV, propomos o desenvolvimento de um jogo em RV. Este jogo deverá ser jogado por duas pessoas, salientando a necessidade de explorar a RV como um modo de interacção e comunicação entre as pessoas, ao invés de ser uma experiência isolada. O jogo consiste num jogo de futebol, em que os jogadores se defrontam. Não é suposto ser um jogo no qual os jogadores andem pelo campo inteiro; em vez disso, cada jogador está na sua respectiva baliza, rematando a bola para marcar gols e tentando defender os remates do adversário. Tornando este jogo numa experiência partilhada, pretendemos criar e fortalecer laços entre pessoas através de um método tão propício a isso, que é o divertimento.

Palavras Chave

Realidade Virtual; Óculos de Realidade Virtual; Futebol; Jogo Social
## Contents

1 **Introduction** ........................................... 1  
   1.1 Motivation .................................................. 3  
   1.2 Objectives .................................................. 4  
   1.3 Document Structure ........................................ 4  

2 **Related Work** ........................................... 5  
   2.1 What is Virtual Reality (VR) .................................. 7  
   2.2 History of VR ............................................... 8  
   2.3 Exploring VR ................................................ 10  
      2.3.1 Detection ............................................... 10  
      2.3.2 Visualisation .......................................... 11  
   2.4 VR Games .................................................. 21  
   2.5 Social VR .................................................. 22  
      2.5.1 Social Games ........................................ 23  
      2.5.2 Social Rooms .......................................... 24  
   2.6 Current VR Problems ....................................... 25  
      2.6.1 Industry ............................................... 25  
      2.6.2 Content Quality ...................................... 27  
      2.6.3 Mentality ............................................ 28  
   2.7 Summary .................................................. 29  

3 **Solution Architecture** .................................. 31  
   3.1 Game Design ................................................ 33  
   3.2 Inspirations ............................................... 34  
   3.3 Technology ................................................ 35  
      3.3.1 Hardware ............................................... 35  
      3.3.2 Software .............................................. 36  
   3.4 Architecture ............................................... 37  
   3.5 Summary .................................................. 37
# List of Figures

2.1 Video Games revenue evolution by year ........................................... 7  
2.2 First Head-Mounted Displays (HMDs) ............................................. 9  
2.3 Head-Mounted Displays (HMDs) from the 90’s ................................. 10  
2.4 Tethered HMDs ............................................................................. 15  
2.5 Smartphone HMDs ........................................................................ 17  
2.6 Standalone HMDs .......................................................................... 18  
2.7 Future releases .............................................................................. 20  

3.1 Players’ position in game ................................................................. 34  
3.2 Disc Arena Setup .......................................................................... 35  
3.3 Design Architecture ....................................................................... 37  

4.1 Players and pitch models ................................................................. 43  
4.2 KinectManager and AvatarController components ........................... 45  
4.3 Lobby Scene .................................................................................. 48  
4.4 Network Solution Design with Kinect Components ........................... 51  
4.5 Inside Player Head .......................................................................... 53  
4.6 View Frustum ................................................................................. 54  
4.7 Composition of different colliders ................................................. 57
List of Tables

2.1 HMDs specifications comparison ........................................... 21
Listings

4.1 AvatarController registry .................................................. 49
4.2 FloorOffset node position update ........................................ 55
4.3 Disable Positional Tracking ............................................... 56
4.4 OnCollisionEnter custom implementation ............................. 57
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VR</td>
<td>Virtual Reality</td>
</tr>
<tr>
<td>AR</td>
<td>Augmented Reality</td>
</tr>
<tr>
<td>HMD</td>
<td>Head-Mounted Display</td>
</tr>
<tr>
<td>HD</td>
<td>High Definition</td>
</tr>
<tr>
<td>FoV</td>
<td>Field of View</td>
</tr>
<tr>
<td>DoF</td>
<td>Degrees of Freedom</td>
</tr>
<tr>
<td>HLAPI</td>
<td>High Level API</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
</tbody>
</table>
## Introduction

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Motivation</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Objectives</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Document Structure</td>
<td>4</td>
</tr>
</tbody>
</table>
In the last decades the video game industry has registered a huge growth. Through the years, not only have game platforms become more powerful and capable of supporting better games, but they have also evolved in the way video games are made available to the general public. Video games are now everywhere, not only on computers and video game consoles but also smartphones, smart TVs and other equipment.

The latest development has been Virtual Reality (VR). VR games try to take the gaming level one step forward by putting the player in virtual world of the game, creating endless possibilities to be explored by developers’ creativity.

This work will only focus on the gaming side of VR, but many experiments can be made to explore the potential of this technology in applications such as virtual prototyping, simulators and training, telepresence and teleoperation [1], for different areas like Education, Healthcare, Military, and other forms of Entertainment.

1.1 Motivation

Today’s VR is still taking its first steps. After an unsuccessful “VR fever” in the 90’s the technology is now reviving under new studies, developments and hardware capabilities, and the first generation of commercial Head-Mounted Displays (HMDs) is already in the market. Since 2012, with the Kickstarter campaign for the Oculus Rift, VR has started a renaissance from its failure in the 90’s. Different companies started developing their products, and most arrived to the market by 2016. The expectations were big, and although its market has continuously grown, it failed to achieve such numbers that would undoubtedly make it considered as a success [2] [3]. Soon analysts and media started forecasting another death for VR.

Opinions still diverge, but with the continuous investment in the technology by some of the main companies in the area, we believe this time VR is here to stay. Nevertheless, only hardware isn’t enough, and so it is fundamental that new innovative games and experiences continue to be developed.

By nature VR is a more personal experience. When people put the headset on they get separated from the ones around. That may be fine for a while, but after the initial phase it won’t be enough. In general humans have the need to socialize, and that doesn’t change by strapping on a VR headset [4]. We all enjoy sharing the things we enjoy with the people we like. That is one of the strengths of social media, how easily we can share things with others. Individual games have the potential of creating “more real” experiences, because it is only the player and the virtual world, with no outer interference. But from being real they can start being lonely. That is why VR social experiences can play an important role in the success of VR.

VR social experiences, either in a multiplayer game with strangers, or in a social chat with a group of
friends in a virtual environment, have the ability of transporting the real world connections to the virtual world, simulating not only our presence, but other’s as well.

1.2 Objectives

This thesis has the goal of exploring the creation of a shared experience within VR, more specifically a game to be played between two people.

We want to study the technology available today and test the difficulties beneath the creation of a game with those technologies, investigate some of the common problems and explore VR as a way of interaction and communication, instead of giving people isolated experiences.

1.3 Document Structure

The rest of this thesis is organized as follows: Chapter 2 introduces the technology, the different types of HMDs and existing components that can be integrated. It also names a few of the most successful VR games and experiences, as well as the problems VR is facing now. Chapter 3 describes how we projected our solution to be in terms of hardware and software. Chapter 4 explains the implementation process, the difficulties we faced and how we managed to resolve them. Chapter 5 shows our evaluation methodology and finally Chapter 6 summarizes the main points of the project and the future work.
Related Work

Contents

2.1 What is VR ......................................................... 7
2.2 History of VR ..................................................... 8
2.3 Exploring VR .................................................... 10
2.4 VR Games ....................................................... 21
2.5 Social VR ......................................................... 22
2.6 Current VR Problems ........................................... 25
2.7 Summary ......................................................... 29
Commercial video games started with the arcades, around 1970. In the following years, the world saw the appearance of the first home video game consoles, personal computers, handheld game consoles and even the first networked game [5].

Like any other, this industry had its ups and downs as we can see from Figure 2.1, but its growth is undeniable, with a record of almost $140B last year. Nowadays this is led by the mobile industry, but we can see the VR starting to appear.

In this Chapter we will explain what is VR and present a brief history of the technology. Then we can see its evolution into today’s technology, focusing first on the different types of HMDs and detection hardware. We will go on to explore the different genres of games and social experiences that are being made for VR. Finally we will dive into the problems VR needs to face.

![Figure 2.1: Video Games revenue evolution by year](source: Pelham Smithers)

### 2.1 What is VR

The term VR, as we use it today, consists of replacing our reality, our real world, with a new simulated environment, that from our perspective appears as real.

We are aware of the world around us through our senses: sight, taste, smell, hearing and touch. This means that our experience of reality is a combination of the information given by our senses, and how our brain processes that information. That said, what if we present our senses with made-up information? The way we see our world would also change as a response to it [6]. This is what VR is all about, tricking
the senses with “false” information in order to make us perceive another reality.

In a more technical term, VR is a 3D computer generated environment presented to the user in a way that makes it feel and appear like a real environment, with which users can interact [7].

To interact with a virtual world, the user can wear special goggles, called HMDs. These goggles have either one or two High Definition (HD) screens through which the user sees the Virtual World. To date, the first generation of HMDs are already in the market, available from different brands, specifications and purposes, and they will be covered in chapter 2.

Being VR related to senses, many HMDs also come with earphones integrated with the visual display. Although least frequent, special gloves or hand trackers can also be used. This way, the computer controls three of our five senses, making the VR experience more real than ever before.

2.2 History of VR

As said before, our notion of the world is given by the senses. When talking about VR it is pretty obvious that sight plays the key role in “teleporting” us to the virtual world. Over the years different techniques were used to create that illusion.

The first attempts date to the 19th century with 360 degrees paintings, creating images like today’s panoramic photos. Years later, a research from Charles Wheatstone showed that our brain is capable of creating three-dimensional objects from two-dimensional (stereoscopic) images. The object used for this, the Stereoscope, is the base principle of some of today’s VR headsets [8].

Through the 20th century some other forms of immersion appeared like the Sensorama, in 1950, an arcade-style machine that would simulate all five senses. But it was already in the 60’s that the technology we see (re)appearing today started to be born, the HMD. The Telesphere Mask, created in 1960 by Morton Heilig (the same inventor of the Sensorama) [9], was the first example of a HMD (fig. 2.2(a)). It provided stereoscopic 3D and stereo sound, but lacked any motion tracking and interactivity. One year later it was time for Headsight, by Philco Corporation, considered the forerunner of current HMDs (fig. 2.2(b)). It provided one screen for each eye, and was capable of tracking head movements using a magnetic motion tracking system. These head movements would allow for the user to move a remote camera in another room and look around the environment. It was designed to create immersive experiences for military purposes. However, it still lacked computer generated images [8]. This goal was achieved in 1968, when Ivan Sutherland and his student Bob Sproull created the Sword of Damocles, the first Virtual Reality (VR) / Augmented Reality (AR) HMD. This display was connected to a computer instead of a camera (hence the first VR display), which generated wireframe rooms and objects. As the perspective showed to the user depended on his gaze [10], it was already able to track the user’s head. Besides the graphics, the User Interface was also very primitive, and the headset was so heavy that had
Despite all the development of this technology, it was only in 1987 that the term “Virtual Reality” started to be used to describe this area. The “inventor” was Jaron Lanier, founder of VPL Research, one of the first companies to develop and sell VR products. These first products were the DataGlove and the EyePhone. The DataGlove started as an input system, like the keyboard and mouse, and was later used for VR. It had fiber-optic sensors which allowed tracking of hand movements and finger bending. The EyePhone was the first commercial HMD, and used LCD displays. However, the image had really low resolution (360x240 pixels) and the display was very expensive ($11,000) and heavy (2.4Kg) [11]. Regardless of these drawbacks, it was an important mark in the history of VR, and in 1992 was the inspiration for the “The Lawnmower Man”, a movie partly based on Jaron Lanier which used real VPL Research equipment. Other films succeeded in making so many people enthusiastic with VR, with “The Matrix”(1999) having a huge cultural impact.

Despite the growing popularity of VR, there was still a gap between fiction and reality, and the 90’s saw some big flops regarding VR technology. Both Sega VR headset (for Sega Mega Drive console in 1993), and Nintendo Virtual Boy (1995) (fig. 2.3) failed to achieve their objectives [8], with reports saying that Sega VR was never released because it was making people sick and giving them headaches [12], due to the limitations of hardware and rendering capabilities at the time.

The 21st century is presenting us with even bigger developments in computer technology. Everywhere we look we see HD displays, graphics are more realistic than ever and motion tracking is already easy to get. Taking all these factors into consideration we can say that for the first time we have all the necessary tools to turn VR into everyday reality.
Exploring VR

Virtual Reality comprises a collection of technologies: 3D displays, motion tracking hardware, input devices, software frameworks, and development tools. [13, p. 7].

Software and development tools play an important role in creating immersive virtual experiences, and without them it would be impossible to have VR. But if we consider only the experience itself and leave out the development, there are two main components of VR: the stereoscopic displays, or HMDs, and the motion tracking hardware. These are the components that make it possible for us to actually see the simulated environment and explore it.

From a functional point of view, we can also divide VR into two categories: detection and visualisation. Detection is the input of the system, in which the tracking system is included. Visualisation is the output, and correspond to what the user sees through the HMD display.

2.3.1 Detection

In the real world, the things we see, hear and touch are a direct mapping from the world itself. When talking about VR this is not true anymore, as we don’t see what is in front of us. However, the simulated environment has to be aware of changes in the real world, at least the user’s head, because a head movement requires a change in the virtual world. If other real world objects are being mapped to that virtual world, for example a stick working as a sword, they have to be tracked too, otherwise the experience will not correspond to what is expected to happen. This is one of the key factors to cause the user a sense of total immersion.

When designing a virtual experience, the way the world is graphically presented to the user and the graphics quality are very important (it is actually one of the reasons why VR failed in the 90’s). But if we really want the user to be fully immersed, to be present [14], it is crucial to turn real world actions and movements into virtual ones. If the user looks left, the “virtual user” has to look left and the image has
to reflect that. When this take some time to happen, or does not happen at all, it causes confusion and may lead to motion sickness.

A human detects real world changes through senses; the system does it through sensors. All HMDs have incorporated sensors with the purpose of tracking head movements. The tracking technology used is different from headset to headset, but they all enable the user to look around in an immersive way. As we will see next, different types of HMDs provide different tracking systems. While some require laser sensors and cameras, others use incorporated gyroscopes and accelerometers to recognise movements within a 3D space. The “type” of tracking is measured by the Degrees of Freedom (DoF) that they offer, which we will cover in a while.

In addition to head tracking technology, other equipment can be used to provide more in-depth experiences. The Kinect [15], introduced by Microsoft for the Xbox 360 game console, is a widely used technology due to its low price and high quality performance. The Kinect v2 (released for Xbox One) provides an HD color camera, a depth sensor, infrared emitters and an array of microphones [16]. Being built for gaming, these sensors work together with the objective of tracking full body motion. Given Kinect’s great results, it is widely used for research and academic purposes. If integrated with an HMD, it would make it possible to track the entire player and not only head movements. This enables features like the user being able to see his hands or feet in-game, without the use of hand tracker controllers.

Although the Kinect is able to track full body skeletons, it does not go to the detail of representing each finger. Even HMD’s hand controllers, like Oculus Touch, are not designed to do that. For this type of interaction sensors capable of detecting each finger are needed. Some possible solutions include VR Gloves like Gloveone [17] or sensors like Leap Motion [18]. While both can make the user visualize his own hands in a VR application, Gloveone makes it possible to “touch” virtual objects, sense weight or even differentiate textures. They are wireless and connect via Bluetooth. Leap Motion on the other hand, does not require any wearable device neither hand contact or touching. It is placed on top of a table, connected to the computer and works based on two cameras and three infrared LEDs. When used for VR it is also common to secure the Leap Motion in the front of the HMD, so that the experience is not restricted to the table area.

Some HMDs also ship with their own hand controllers. They provide motion tracking as well as some buttons/joysticks for specific purposes. The main advantage is that those devices are already integrated with the technology.

2.3.2 Visualisation

Regarding HMDs we can divide them into three types: tethered, smartphone and standalone. The first, like HTC Vive, has its own display and works connected to a computer through a cable. The smartphone type, like Samsung VR, uses smartphones as the HMD display. Finally, the standalone type, like Oculus
Go works by itself, not needing a computer nor a smartphone.

All have pros and cons. The tethered ones have the advantage of having greater computing power and much better life-like graphics. However, as they have to be wired to a computer, portability or walking through the room is not an option. The smartphone type is the exact opposite, while being wireless and portable, the rendering capability cannot be compared to a computer, and graphics are more rudimentary. In between them are the standalone, which offers a VR experience more powerful than smartphone VR but less powerful than tethered VR [19].

Another difference between these types is the technology they have, use or rely on. While the first type described uses cameras and sensors to track movements, and it is up to the computer to process that information, the second one works based on the smartphone sensors like gyroscope and accelerometer. The third type has all the processors and sensors incorporated like a smartphone, but can dedicate all the processing power to its end. Another advantage of standalone headsets is the cost, it doesn’t need an expensive computer nor smartphone.

We will present some of the most famous HMDs in each category and the main differences between them [20] [21], but before that let us clarify the Degrees of Freedom (DoF) concept.

Depending on the technology used for tracking, the system will get more or less information regarding the user’s head [22]. In the case of Smartphone HMDs the gyroscope and accelerometer can only tell about head rotations on each axis. Systems with this limited information are said to have only 3 DoF (orientational tracking or rotational movement). When it comes to Tethered HMDs, these are based on the system cameras, lasers, infra-reds or other technology used, and so, they can also tell the head position and movements (translations). Adding that to the rotation, they are able to get 6 DoF (positional tracking or rotational and translation movement). There is a third category on which the headset has a 6 DoF but the hand controller(s) only have 3 DoF. Most Standalone HMDs belong to this category, but as will be seen next, there are exceptions, with Oculus Go only having 3 DoF and Oculus Quest having 6 DoF.

Regarding positional tracking it can be done in two different ways [23]:

**Outside-in tracking** is when the head position is tracked by external cameras or sensors, within a limited area (room-scale). To expand that area more sensors are needed. Typical Tethered headsets use this external tracking.

**Inside-out tracking** is when the headset tracks its own position with the help of its front facing cameras. Standalone headsets use this technique so that they don’t depend on external equipment. It is less accurate, but gives more freedom to the user.

Next follows some of the main HMDs on each category existing today, as well as some to arrive this year. After each section a picture of the referred headsets can be seen.
• **Tethered HMDs:**

  – **Oculus Rift**
    The Oculus Rift \[24\] is probably the most famous VR headset, and the one that started all this recent investigation and development towards VR. It was founded by Palmer Luckey, had origin in a Kickstarter campaign and is developed by Oculus VR. In March 2014, the Oculus VR was acquired by Facebook for US$2 billion.
    The Oculus Rift consumer edition was released in the first quarter of 2016. Before that three major developer versions were released, code-named DK1, DK2 and Crescent Bay. These versions allowed developers to start testing functionalities of the Rift and exploring ideas.
    The Rift works connected to a computer through two cables: video is sent via an HDMI cable, power and data use an USB. It features two OLED displays, each with a resolution of 1080x1200, performing a total of 2160x1200. A dial is also integrated, allowing it to fit any user independently from the distance between right and left eyes. Also, it was designed to be possible to be used by someone wearing glasses. It has a refresh rate of 90Hz and a 110º Field of View (FoV).
    The tracking system is achieved with a positional tracker, which is a small sensor that is connected to the computer. This sensor monitors the infrared LEDS embedded all around the headset, allowing for a 360º rotation and movement. The Rift also features a gyroscope, accelerometer and a magnetometer to accurately keep tracking of the Rift position in three dimensions. These three sensors form the Adjacent Reality Tracker.
    Regarding audio, the Rift will come with integrated removable headphones with 3D sound technology, or Head-Related Transfer Function (HRTF). This enables the user to understand from which direction a sound is coming.
    The Rift also has Oculus Touch. It is a controller that works together with the headset in order to provide an even more immersive experience. Just like the headset, it has embedded infrared LEDs that are monitored by the positional tracker, apart from giving haptic feedback. Each controller has a clickable thumbstick on top, a trigger for the forefinger and a button on the back for the other three fingers to grip.
    The Rift is compatible with PC games (Xbox has an app that allows games to be played on the Rift, but as a “cinema experience”) but on the other hand, it requires a high-end GPU.

  – **HTC Vive**
    The HTC Vive \[25\] is an HMD developed in a cooperation between HTC and Valve Corporation.
    Just like the Oculus Rift, a developer version was also available before the consumer release, in April 2016.
In many aspects the HTC Vive is similar to the Oculus Rift. Both provide a 90Hz refresh rate and a FoV of 110º. The Vive display is also OLED although with a different resolution: 1200x1080 for each eye, performing a total of 2400x1080. The headset has adjustable straps and can be used with glasses.

One of the main differences between both is the tracking system. HTC Vive’s tracking system, the Lighthouse, consists of two infrared cameras to be placed in the corners of the room. These cameras emit pulsed lasers to follow the Vive’s 70 sensors (37 on the headset). With this, users are able to move freely within the room as the two cameras avoid occlusion problems.

The controllers are wireless and powered by battery, and have a touchpad, a trigger button with two stages for the forefinger and a home button. As with the Oculus Touch, they also use haptic feedback.

One of the most anticipated aspects of the HTC Vive was the integration with the game platform Steam, but it also requires a high-end computer.

— HTC Vive Pro
The HTC Vive Pro, released 2 years after the original Vive, was an upgrade of the first version. The Vive Pro screen resolution is 2880x1600. It also features two front cameras to help with hand tracking, as well as built-in headphones.

— PlayStation VR
The PlayStation VR [26], initially known as Project Morpheus, is Sony’s VR HMD for PlayStation 4. Being specifically developed to work with PS4, it has some advantages over its competition, as much of the hardware support for the PlayStation VR already exists. The head tracking is given by PS camera. PS Move controllers can be used as hand controllers, and even the PS Dualshock 4 (the normal controller) already includes full motion tracking. It will work simply by plug and play.

It has a screen resolution of 1920x1080, a 100º FoV and achieves an impressive refresh rate of 120Hz.

— FOVE 0
The FOVE 0 [27] is an HMD that also emerged from a very successful Kickstarter campaign. It provides a screen resolution of 2560x1440 and a 100º FoV. The refresh rate is marked at 70Hz.

However, the aspect that distinguishes FOVE 0 from all other HMDs is its 120fps IR eye tracking system, with an accuracy level of less than 1 degree, making it the first eye tracking VR headset. This is surprisingly new when compared to the other headsets, as the advantages it provides are huge.
From a rendering perspective eye tracking tells the system to what point of the screen the user is looking at. With this, it can enhance rendering power where the focus is, making it more sharp, and decrease in the rest of the scene, making it more blurry. This technique is called foveated rendering and with it resources can be better managed towards the renderization of what is really important. In heavy computing technologies like the HMDs it can make the difference. In fact, this improvement not only does not have a negative impact on immersion, but it can even improve it because it simulates the behaviour of the human eye in a way that no other HMD can, making the whole experience more real [28]. It also contributes to less eye fatigue.

Another obvious advantage of eye tracking is the possibility to point at menus or shoot enemies with only your eyes.

Despite this very interesting feature, FOVE has failed to deliver a consumer version just yet.

![Tethered HMDs](image)

**Figure 2.4: Tethered HMDs**

- **Smartphone HMDs:**
  - Samsung Gear VR
    - The Samsung Gear VR [29] was born from a collaboration between Samsung and Oculus
with the purpose of bringing VR to the mobile phones.

Instead of being connected to a PC and using its GPU, memory or processor, the Gear VR uses the phone’s hardware. Unlike other smartphone’s HMDs the Gear VR only supports specific Samsung phones, starting with the Note 5, Galaxy S6 and Galaxy S6 edge. This helps to provide a more enjoyable experience as it connects to the smartphone via MicroUSB in order to offer additional hardware. The Gear VR integrates a touch pad on the side of the headset and other dedicated buttons which makes navigation in menus easier, as well as containing its own custom Inertial Measurement Unit for rotational tracking, specially calibrated for the Gear VR, which allows for more smoothness when compared to its phone IMU. Its content library is also powered by Oculus.

Regarding specifications, the Gear VR screen resolution is given by the smartphone’s display and it has a FoV of 96º.

– Cardboard

The Google Cardboard [30] was introduced in 2014 and is an HMD made of cardboard that works with smartphones. Simple as that. You just have to place the smartphone into the Cardboard and then strap it to your head. This simple trick actually works, as today’s smartphones already contain all the necessary sensors to accurately track head movements, and while not providing the same level of immersiveness as “normal” HMDs. It is targeted at people that wish to try VR for the first time without having to spend a lot of money.

Google made the design of the Cardboard open to anyone (it is possible to download the instructions and build one at home), so there are manufactures that provide some other versions of the Cardboard that are made from more durable materials.

– Google Daydream View

The Daydream View [31] is Google’s second VR platform, after the Cardboard. It is similar to Samsung Gear VR, in that it uses a smartphone and not all Android devices are compatible. It is made from lightweight fabric (instead of cardboard) and comes with a two-button small controller that interacts with the virtual world through waving or button pressing for predefined actions.

The second version, which introduced a few improvements like wider FoV, was launched in 2017.

– Zeiss VR One Plus

The Zeiss VR One Plus [32] is one kind of durable material Cardboard HMD. It is similar to the Samsung Gear VR, except that it accepts any Android or iOS smartphone between 4.7 and 5.5 inches.
It is made from plastic and being Zeiss a world leader in the manufacturing of optical systems, it provides one of the clearest HMD lenses, that does not get blurry or foggy thanks to several ventilation slots. The VR One also has a transparent front shield allowing the use of smartphone cameras for AR apps.

In order to support the large variety of smartphones, Zeiss has created custom trays for several different models that can be bought separately.

Screen resolution is completely phone dependent, but the Zeiss top quality lenses offer a FoV of 100°.

![Samsung Gear VR](image1) ![Google Cardboard](image2)

![Google Daydream View 2](image3) ![Zeiss VR One Plus](image4)

**Figure 2.5: Smartphone HMDs**

- **Standalone HMDs:**
  - **Oculus Go**

The Oculus Go [33] is Oculus first Standalone headset. After an initial setup with the help of the smartphone app, it becomes totally independent from PC, consoles or mobiles. It has orientational tracking, meaning a 3 DoF, the same as its controllers. The Oculus Go is equipped with a Snapdragon 821 processor and has an OLED display with a 2560x1440
resolution. It has a refresh rate in between 60Hz and 72Hz and a FoV of 100º. It provides open-ear audio instead of headphones.

- **Lenovo Mirage Solo**
  The Lenovo Mirage Solo [34] is a collaboration between Google and Lenovo, running on top of Google’s Daydream platform.

  Although similar to the Oculus Go, it is more powerful thanks to a better processor and tracking system. It has two front facing cameras that track the space around, providing a 6 DoF, and single controller with 3 DoF. Regarding specifications it is equipped with a Snapdragon 835 processor, a LCD display 2560x1440, 75Hz refresh rate and a 110º FoV.

- **HTC Vive Focus**
  The Vive Focus [35] originally was only launched in China. With integrated audio, OLED 2880x1600 display, 110º FoV, 75Hz of refresh rate and a Snapdragon 835 processor it marks its place on the top Standalone HMDs. However, HTC is aiming it more towards enterprises rather than consumers.

![Figure 2.6: Standalone HMDs](image)

**Future Releases:**

- **Oculus Quest**
  The Oculus Quest [36], yet to be released in the Spring of 2019, is Oculus second Standalone headset. The Quest is a much more powerful Standalone than its predecessor Go.

  With 4 sensors in its corners and a new tracking technology called Oculus Insight, the Quest provides a 6 DoF movement, also adding new controllers like the Oculus Touch with 6 DoF too [37]. The rest of the specs confirm this as one of the most anticipated Standalone headsets, namely an OLED display with 2880x1600 screen resolution, 72Hz refresh rate, an unofficial 100º FoV and Snapdragon 835 processor. Also, the well received open-ear audio from Go is still present.
– Oculus Rift S
The Oculus Rift S [38] is the new upcoming version of the original Rift. To be released also in the Spring of 2019, it is a tethered headset with some significant upgrades on the Rift. The screen is LCD with 2560x1440 resolution. The refresh rate is target at 80Hz, and the FoV is still unknown, but said to be slightly larger than the Rift [39]. Regarding tracking, it will use the same technology as the Quest, the Oculus Insight, using its 5 cameras. The controllers are also the same as the ones for Quest.

– HTC Vive Pro Eye
The Vive Pro Eye [40] will be the next headset to be released by HTC in Q2 of 2019. It is much more similar to the Vive Pro on all specs, but with one major difference: Eye tracking. Like the FOVE 0, the Vive Pro Eye provides the tracking of the user's eye, allowing foveated rendering. The price is yet to be announced, but it will most likely be target at enterprises like the Vive Focus.

– HTC Vive Cosmos
The Vive Cosmos [41] is HTC's bet for the consumer market. With a lot of new characteristics, it will undoubtedly be one of the most powerful tethered HMDs in the market. While some information has not been officially confirmed, this may be more than a tethered device. Without disclosing too much, HTC hinted that this headset may be capable of connecting with other devices too. The tracking system is also different from regular tethered headsets, as it is now inside out, given by the four cameras around the device. This means a 6 DoF positional tracking. Another new feature is the possibility to flip up the screen at any time, in order to see the real world. The screen specifications have not been announced yet, but will be even better than the one on Vive Pro. The controllers are also new, with the similarity to Oculus Quest controllers capturing the attention. Finally, HTC will release a new platform called Vive Reality System, which will power the Vive Cosmos [42].

– Valve Index
The Index [43] which was the last HMD to be announced, is a Valve work separate from HTC Vive. Providing a display resolution similar to the Vive Pro, the Index stands out for having a 120Hz refresh rate, with an 144Hz experimental mode. Also the FoV is said to be 20º better than HTC Vive, being adjustable and optimized for each user. Despite having two front facing cameras it still uses two laser base stations for tracking, expandable to four, for a 10x10 tracking area. But the great innovation are the controllers. With 87 sensors each, they are able to track finger position and pressure, besides the hand position and motion. This can tell how hard the user squeezes the controllers and opens possibilities for natural grabbing, dropping and throwing movements.
Figure 2.7: Future releases
Table 2.1 presents a resume of the information above.

<table>
<thead>
<tr>
<th>Headset</th>
<th>Type</th>
<th>Resolution</th>
<th>Refresh Rate</th>
<th>FoV</th>
<th>DoF</th>
<th>Controller DoF</th>
<th>Other Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oculus Rift</td>
<td>Tethered</td>
<td>2160x1200</td>
<td>90Hz</td>
<td>110º</td>
<td>6</td>
<td>6</td>
<td>Constellation Camera Tracking</td>
</tr>
<tr>
<td>HTC Vive</td>
<td>Tethered</td>
<td>2400x1080</td>
<td>90Hz</td>
<td>110º</td>
<td>6</td>
<td>6</td>
<td>Lighthouse Tracking</td>
</tr>
<tr>
<td>HTC Vive Pro</td>
<td>Tethered</td>
<td>2880x1600</td>
<td>90Hz</td>
<td>110º</td>
<td>6</td>
<td>6</td>
<td>Lighthouse Tracking</td>
</tr>
<tr>
<td>PlayStation VR</td>
<td>Tethered</td>
<td>1920x1080</td>
<td>120Hz</td>
<td>100º</td>
<td>6</td>
<td>6</td>
<td>PlayStation Camera Tracking</td>
</tr>
<tr>
<td>FOVE 0</td>
<td>Tethered</td>
<td>2560x1440</td>
<td>70Hz</td>
<td>100º</td>
<td>?</td>
<td>-</td>
<td>Eye tracking</td>
</tr>
<tr>
<td>Samsung Gear VR</td>
<td>Smartphone</td>
<td>Phone Dependent</td>
<td>Phone Dependent</td>
<td>60º</td>
<td>3</td>
<td>-</td>
<td>Samsung devices only</td>
</tr>
<tr>
<td>Google Cardboard</td>
<td>Smartphone</td>
<td>Phone Dependent</td>
<td>Phone Dependent</td>
<td>96º</td>
<td>3</td>
<td>-</td>
<td>Very low cost</td>
</tr>
<tr>
<td>Daydream View 2</td>
<td>Smartphone</td>
<td>Phone Dependent</td>
<td>Phone Dependent</td>
<td>100º</td>
<td>3</td>
<td>-</td>
<td>Android specific devices</td>
</tr>
<tr>
<td>Zeiss VR One Plus</td>
<td>Smartphone</td>
<td>Phone Dependent</td>
<td>Phone Dependent</td>
<td>100º</td>
<td>3</td>
<td>-</td>
<td>Any 4.7 to 5.5 inches device</td>
</tr>
<tr>
<td>Oculus Go</td>
<td>Standalone</td>
<td>2560x1440</td>
<td>60Hz-72Hz</td>
<td>100º</td>
<td>3</td>
<td>3</td>
<td>Snapdragon 821 Open ear audio</td>
</tr>
<tr>
<td>Lenovo Mirage Solo</td>
<td>Standalone</td>
<td>2560x1440</td>
<td>75Hz</td>
<td>110º</td>
<td>6</td>
<td>3</td>
<td>Snapdragon 835</td>
</tr>
<tr>
<td>HTC Vive Focus</td>
<td>Standalone</td>
<td>2880x1600</td>
<td>75Hz</td>
<td>110º</td>
<td>6</td>
<td>3</td>
<td>Snapdragon 835 Aimed at enterprise</td>
</tr>
<tr>
<td>Oculus Quest</td>
<td>Standalone</td>
<td>2880x1600</td>
<td>72Hz</td>
<td>100º</td>
<td>?</td>
<td>6</td>
<td>Snapdragon 835 Oculus Insight</td>
</tr>
<tr>
<td>Oculus Rift S</td>
<td>Tethered</td>
<td>2560x1440</td>
<td>80Hz</td>
<td>?</td>
<td>6</td>
<td>6</td>
<td>Oculus Insight</td>
</tr>
<tr>
<td>HTC Vive Pro Eye</td>
<td>Tethered</td>
<td>2880x1600</td>
<td>90Hz</td>
<td>110º</td>
<td>6</td>
<td>6</td>
<td>Eye tracking</td>
</tr>
<tr>
<td>HTC Vive Cosmos</td>
<td>Tethered</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>6</td>
<td>6</td>
<td>Flip Screen Other devices?</td>
</tr>
<tr>
<td>Valve Index</td>
<td>Tethered</td>
<td>2880x1600</td>
<td>120Hz</td>
<td>130º</td>
<td>?</td>
<td>6</td>
<td>Finger Tracking Controllers</td>
</tr>
</tbody>
</table>

Table 2.1: HMDs specifications comparison

2.4 VR Games

There are a lot of VR games out there for all the different platforms and headsets. Some are platform specific and are only available on that headset’s store/library, while others are cross platform and can even be played between players using different headsets.

It would be both impossible and pointless to try and name them all, so we conducted some research to discover the most acclaimed ones by the critique. As opinions are personal, we only selected games that we saw referenced across several pages [44] [45] [46] [47] [48].

Some already existing genres of games fit perfectly in VR like first person shooting or car racing.
These games already provided a first person camera, usually controlled with a mouse or joystick, that is now controlled by the player’s headset. Given this fast iteration from “regular” game into VR game, many shooting and horror games were among the first games to appear in the VR market. However some had doubtful quality, which led to some disbelief towards VR. This picture is now changing, with good games starting to be released. To name some, Fallout 4 VR and Robo Recall are two good shooter examples. Resident Evil 7: Biohazard VR is a good mix between shooting and horror and said to be even better than the non-VR version. The Elder Scrolls V: Skyrim VR although not being a shooter is also a great action example. Regarding racing, Project Cars 2 has very good reviews.

Other types of games that work very well with VR are exploratory games. Either through the eyes of a bird, on board a spaceship or diving deep into the ocean, the ease of looking around inherent to VR makes it possible to create stunning immersive experiences, as in Eagle Flight.

However, VR is not just about creating games in first person view. An interesting approach to adventure / platform games is a third person view. In these games the user still controls the player, but not as if he is the player. Being VR it is still possible to look around the scene and get the same immersiveness of first person games, but with the ability to also see the player itself. Moss, Edge of Nowhere and Astro Bot: Rescue Mission fit this category very well.

Rhythmic and fast moving games are also proving to be a good bet. These “Wii type” of games were always engaging and fun, but with VR they gain a new life. Also, in some cases, they can even be considered as some exercise for its rapid movements. In this category Sprint Vector is a good representative and Beat Saber is a must have.

Sports games are always one of the main categories on all platforms. In VR things are a little different. Typical popular sport games like football or basketball exist in VR, but most times they are adapted to some kind of single-player game, and not a full team control game like FIFA or NBA. Regarding VR, some single sports tend to be better at creating the desired immersion. Some examples are The Climb or BoxVR. Some non-real sports are also very popular, with Echo Arena and Sparc being to good examples.

Finally, we want to mention some puzzle categories. Games like The Talos Principle VR and Hitman Go: VR Edition are entertaining games, without much action and fast movements which makes them easier for VR beginners to start playing with it.

2.5 Social VR

Social networks are not a consensual subject. While it is undeniable they have the power to break the distance between people and bring them together, it is also true that many times they also create a distance for the people near and the world around us, given the time spent looking at the smartphone.
When Facebook acquired Oculus, Mark Zuckerberg said VR “will be the most social platform ever” [49]. But that was not the case in the first years of VR, with games being mostly isolated experiences [50]. In fact, it is unlikely for it to happen in the next few years, but companies have already started realising the importance of making VR a social and shared experience. While at first glance VR can look like a disconnected, isolating experience, it has the potential to connect people in brilliant new worlds. [4]

Being a social platform doesn’t mean exclusively being a social network. Although social networks can exist inside VR, making VR social is more than that, it is about sharing an experience with someone simultaneously rather than sharing some content. That experience can come in various forms; be it a movie, a game or a 360º picture.

### 2.5.1 Social Games

There are already some multiplayer games for VR, but not all provide the same kind of interaction between the people playing them. The games we want to point out here are those that involve direct social interaction between players. These games can either be played with friends or strangers, and they emphasize how VR can help strengthen connections. One clear example is Catan VR, based on the worldwide popular board game The Settlers of Catan. This VR game attempts to mimic the original game by putting the players, each one represented by a different avatar, around a virtual table with the board in the middle of them. A perfect example of how these experiences can positively affect our relationships is described by Nicole Lee, Senior Editor at Engadget, games and technology reviewer, in her review of this game entitled “Catan VR gets closer to the real thing than any app. The key: Social interaction”. After presenting her views on the game itself, Lee finishes the review by saying [51]

> One of the reasons I love board games so much, is that it is often a backdrop for social interaction, where people can come together, laugh, and joke around, while playing a game. A strict app-only experience, while still enjoyable, does away with the social part. But in VR, suddenly that social aspect is back. Now (...) I can play Catan with friends and family who live across the globe, and still capture that same feeling of camaraderie that I would have when playing the game in person.

Maybe board games can more easily touch that side of people because of remembrances of the past, but there are more types of games capable of exploring social interactions. Another example is the game Star Trek: Bridge Crew on which up to four players team up in order to complete missions. Each player has a specific role and they must communicate with each other so they can succeed. Given its nature, cooperative games are more likely to stand out in the interaction component.

Another example we would like to point out is Keep Talking And Nobody Explodes. One player, the Defuser, has a bomb in his hands, but needs the help of the others to defuse it because it doesn’t have
the bomb defusal manual. The other players, the Experts, have the manual but can’t see the bomb. This
game has the particularity of having only the Defuser in VR, while the other are next to him, consulting
the manual and all together trying to stop the bomb from exploding.

Finally, *Sports Bar VR 2.0* puts the players hanging out in a bar where they can chat and play typical
bar sports like darts or pool.

In a slightly different approach, one other interesting use of VR is to gather people around the same
physical real world space and let them enjoy the same virtual experience together. The Lostroom, in
Lisbon, is a escape game that differs from all other escape games for being done in VR. The group of
people are put in the same room, each one with its headset and controllers and similarly to a normal
escape room the group has to solve all the challenges they are faced with before the time finishes.
Because it is VR, the most creative scenarios can be developed, producing an amazing experience. It is
a type of experience that was born in computer games, brought to the real world and now has returned
to the digital. Admittedly today’s technology people have to be seated, because they would bump into
each other on a room-scale experience, but maybe in the near future it will be possible.

### 2.5.2 Social Rooms

The other branch of Social VR are the social rooms. In these applications users share a virtual space,
either public or private, where they can talk, play simple games, watch movies and many other activities,
together at the same time. We are going to refer a few that have different characteristics between them.

*vTime* doesn’t try to provide a bunch of different activities for people to be entertained. It simply
involves sitting around like campfire and chatting with three other people, represented by 3D characters.
By focusing on the conversation, it is able to present wonderful pre-rendered scenarios for people to be
in and look around.

In stark contrast to this there is *Rec Room* which has total focus on the gamification of the social
interactions. Here players have at their disposal a wide variety of sport games, such as ping pong,
basketball, disc golf or paintball. The graphics are more cartoon style instead of semi-realistic 3D of
*vTime*.

*AltSpace VR* was one of the first VR experiences to appear and is among the biggest VR networks.
It has two main branches, the typical mini games with other people, and the events. These events are
what distinguishes *AltSpace VR*. They are organized on a regular basis and can involve open mic poetry,
meditation events, tabletop gaming, or even an improv night [52]. Often some celebrities also give them
shows and talks.

In *VRChat* users can build their own content to the world. This freedom of random elements make
it very popular. In terms of activities, it is equivalent to other discussed experiences, with chats, games,
movies and so on.
Being the owners of Oculus, Facebook also have their personal VR experiences. In Facebook Spaces users create their own avatar, and invite friends to join them. With the ability to visit real world places, or meet “inside” 360° photos, they chat, share pictures and Facebook videos, basically the experiences the Facebook app and website provide, but in VR. 3D drawings are also an option, with those drawings becoming virtual interactable objects. Facebook Spaces is available for Oculus Rift. There is also Oculus Rooms, available on Gear VR and Oculus Go, that is similar to Spaces, but with some added features. For example, it enables app launching and going multiplayer with the friends in your room. Oculus Rooms also let the user customise his own room, choosing the room layout, furniture, time of the day and which photos to display on the walls. One big difference between these two and the other experiences is that users can only share their space with people they are friends with on Facebook.

Many other VR Social Networks/Rooms exist, but these are some of the main ones. We won’t cover them all, as our main goal is to show the possible experiences right now in Social VR, and the rest end up having similar characteristics.

The social impact that VR can have on society has the force to be a game changer for the success of VR, attracting more users and with that more investment in research and development of headsets and content.

2.6 Current VR Problems

When I first thought about VR problems, what came to my mind was what is normally referred to as VR sickness. And while that is indeed a problem, VR problems go far beyond that. Before thinking about the problems inside VR content, we should focus on the problems inside VR industry and its adoption.

2.6.1 Industry

As we mentioned on Chapter 1 we already had an attempt at VR in the 90’s that failed. How can we know the same is not going to happen again now?

If it fails, for sure it won’t be for the same reasons. The truth is that at that time, the technology wasn’t ready for VR. Heavy headsets, low resolution screens and poor graphics put all VR hopes to an end. Right now, it is safe to say those three problems have been taken care of (pixels are still noticeable on today’s headsets, but that is not really a showstopper issue). For the last 7 years some of the biggest companies in the world have been investing millions of dollars in order to create the best HMDs we can imagine. And the reality is that VR is not near to be a failure. It still has some problems, but the idea that stands now is that maybe things are not going as fast as initially thought, but they will get there. And maybe things are not even going slowly at all. The idea that VR is under-performing comes from
the initial hype around its reappearance and from the possibly too unrealistic forecasts that were made.

On a keynote at VRLA in 2017 John Riccitiello, CEO of Unity, presented his thoughts on why the VR industry didn’t grow as fast as some analysts estimated, and the challenges yet to be overcome for that to happen [53]. One of his ideas was precisely that, unrealistic forecasts. Quoting some of his words,

One of the forecasts I read recently said that the VR/AR marketplace is going to be $164 billion three years from today. Now... the entire game industry, hardware and software—including the juggernaut that is China—is only two-thirds that size, after most of my lifetime building to that point.

Two years after, we can say that such numbers are completely exaggerated. This shows us how high some of the previsions were and why the industry wasn’t able to reach those numbers. His belief is that VR will grow, just more slowly than the industry forecasters have said [54].

Regarding the challenges, Riccitiello appointed two main aspects VR would need to solve:

**Price:** the VR experience is expensive. This is probably the number one thing preventing it to be more consensual by now. Both Rift and Vive cost over $400. Besides that, a VR-ready computer is also needed, and that can easily cost more than twice the headset value. PlayStation VR comes a little cheaper, but it also needs both the headset and the PS4 itself, making a total around $600. Even the apparently cheaper solutions like Gear VR require Samsung flagship devices.

**Usability:** tethered headsets are not very user friendly. There are USB cables, external cameras mount and calibration. A consumer product should be much simpler and portable.

The proof that these were valid concerns is the release of the first standalone headsets, at the end of that year. Standalone headsets clearly point at giving consumers a VR-ready experience out of the box. No cables and cameras are needed, it is easily packed, and because it is standalone equipment, the price is also lower. Right now standalone seems to be the future of VR gaming, but this year Oculus and HTC will be releasing new tethered headsets. And we don’t know if Sony will break their relation between VR and the consoles, so it is not easy to make big predictions about the future.

Last March Sony reported to have sold 4 million units of the PlayStation VR [55], which is estimated to be more than three times the number of Oculus Rift or HTC Vive units. If we consider the millions of PS4 sold since its launch (more than 90 million), and the number of game ready computers worldwide, these numbers are not that good. But this is a virtuous cycle. On the one hand, big game studios will only invest time and money in VR content when it has a much larger user base, so that they can have a return on the money invested. On the other hand, new users won’t be overly attracted into investing that much money in VR technology if the games are not appealing. The reasons that lead to PSVR selling the triple of its competition was the lower price and the easiness to mount and play [56]; the two characteristics we have already pointed out before. In order to break this cycle, prices will have to come
down even more, so that new people are keen on giving VR a chance.

Also, as said before, companies should keep in mind the importance of social experiences. If they want more people in VR, they’ll have to turn the platform into a space where people can play together [4].

2.6.2 Content Quality

Another often mentioned problem is the lack of good content, which is ultimately connected to the low number of users. When taking a look at the games already available it is noticeable that this technology still isn’t a priority for big studios, as most content available is created by small studios and indie developers. It doesn’t mean that they cannot create good content, their reach is just much smaller. Right now it is their responsibility to prove whether or not this time VR will be capable of taking off. Ultimately, this success will depend on the energy, time and money that big companies invest in VR, and although we are seeing it happen on the hardware side, with big names like Google, Facebook and Valve creating better headsets every year, the games will only evolve if the user base grows. However, no matter how much companies invest in creating the best HMDs, or if big studios start creating content, if the content itself has problems then VR will never be successful.

VR games have characteristics that differentiate them from the normal games. With all the greatness of VR other problems different from the usual ones also appear.

The most serious problem all VR developers have to deal with and take special care when developing applications is motion sickness, also referred to as simulation sickness or VR sickness. Motion sickness “is the condition which appears due to a mismatch between the information that the person receives through his vestibular system (inner ear and brain) and the visual data” [57]. In the case of VR games and applications, it is motion that is seen but not felt [58]. When this happens, the brain gets confused by the different signals that are being received and thinks it is because it is being poisoned. The defence mechanisms involve nausea and vomit, that everyone has already experienced on car or bus trips.

Another causing problem of motion sickness is latency. In this context, latency is the delay between the user doing some movement and that movement being seen in game. Latency for this kind of systems is measured in milliseconds, and it is advised that for VR games it should be kept under 100ms in order to avoid sickness. The refresh rate should always be above 60Hz to keep the game moving smoothly and prevent stuttering.

Many experiments have been done in order to understand how this effect can be mitigated or even completely removed, and some techniques were discovered. The first two techniques we’ll address are almost rules. When dealing with VR it is crucial that the camera doesn’t move unless the user has that intention. This means that camera animations are “forbidden”. Also, when movement does happen, it is very important not to have head wobbling (those little up and down movements typical of first person games), for the dizziness and nausea it provokes [59].
In games that the player has to walk across the virtual world, the best way to do it so that VR sickness doesn’t happen is not to walk at all. Instead, teleportation was found to be the best way to deal with the need of moving the player. Because the movement is instantaneous the brain doesn’t get confused. The downside is that it momentarily brake the immersion. However, no rules apply to everyone. Some people are more sensitive to motion sickness than others, so most times it is better to let the user choose the type of movement he wants to experience. A good alternative is to make the player walk, but without acceleration. Moving at constant speed is said to really help reduce motion sickness. Another trick to add when walking is to reduce the FoV of the camera, because fewer objects are seen in movement by peripheral vision.

Still, sometimes there is the need to move the camera yourself and break the first “rule”, like scene transitions for getting into a vehicle, falling on/getting up off the ground. Those animations involve too many head movements, more than enough to make the user dizzy. And in fact, the really important thing is the state, either inside the car or outside the car, all the in between is not needed. In those situations a common approach is to create fade through black or blink transitions. Tom Forsyth, former Software Architect at Oculus, who was involved in the creation of the SDK, perfectly described some of the referred techniques at the first edition of Oculus Connect conference, in 2014 [60].

The setup of the environment also plays a key role in avoiding motion sickness. Repetitive visual patterns, like stairs and checkered floors, should be eliminated or at least smoothed or blurred. The shaders and materials used in the scene should also be appropriate to the VR environment, taking into account the different rendering capacities of a smartphone compared to a computer. Low frame rates also induce sickness, besides breaking the sense of immersion.

Apart from motion sickness, caution also has to be taken with the users’ fatigue. By their very nature VR games are much more exhaustive than normal video games. Because the screen is very near to the eyes and the headset weights on the head, playing periods should be shorter, with more breaks in between, when compared to normal games. Using very bright colors makes the eyes get tired more quickly [57]. Some consequences of getting tired in VR and long playing periods include eye soreness, trouble focusing, dizziness, disorientation and loss of spatial awareness [61].

2.6.3 Mentality

The downside of Social VR is that some people do not know how to behave. These days people are hidden behind the anonymity of the Internet; their bad actions are able to pass undetected. On social networks we see people being rude to other people all the time, simply because they can.

Unfortunately, these kind of behaviours also migrated to Social VR. In a survey of 609 people, the results of which are available online, people have reported they avoid Social VR because of the fear of harassment [62]. Giving a couple of examples, 49% of women reported having experienced at least one
instance of sexual harassment; 30% of male respondents reported racist or homophobic comments.

Given the importance of Social VR, harassment and fear of harassment are big obstacles to the growth of VR as a whole. We will have to find ways to flag these kind of people, devising systems that can keep bad behaviour out [63].

### 2.7 Summary

We started this chapter by giving a brief explanation on what is VR, how it is related with the human senses and how we can interact with it.

Then we took a look at the History of VR, from its initial experiments with stereoscopic images, passing through the first computer generated images until the end of the 20th century, with the big disappointment over VR’s failure.

Next we went into today’s technology in more detail, describing the components of a VR system. We explained different mechanisms of tracking and then the existing categories of HMDs and the differences between them, giving examples of some of the main VR headsets on each one. Table 2.1 summarises that information.

Although VR can be used in many areas, the focus of this work is the gaming area, so we presented the VR games that are standing out the most and seem capable of attracting more people to the world of VR. We also showed how games can have a social component that can be explored to strengthen connections between people, and from there we addressed the importance of VR Social Experiences, highlighting some different possibilities.

Finally, we discussed the problems VR is facing now and will face in the future, from the overall industry and its incapability (or not) to grow the market and user base, to the most frequent issues developers have to deal with when creating VR experiences, and to the traumatizing experiences some people go through because of other’s behaviour.
3 Solution Architecture

Contents

3.1 Game Design ........................................... 33
3.2 Inspirations ............................................ 34
3.3 Technology ............................................. 35
3.4 Architecture ........................................... 37
3.5 Summary ................................................ 37
In Chapter 2 we explored the actual state of VR, focusing on the headsets, on the existing games and social experiences and on the VR problems. We finished the chapter by showing how social games can impact the VR industry, in both positive and negative ways.

For this project we thought about those VR problems and in what way we could help tackle them. We had identified those problems to be of three kinds: industry, content quality and users’ mentality. It was obvious that our effort should focus on the content, as we wouldn’t be able to act on the hardware and industry issues, related with pricing and mobility. As for the mentality problems, we could act on the creation of mechanisms to prevent and signalise bad behaviour, but for that some psychology knowledge would also be desired.

3.1 Game Design

When designing the idea of what experience to create, two points were clear: it was obviously going to be something in VR, and it would need to involve at least two people. Moreover, as the end result of this thesis is not something that needs to be deployed in the market, we would also be able to explore a little further than VR technology and create something different from the typical VR game.

What we came up with was the idea to create a football VR game to be played between two people, one against the other. Players are placed in a virtual football pitch, one on each goal as pictured on fig. 3.1, and have the objective of any football game, scoring more goals than the opponent. For that, players use their feet to kick the ball and try to score, and any part of the body to try to stop the other player from scoring. Although players are allowed to move freely they can only do it inside a certain area, as they are not supposed to move all around the pitch.

Another detail is that we don’t want to make it as a penalty shoot-out kind of game, where before each turn the ball is placed on the mark. Instead, the ball should move freely from one player to the other, except when it gets out of reach or a goal is scored. To achieve this, some boundaries and restrictions to the ball movement need to be applied.

The social part of the project is given by having a two-player game, with the users being in the same virtual and real space. It should create an interesting dynamic similar to the Lostroom escape game that we referred on Section 2.5.1. We think that the fact that they can directly talk to each other increases not only the state of presence, but also the competitiveness among them.

Although there is not a limited gameplay duration, we believe that given the freedom of movements it should not last more than three or four minutes, because it is easy to lose the notion of the real world space.
3.2 Inspirations

Having the base idea defined we went looking for some inspirations that would help to better draw the outlines of the project. The game developed is a composition of different aspects and characteristics of the following games:

Penalty Rift [64] was one of the first developed VR experiences. It was created for Oculus DK1, and consists of a penalty shootout game, but having targets on the goal instead of a goal keeper. What makes this game very interesting for our project is that the user's movements are captured using a Kinect, that animates the player avatar in game.

Goal Keeper [65] is a simple browser football game published by Miniclip in which the player has to defend incoming balls and dodge plastic bottles. As the game is played in a 1st person view, the only part of the keeper that is seen are the gloves, which are controlled with the mouse. When playing it, often the gloves cover the vision for the upcoming object while defending the previous one, and if it happens to be a bottle it is not easy to escape. This was a little frustrating and led to the thinking that this problem would not occur in a VR game.

Disc Arena [66] (later renamed Project Arena) was a project from CCP Games on which two players were put face to face playing a light-disc style of dodgeball. Initially the players’ movements was tracked with a Kinect, but those mechanics were later changed with the release of the first controllers.

Final Soccer VR [67] is another football game for VR. It has several game modes, one of them being a two-player penalty kicks. As opposed to Penalty Rift, trackers are put on the users feet.
These games allowed us to gather some interesting characteristics that we believe to be promising and able to integrate well together.

We liked how Penalty Rift was able to use the Kinect, which had been created for a different technology, to replace the need for tracking movements, as at that time controllers didn’t exist yet. As to the Goal Keeper game, it has nothing to do with VR, but the first-person camera of the game helped having the visual idea of how the keeper view would look like. The Disc Arena showed us the hardware setup necessary to make all this work. Figure 3.2 shows a photo taken at a demonstration of the game. We can see that each player has its own headset, computer and Kinect pointed at him. Finally, on Final Soccer VR, we could visualise a game that at the end of our implementation should be very similar, even though they have a big and important difference, the tracking mechanism used.

![Disc Arena Setup](image)

Figure 3.2: Disc Arena Setup

3.3 Technology

In this section we will present the technology used in the development of this project.

3.3.1 Hardware

As we saw in Chapter 2 many HMDs available today have hand controllers. With the developers creativity these controllers can be used for all kind of actions, from the most simple like picking up objects, to some
elaborated ones like acting as the player's legs and feet (e.g. Sprint Vector). However, for our game we opted for something different.

As the game involves being both the goal keeper and the striker, a complex use of the controllers would have to be designed to handle both situations. Instead, using a Kinect seems like a good option. Although we never got to try it, Penalty Rift is an example of this approach that seems to work fine.

The choice of using a Kinect for tracking narrows the available options for HMDs, because we will need a computer to receive and process Kinect's data. Therefore, a tethered headset turns out to be the possible option. From the list of tethered headsets that we resumed on table 2.1, the FOVE 0 was not released so it is out of question. The PlayStation VR is made to work with the PlayStation 4 console, requiring third-party software to be able to connect to a computer. That said, we are left with two (or three) options, the Oculus Rift and the HTC Vive (normal or Pro). When using the controllers, the HTC Vive tracking system, the Lighthouse, is more advanced and enables more complex interactions. But that is not our case, so for the experience we will build, either one or the other should be equivalent.

3.3.2 Software

Unity is the most used engine for creating games. According to John Riccitiello, on his keynote at VRLA 2017, two thirds of the world VR/AR content is built with Unity [53]. One of the reasons for that is the engine's ability to support building games to a wide variety of platforms. Also, it is easy to learn and to use, provides good documentation and has a huge community behind it, where people can search for help when facing difficulties with their applications. For all that, we chose to use Unity as our game engine.

For the HMDs we need to install the respective software. Both Oculus and Vive have their own setup installer, which installs the required runtimes and any other software necessary to be able to run applications on the headset. Unity already has built-in support for Oculus Rift, so no additional setup is required, although an Utilities plug-in is available from Oculus, with scripts and prefabs that can be helpful during the development. For HTC Vive, the process is also very simple, as the only thing needed for Unity is to download the Steam VR plug-in, from Unity Asset Store.

Regarding Kinect, first we will need Kinect SDK for Windows. Then we will need something between Kinect and Unity, and we found two applications that seem to do the work. One option is Middle VR for Unity, that promises to have compatibility with most VR systems and interaction devices, on which Kinect is included. But besides that, there isn't much more information. The other option is ZigFu, which we know to had been already used on other thesis projects.
3.4 Architecture

Having the technology chosen, it is time to show how we plan these components to interact with each other. Users will be having two different tracking systems pointing at them at the same time; the Kinect for the body movements and the HMD’s tracking cameras or lasers for head movements. Both systems will provide the Unity application with this information, which in turn makes the virtual player mimic the user movements. The first thing we need to beware of is that both systems track the user’s head, because Kinect is a full body tracker. However, we prefer to use the data provided by the HMD as it is much more accurate.

Regarding the rendering in a VR headset in Unity it is automatic as soon as VR support is enabled and the SDK chosen for the headset to use. As for the Kinect, we will create a 3D model to represent the player and use the sensor data to move its bones.

This is how each instance of the game will work locally, but then we need to have the two working remotely. They will connect to a server that will be the responsible for keeping the game state and synchronising the game objects. We have few experience in the network area, so we will use Unity’s network components as it abstracts low level tasks for an easier use. The figure below shows this architecture.

![Figure 3.3: Design Architecture](image)

3.5 Summary

In this chapter we gave an overview of how our game will be like. We started by describing the idea behind it, the type of gameplay it will have and the social component of it.

Next we presented some other games that giving their characteristics were an inspiration to the design of our game.

Then we discussed the technology to be used, the choice of hardware for the different components
of the game, as well as the software to be used during the development. We finalised by showing how the different components will interact between them.
4

Implementation

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Kinect</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Network</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Adding VR</td>
<td>52</td>
</tr>
<tr>
<td>4.4</td>
<td>Gameplay</td>
<td>56</td>
</tr>
<tr>
<td>4.5</td>
<td>Summary</td>
<td>58</td>
</tr>
</tbody>
</table>
This Chapter contains all the implementation process of the creation of the game. It explains the steps we have taken throughout the development and the challenges we faced along that way. The project implementation is available at https://bitbucket.org/BFips/rift-thesis/.

We divided the game implementation in two distinct main areas that we opted to tackle separately, the VR part and the Kinect part. We could have started with any of those, but we chose to start with the Kinect for a couple of reasons; first of all, it sounded more complex than the integration of VR in Unity. As so, we preferred to start with the bigger integration challenge during the first periods of development. Also, contrary to the use of a tethered HMD, it doesn’t require a high-end computer to develop, which means more freedom because we could work on any Windows machine.

This Chapter is divided in a way that each section represents one of our biggest challenges.

### 4.1 Kinect

The first thing we did with Kinect was installing the Kinect for Windows SDK (note that the Kinect is a Microsoft product, as so, all the development we are going to refer next is only valid for Windows). Besides the necessary libraries it also installs a set of tools, components and samples that can be useful for developers. For instance, sometimes when the player wasn’t responding to our movements we used Kinect Studio to check whether the problem was within the application we were developing or with the hardware. With Kinect Studio it is possible to turn on the Kinect and visualise all types of data from the Kinect sensor like RGB image, IR image or the skeleton mapping.

Microsoft also provides a Unity package that contains some basic scripts and examples. These examples track the user and create a skeleton representation of the person detected in front of the camera. With little work we completed our first goal of having Kinect running in Unity, as it involved only research on what software would be needed.

For our game a skeleton representation wasn’t enough, as we pretended for each player to have his own 3D body. It creates a better sense of immersion if the players are able to see their own arms and legs, as well as the opposite player. To create the 3D models for the players we used an online character generator provided by Autodesk [68]. With it we generated a 3D avatar dressed as a football player, so we could start playing with skeleton animations. We also took the chance to start thinking about the visual aspect of our game scene. It was still quite early to concern about that, but as we were working a little with 3D models we opened SketchUp, which is a program for creating 3D models, and built some tests. At the beginning we were aiming too high, importing models of real world football stadiums. Whilst that would have created an impressive experience, it was much more complex than we needed, both in terms of model complexity for manipulation and pitch/stadium sizes. So, we ended up drawing our own simple football field, with suitable measurements and a more simplistic style. Finally, we put our model...
inside the pitch, creating our game and test scene for the next parts. Figure 4.1 shows this result, with fig. 4.1(a) focusing on the player model, and fig. 4.1(b) focusing on the players positions in the game.

From there we passed to the next step, which was having our own 3D avatar being controlled by Kinect, and that turned out to be our first big challenge. We started by making a better research on the references we had regarding the use of Kinect with Unity, ZigFu and Middle VR for Unity. We had the idea of ZigFu being less powerful and complete, but nevertheless it was our first choice, because it had a bigger community and also we could try to get in contact with former students who had used it before. However, what we found was unexpectedly bad. It looked outdated and the documentation regarding Unity was obsolete. There is a GitHub entry with 10 repositories, one of them for Unity, that hasn’t been touched in almost 5 years, and at the moment the site has already disappeared. To sum up, the project is dead, so we changed our attention to Middle VR for Unity. This product is developed by a company of name Middle VR with offices in France and China, that is focused on the Professional business, developing AR and VR applications for other companies’ needs [69]. Also, they commercialise their software for developers to use. Although not being the main focus, their software for Unity also supports interaction with most interaction devices, including Kinect [70]. They provide four different licenses of the product, one of them for free (with less support and other disadvantages) and a 30 days trial for the rest. Given the few information available we were not very enthusiastic about the product, and even the free version was out of question as it would force us to have a Middle VR watermark. Prices are not publicly available and there isn’t any version to download; a small form has to be filled with company name, person details and role. Nevertheless, we though that if the product was capable of providing such an easy integration as they advertise, we should give it a try. So, we filled the form, asking for prices information and a trial version, but we never received back any answer.

While waiting for an answer we started searching for alternatives to help us on our problem. We could have tried to parse the Kinect data ourselves and map it to our avatar bones, but that would have been a very low-level task that although challenging and interesting didn’t quite match the learning objectives we had in mind for this project.

We managed to solve the problem with the help of an Unity plugin, Kinect v2 Examples with MS-SDK and Nuitrack SDK, that can be found on the Unity Asset Store. Although being paid, the developer of this package, Rumen Filkov (which has given us great help and support every time we needed) provides it for free for any student who wants to used it for education, lecturing or research, which was our case.

### 4.1.1 Kinect Plugin

This plugin contains several C# scripts, resources and over 50 demo scenes. It is very complete (much more than we needed) and has examples on how to do practically everything with the Kinect, from moving custom avatars, face tracking, clothes fitting room scenes, and so much more. For our case we
Figure 4.1: Players and pitch models
were only interested on the Avatars Demos. These demo scenes show a 3D avatar that is animated by the user movements. This is exactly what we wanted, so we investigated how it was being done, using which components, so that we could replicate that setup on our avatar.

Some components are needed for it to work. One of them is the **KinectManager** which is the main component of this package. The **KinectManager** is the responsible for controlling the sensor and getting the data streams and all other components rely on the data provided by it [71]. As we can see from fig. 4.2(a) it has many configurable settings that can be set according to the user needs. This component should be added to a persistent object in the scene. We could create an empty object for it, but we chose to

The other needed component from this package is the **AvatarController**, that transfers the Kinect-captured user motion to the humanoid model, whose component it is added to [72]. In other words, is the component responsible for making the avatar move. It is also configurable, allowing to define some aspects of the avatar movement as shown on fig. 4.2(b).

The third needed component is the **Animator** component, which is provided from Unity. It is used to assign an animation to a GameObject and typically requires a reference to an **Animator Controller** [73], which defines the animation. However, in this case, we use it without any controller because the animation is given by our movements.

After adding all of this to our scene we were able to move our arms, legs and head, and even give some little steps to both sides, back and forth.

## 4.2 Network

After having successfully completed the avatar movement, the next challenge was to create a networked demo scene on which two computers with a Kinect each would transfer the information between them so that two avatars could be moved independently from one another.

This was without any doubt the part of the project that took us the longest. We have never had any experience on creating a networked Unity application, so we had to investigate and learn the existing methods and possibilities ahead.

### 4.2.1 UNet

We started by studying the UNet, which is Unity’s multiplayer solution. Today UNet is already being deprecated and a new Network solution is being developed by Unity to gradually replace UNet over the next two to three years, but for now it is still one option available for use.

UNet provides two different Network APIs, one High Level API (HLAPI) and one Low Level API, or Network Transport API. Given our knowledge on the area and the needs for the project the HLAPI was
Figure 4.2: KinectManager and AvatarController components
more suitable, as it is not an advanced multiplayer game. With this API it is possible to:

- Control the networked state of the game using a Network Manager
- Operate “client hosted” games, where the host is also a player client
- Serialize data using a general-purpose serializer
- Send and receive network messages
- Send networked commands from clients to servers
- Make remote procedure calls (RPCs) from servers to clients
- Send networked events from servers to clients

The NetworkManager is Unity’s component responsible for managing all the network aspects of the game. Its responsibilities include controlling the loading of game scenes for all players, spawning new objects, keeping the game state and performing the matchmaking. It can be extended if there is the need for custom functionality or listening to any of the provided server or client callbacks. One already existing extension is the NetworkManagerHUD which provides a basic UI for players to join the game. In the Unity Asset Store it is also possible to find an Unity developed package called Network Lobby. Among some prefabs, scripts and resources, it has a very interesting component that is the LobbyManager. It provides a more advanced and prettier HUD and some extra properties when compared with the other referred Managers. Its purpose is to show the proper interface to allow players to join a game, display a pre-game lobby where players can change their name and see the other players who have joined the same room. When the minimum number of players have joined the room, a countdown appears and then the game starts.

When configuring the game scenes, they should have only one NetworkManager each, that can even be a persistent game object throughout all game. That object should not have any other network component, being the most common case the creation of an empty game object just for this. On all other game objects that need to be synchronised through the network we must have a NetworkIdentity component. It controls a GameObject ’s unique identity on the network, and uses that identity to make the networking system aware of that GameObject [75]. In this component we configure the authority of that game object, that is, who is the responsible for managing this game object, the server or the client. The possible values are Local Player, Server Only or none. Besides that, the moving objects should also have a NetworkTransform component, which is responsible for the synchronization of the object’s movement and rotation across the network.

In network games a Server is an instance of the game that manages several aspects of the game, which other players connect to. Each player also runs an instance of the game, and is called a Client.
With UNet the same thing happens, but one of the clients can also be the server at the same time; this “special” game instance has the name of Host. This concepts of authority and the server/client relation are all explained and well documented on Unity’s Manual [76].

Regarding our game we opted to use the LobbyManager component. To create the game, the LobbyManager requires the setup of a few things. First we need to create two scenes a pass them to the LobbyManager. The Offline Scene is the same scene on which we are configuring the Manager. It will be the loaded scene when the application starts. When the two clients are connected, the game transitions to the Online Scene, which is our scene with the pitch. Two other important properties of the lobby that need to be set are the Lobby Player Prefab and Game Player Prefab. The first is the object that represents the player while in the lobby. The second is the “real” player object, which in our case is the 3D football player avatar. Both prefabs need to have the already mentioned NetworkIdentity.

Our game has four main game objects: the pitch, the ball, the Kinect Manager and the game player(s). Because this is a network game we needed to set the right network components to each of them. The first “rule” is simple, attach a NetworkTransform to all game objects that need to move or rotate, which in our case is the ball and the player. By having a network component they also need to have NetworkIdentity to really work, as well as have their authority specified. For the player it is simple, it must have a Local Player authority, as it is a client game object. As for the ball, it is not supposed to be managed by any client, but also cannot exist only on the server, so the default state should be maintained. The pitch is a static object, so it does not need a NetworkTransform. While it is only really needed in the game scene, we choose to create it also on the lobby scene, placing it nicely as the background of the main menu and the lobby menu, as shown in fig. 4.3. Finally, the Kinect Manager object could by analogous to the pitch, as it also does not need a NetworkTransform and is only needed on the game scene, however we chose to make it a little different. The Kinect Manager passes trough an initialization process that depending on the network and the machines could take time enough so that the scene transition after the countdown end was not as smooth as we pretended. To deal with that, we instantiate the Kinect Manager in the lobby scene and enable the DontDestroyOnLoad parameter, because by default Unity destroys all game objects when transitioning from one scene to another. This way the object state is kept and it does not need to be initialised again.

With this we though we had everything in place, so we launched two instances of the game, one as a host and the other as a client, joined the same room and both players appeared when the game scene was loaded. However, none of the avatars was moving. We noticed that although each player object had its AvatarController component, the list of registered controllers on the Kinect Manager was empty. That is because it is expecting the AvatarController’s to be already on the scene when it is initialised. As in our case that does not happens, we needed to manually register both controllers when the players are being created.
Figure 4.3: Lobby Scene
void Start () {
    if (isLocalPlayer) {
        RegistPlayer(gameObject);
        return;
    }
    playerCamera.enabled = false;
}

void RegistPlayer(GameObject player) {
    AvatarController ac = player.GetComponent<AvatarController>();
    if (ac == null) {
        return;
    }
    KinectManager km = KinectManager.Instance;
    if (km == null) {
        Debug.Log("No Kinect Manager");
        return;
    }
    long userId = km.GetUserIdByIndex(ac.playerIndex);
    if (userId != 0) {
        ac.SuccessfulCalibration(userId, false);
    }
    km.refreshAvatarControllers();
}

Listing 4.1: AvatarController registry

Notice how we only call `RegistPlayer` if the condition `isLocalPlayer` is verified, and if not we disable the player camera. This is because in with UNet, there are multiple instances of the Player object, one for each client. The client needs to know which one is for “themselves” so that only that player processes input and has the camera attached [77]. This is an attribute of every network component that is set during creation. Each client will have his own player with this parameter as true because we configured the game player prefab’s authority is set to Local Player.

After fixing this, we tried again. This time we could move our player, as it was already registered, but the movements were not synchronised to the other client, that is, the player stayed still. We realised that we had made a mistake about a basic concept: that adding a `NetworkTransform` to the player was not enough, because the avatar movements are not changes in the player position, but in the bones of the skeleton. There is one network component that at first sight we though could help, the `NetworkAnimator`,

49
which allows the synchronisation of animation states for networked objects. This is what we needed, the problem is that it does that by synchronising the state and parameters from the AnimatorController, which we do not use as explained before. Therefore, the only solution we were seeing was to add a NetworkTransform to each bone and have them being synchronised one by one.

4.2.2 KinectDataServer/KinectDataClient

We put the previous solution on standby and went searching for different alternatives. Going back to the Kinect plugin we talked about in the previous section, there was a demo scene called KinectDataServer. The purpose of this demo was to show how to run a Kinect application on a non-Windows platform, for example a smartphone HMD. The (Windows) machine on which the Kinect is connected to runs an application with the KinectDataServer component, and the smartphone runs another application with the KinectDataClient, which receives the sensor data over the network (sent from the KinectDataServer) instead of receiving directly from the Kinect. Contrary to our approach, these components use UNet Low Level API.

Despite our needs being different from the demo’s purpose, we researched and made many experiments in order to explore the way these components interacted with one another, and we ended up building a solution architecture exploiting the many configurable parameters that they provide. The basic idea was to “simulate” synchronization by having four Kinect Data Clients, receiving data from two Kinect Data Server, as simplified in fig. 4.4. Each Kinect Data Server was responsible for sending the sensor data to two Kinect Data Clients, one on his local player and the other on the remote player.

While we eventually managed to make this solution work, there was a very noticeable delay between the person movements and the player movements, even on the local player, because it is receiving the data over the network. As we saw on Section 2.6.2 this is one of the main causes of VR sickness, and so we continued looking for alternatives.

4.2.3 Photon Unity Networking (PUN)

The next thing we tried was PUN. Photon is the number one network engine, and being Unity one of the top game engines, Photon have created PUN which is a Unity package for the creation of multiplayer games. Like the UNet, it takes care of all the game low level tasks, like handling connections, matchmaking and so on. It has some functional differences from UNet, for example while on UNet the game runs in Unity instances (the server or host), PUN uses dedicated servers for it, but overall features are relatively equivalent.

As PUN is used by a lot of people, and we didn’t know what we could do with it, we decided to study it a little to see if it has anything related with Avatars that could be helpful. However, we only got to
discover something similar to the *NetworkAnimator* from Unity, requiring an *AnimatorController* with the animations configuration.

### 4.2.4 Creepy Tracker

The *Creepy Tracker* is an open-source toolkit to ease prototyping with multiple commodity depth cameras by automatically selecting the best sensor to follow each person [78]. The ability to work with multiple Kinect sensors caught our attention, because our project also needs two Kinects, although not in the same way, as in our case each Kinect is only tracking one person and their data does not need to be merged. Also, it is able to transfer Kinect data through the network, used on the telepresence portal example.

Another interesting example scenario mentioned the use of Kinects for VR applications, as a way to create a large gaming area, similar to HTC Vive Lighthouse tracking system. Because of our interest on these scenarios we went to talk with Mauricio, one of the developers of the toolkit. We explained what our project was about, what we needed to do with the Kinect and questioned if the *Creepy Tracker* could be helpful for our project. After some analysis, we concluded that while it could indeed be used in some ways, it wouldn’t be the most adequate project for it to be used in, as the biggest strength of the toolkit is its ability to handle the data from multiple Kinects, which is something we do not need to do.

At the end, Mauricio suggested that the approach of assigning a *NetworkTransform* to all the bones should be simple and performant, and probably would solve the problem.
4.2.5 Network Skeleton Component

As we were about to return to the multiple NetworkTransform approach, while doing some research about it we found a thread in a Unity Forum with the title “Synchronizing an entire skeleton” [79]. There, a member of Unity was sharing a component that “synchronizes the bones of an entire skeleton. This is intended to be used by dynamic animation systems driven by user input, such as Kinect systems doing motion capture for shared VR/AR environments”. This component, that was never released with any version of Unity, answered exactly to our problem, also advertising to be much more efficient that using multiple bone NetworkTransform. Overall, following the usage instructions was easy, but we still had to pay attention to one point. For skeletons with too many bones, the amount of data could be larger than the supported limit of about 1440 bytes, which was our case. Our solution was to set the SyncLevel parameter, which tells how deep in the hierarchy should the component synchronise the bones' transforms. The deeper bones of our skeleton are the fingers of the hands, which we do not care about in this game. As so, by setting the level to sync only until the hand bone we were able to cut in the amount of data to be sent.

As a result, we were able to have each user moving its own avatar, with less latency than we had with the Kinect components.

After so many tentatives with some different technologies, it was a fulfillment to have finally achieve that goal. We were close to the solution right from the start, but it was also great that we had to explore and learn such different strategies to solve the problem.

4.3 Adding VR

Having the synchronization of skeletons complete we had two options, we could either start working on the game physics and mechanics, like being able to kick the ball, or we could start thinking about integrating VR in the project. Although strictly speaking those options were not connected, it proved to be simpler to add the VR rendering first because it was being difficult to have a notion of distance to the ball without the VR headset.

Enabling VR in Unity is a trivial task, as it is just a matter of ticking a checkbox on the player settings. But after enabling it we need to configure the game camera's properties, including its position, and by then we started having some issues to deal with. One of the problems we faced was that by placing the camera inside the player game object head we could see some interior meshes that we did not want, such as part of eyes and teeth (fig. 4.5). The other problem remits to the headset session space, and where does it consider its original position to be. We will deepen them and the found solutions.
4.3.1 Inside Head Rendering

Even before we started enabling VR on the project, we had already placed the camera inside the player’s head to get the feeling of seeing through the player’s eyes. However, we were not expecting to see anything besides the pitch, as by default Unity has the backface culling enabled, so it was a surprise to see the teeth and some other parts of the player. We realised this was not a matter of culling the back of some meshes, nor meshes having its normal vector pointing inwards; these meshes we were seeing were really inner parts of the head mesh.

Even though we knew this problem since the beginning of the development, we only started working on it by the time we enabled VR, because with normal camera rendering we easily moved the camera somewhere else, but know that was not possible as we need the camera to be the player’s eyes.

We had some possible solutions in mind on how to counter the issue, starting by adjusting the camera’s near plane. All the objects being rendered are inside the view frustum (fig. 4.6). The near plane cuts out everything that is too near the camera, while the far plane cuts everything that is too far away. The size of these planes is given by their distance to the camera, and the camera’s FoV. On our case, as we needed to exclude those meshes inside the head we needed to increase the value for the near plane.

Another alternative was to adjust the camera position, by placing it more close to the undesired meshes, so that they would get cut by the near plane. Despite both options seemed to have worked with a normal camera, with the VR headset on, the head movements and rotations showed that depending on those movements, the tracked head position by the HMD and by the Kinect can differ in a way that some artefacts would still appear. Also, because we were making changes on the near plane, it would also cut out parts we wanted to show up, like shoulders or even the hands, if placed too near the camera.

At the same time we were trying the previous solutions, it seemed quite obvious that the ideal sce-
nario would be for those meshes not to exist at all. We imported the 3D model in 3ds Max and started deleting the interior vertices. As we are not experts on modeling in general, and 3ds Max in particular, many times we ended up deleting more than we wanted, creating “holes” on the model head and having to go back with it. We noticed that some specific elements of the model did not belong to the main body mesh, and that gave us the idea of separating the head from the rest of the body into two different meshes. So, we selected all the head vertices (much more easier than trying to delete inner vertices) and than detached from the rest of the model. When importing this modified model in Unity, the model parts now showed the head apart from the rest.

Unity allows placing the scene objects on different layers, as well as choosing which layers each game camera should render. We followed this approach, creating a new layer for those objects to be ignored, like the eyes and head. Next, we removed that layer from the camera's rendered layers. This way, no matter the different movements the player makes, the head part will never be visible.

Notice that we can only do this for the local player, otherwise we would not be able to see the other player's head.

4.3.2 Tracking Origin

Our first experience with VR in this project was with an Oculus Rift DK2. Using it was very simple, as we just positioned a game camera in the player's head and it run without problems. This was in the very beginning, even before we had the network part working. In the latest stages of the project we began developing with an HTC Vive, and we noticed things were a bit different with this one. When starting the game, the camera would be positioned way above the player. We found out this behaviour exist because these two devices use different tracking origins. There are two types of tracking origin:

- **Device Tracking Origin:** at some point during the device initialization or start of the session, its
real world position is used to mark the origin of the session space. This is what happens with the Oculus Rift; the initial position of the game camera (player’s head) will match the physical position of the headset. To give an example, if the headset is on the user’s head when the game starts, it will appear Ok in game. On the other hand, if during that initialization the headset is placed on top of a table, that will be its origin. By the time the user straps it on the head the game camera position will rise the same distance between the table and the head.

- **Floor Tracking Origin**: the origin of the session space is the floor, and the game camera maps the real height of the device. This is used by the OpenVR SDK, which is used for using Unity with the Vive.

When a game camera is being used with VR it is not possible to change its position, because it is being computed all the time by the tracking system. If we need to manipulate its position or rotation, the way to go is to add it as a child of another game object, and manipulate this one instead. This is the correct way to handle the use of devices with different tracking origins.

On our game we have a 3D model of the player, and two things need to happen at the same time: the player’s feet must be touching the ground, and the game camera must be at the eyes height. Because users have different heights, but are using the same game player, which height is fixed and does not match everyone’s, only one of these restrictions was being respected if using Floor Tracking Origin. For example, if the user is taller then the player object, either we get the feet on the ground and the camera floats above the player, or stick the player’s head to the camera and the feet do not touch the floor. To accomplish both the both at the same time we needed to add a parent node to the camera, which we called *FloorOffset*. As the game player height does not change, we can attribute it to a constant and then, in runtime, calculate the difference between that height and the real user’s height. That difference is then applied to the offset node.

The following code belongs to a script running on the *FloorOffset* node and shows the implementation for this.

```csharp
private const float HEIGHT = 1.63f; //player game object height

void Update () {
    transform.position = new Vector3(transform.position.x, 
                                      HEIGHT - camera.localPosition.y, 
                                      transform.position.z);
}
```

*Listing 4.2: FloorOffset node position update*
While at first this approach seemed to work, on some kind of movements affecting not only the vertical axis, like bending, we noticed that the camera’s position would mismatch the head position. We eventually found out a method for disabling the positional tracking of the camera, leaving the sensors to only track the rotation, creating a similar behaviour to the Oculus Rift. So, instead of constantly updating the position of the FloorOffset object, we simply turn the positional tracking off when starting.

```csharp
void Start () {
    UnityEngine.VR.InputTracking.disablePositionalTracking = true;
}
```

Listing 4.3: Disable Positional Tracking

This proved to be much simpler and more adequate to our specific case, because of our need to match the camera and the 3D model, something that does not happen on most games.

With this we finalised the integration of VR in out game. It was not as straightforward as we imagined, but it allowed us to see and learn about some different VR related problems we have not thought about before.

### 4.4 Gameplay

The network synchronisation of the skeletons, for the challenges we were faced with, and the integration with VR, for the amazing new perspective it gives to the project, were for us the most interesting phases of development, besides being the base of the game. Having that completed it was time to start implementing the game itself, with the logic and physics.

We started by creating two different physics materials for two important pieces of the game, the ball and the ground. In the physics material we configure the bounciness and friction of all objects using that material. Having a general idea of how the elements should behave, then it is a matter of tweaking the values until getting a good result. On our case we knew that the ball needed to have some bounciness, otherwise it would fall on the ground and stay still, like a heavy stone. Regarding friction, the ground is supposed to represent grass, which is not as slippery as ice, so an intermediate value was good.

So that game objects can interact with each other using the physics engine, they need to have both a Rigidbody component attached to them and at least one collider. Colliders can have different shapes like a box, sphere or capsule, and we must configure its position, size and orientation. For the ball the best choice was obviously a sphere collider, while for the ground a box collider was adequate. Regarding the player body we created many capsule colliders, with different sizes, that would “envolve” all the player’s members, as seen in fig. 4.7. It is possible to ask Unity to compute a mesh collider adjusted to the
game object’s mesh, but it is much less performant. Besides, most times there is not even a noticeable
difference in behaviour between an approximation like this one and a mesh collider.

![Composition of different colliders](image)

**Figure 4.7: Composition of different colliders**

When two colliders enter the same space they trigger a collision, and at that time the physics engine
runs the calculations regarding the direction of the objects colliding, the mass of the rigidbody compo-
nents, and other factors. The return of that collision will be the proper reaction of the colliding elements.
Colliders also have a property to set them as being only triggers. We use that option when we need to
know if two game objects intersect each other, but without creating a collision. In those cases a trigger
event is launched, instead of a collision one.

There are some `OnCollision...` and `OnTrigger...` callbacks that we can listen to when there is the need
of making manual calculations. In our case we were not satisfied with the result of the player shooting
the ball, so we attached a script to the ball upon collision checks its own position and the colliding foot
position in order to know the resulting direction vector. Then it applies a force of type impulse to the ball,
depending on Force constant we have defined, the collision impulse and the calculated direction.

```csharp
void OnCollisionEnter(Collision collision) {
    if (collision.gameObject.CompareTag("Foot")) {
        Vector3 footPosition = collision.transform.position;
        Vector3 ballPositon = _transform.position;
        Vector3 direction = (ballPositon - footPosition);
        _rbody.AddForce( FORCE * collision.impulse, ForceMode.Impulse);
    }
}
```
Listing 4.4: OnCollisionEnter custom implementation

One example of use of the trigger colliders is to detect when the ball enters the goal. On this case, we attached a box collider to each goal, configured as trigger. We tried to match real football rules, where a goal is only valid if all the ball passes the goal line, so our goal collider does not actually “involve” the goal, it is positioned a little behind the goal line. When the ball touches those colliders, the event is triggered and we know that a goal occurred and we need to re-position the ball so that the game could be resumed.

We also added colliders on the walls and ceiling, creating a “cage” so that the ball could not be able to escape. At first we created one big unique box collider with the pitch size, but colliders are not “reversible”, so that was leading to the ball being always inside the collider, enabling weird collisions and behaviour. The right way to do this was to create five different colliders, on for each plane (four walls and the ceiling).

4.5 Summary

In this chapter we detailed all the challenges we face during the of the implementation of this work, and how we managed to overcome those challenges.

We started by integrating the Kinect in Unity and using it to make a 3D avatar respond to our movements.

Then, we researched how to synchronise two Kinect skeletons over the network, so that players were able to see the opponent moving. This was the part that took us the longest, because we had the need to try different approaches, either because one did not work, or was not performant enough. On the other side, it allowed us to gain a better knowledge about some network concepts.

Next we integrated VR in the game. Unity makes the integration itself be almost direct and easy for developers. We still had some challenges, with the need to use a 3D modelation tool, and understanding why the camera could be floating above the player, but overall it was simpler then the previous part.

Finally, we resumed what we have done regarding the physics engine and the game logic.
Evaluation
As the purpose of this project was entertainment and creating social interaction (instead of educational for example), the main goal we proposed to achieve was to provide users an interesting, fun and interactive experience. It was also important to test if usual problems associated with VR, like motion sickness, were avoided, otherwise users would not enjoy the time spent playing the game.

From our own experiments during the many stages of development, at first we were a little afraid of moving. Opposed to most VR games, here we really see and control our legs and arms, creating a weird but good sense of being in another body. Regarding that, one thing we noticed was how different people perceived the distance to the floor. Because the no one was the same height but the avatar’s height is fixed, some got the impression of being too near the floor. But the feeling of being in a football pitch is great, despite it not having the same level of realism it could have if developed by good games studio.

Unfortunately we never got to make real tests with users, just some colleagues that helped us with the project during the development process by moving at the Kinect or using the headset for us, so that we could fix and tweak settings and values on Unity.

Nevertheless, next follows our plan on how we planned to evaluate the project. Our planned method is composed by a survey and direct observation.

We believe the best way to reach any conclusions on whether or not users were having a good experience would have been to ask them. For this purpose, surveys would be created as a form of evaluating the experience. The surveys would make questions to analyse the general enjoyment while playing the game, how would the user rate it regarding motion sickness, the immersiveness of the virtual world, and at what level did they feel they were with another person in that world. These surveys should be anonymous to prevent people from feeling any kind of intimidation or pressure when giving responses. Also, they should be online, because paper survey filled after the experience can more easily be mapped to a user.

Another simple and way of evaluating if the game’s objectives were met is to see if players feel like playing the game again, but this methodology has some limitations as it only works on one way. If players have the will to keep playing it means the project met its objectives. However, the opposite conclusion does not apply, as many reasons can lead to the player not wanting to play again.

Also, during the playing tests it would be important to have an heterogeneous group of people regarding different aspects:

• **age**: people with different ages tend to like different things. Normally younger people are more enthusiastic when it comes to technology, making it interesting to see how children, younger adults, adults and senior would respond differently towards this game.

• **experience with VR**: people who have already used VR or is used to it, might be less susceptible to VR sickness. For this reason we would also intend to have players who are completely new to VR systems.
• **experience with video games**: although VR games are different from the “normal” video games, it would be interesting to test people who usually do not play video games to see if they feel different from those who do, and also if this kind of game is more appealing to them comparing to the others.

• **feelings towards football**: being this a football game, it is more likely to please people who like football. So that it doesn’t interfere on the results it would be good to have both people who enjoy football and people who do not.
6

Conclusion

Contents

6.1 Conclusions ......................................................... 65
6.2 Future Work ....................................................... 65
6.1 Conclusions

Virtual Reality is one of the topics of the moment. While some have no doubt that it will be the next big technological evolution, others believe that it is destined to fail, and many are still in between with both some enthusiasm and disbelief about the technology.

In spite of the lack of investment from big game studios in the development of good games, every day new VR content gets its way to the market, but many analysts claim that there is not yet a killer app; that one app or game that will change the course of VR for good. It is not guaranteed that that killer app is fundamental for VR, nor that will ever exist. Of course some games are more popular than others, but having that one game to revolutionise the industry may not be the key. In the beginning most VR experiences were isolated, but people have started realising how fun and better VR be when shared.

Based on that, we decided to create a VR game with a social component. The concept started growing around making a football game to be played between two people. Because we wanted the game to be as realistic as possible regarding how players would move and interact with the environment, we decided to add Kinect sensors to the experience, in order to track all body parts of the users, and remove the need of wearing specific trackers.

The result was a game on which each player is placed on one goal, and they can kick the ball from one goal to the other, trying to score and not concede goals. Having another person there playing with us really makes a difference, because even though we only see their virtual representation, we can hear and we react to the same events.

Although the gameplay is not quite finished with some parts needing to be perfected, we think that the game already provides a good amount of fun for the people trying it. We are able to kick the ball in the direction we pretend (most times), and see the other person moving, and it makes us feel that we are indeed in the company of somebody.

6.2 Future Work

Regardless of what we were able to achieve in this work there is always room for improvement.

There is one feature that we had planned to implement, which was to keep the ball always moving, so that it did not need to be re-positioned after every single kick. From our perspective, it would not only be more challenging but also provide a better immersion for the players, because in the real world, objects do not teleport from one point to another, so we wanted to minimize the occurrences of this situations as possible as we could. We have given it some time to think about ways to achieve that, but we were not able to implement them yet. Nevertheless, it would involve creating a sort of attraction force in the goal direction, so that the ball does not stop far away from the players.
Performing proper user testing would be the next step to take. For the eventual continuation of the work there are a couple aspects and features that it would be interesting to develop.

First, being this a football game it is important that real football rules are tested and verified. At the moment we have no rules implementation, so it is possible to do anything that the physics may allow, for example using the hands when not in a “goal keeper state”. Also, providing an User Interface (UI) for displaying the current score is fundamental on all games. Some customisation would also be interesting, as we have created three player models, with different football uniforms, but we always instantiate the same one.

With the release of standalone HMDs we see here the opportunity to extend the usability of the game. As we saw during the research on how to implement the Network synchronisation, it is possible to send the Kinect data to other devices over the network. In that section we mentioned smartphone HMDs, but standalone ones would be an even better option, given their better computational power and graphics. It would turn the gaming experience free of cables which is one of the current problems in VR. Also, the communication between players is an important element of our game, we could devise mechanisms to have audio working on networked games between users who are not in the same physical space.
Bibliography


[3] CCP Games: ‘We Expected VR To Be Two To Three Times As Big As It Was’. [Online]. Available: https://uploadvr.com/ccp-games-we-expected-vr-to-be-two-to-three-times-as-big-as-it-was/


[58] Motion Sickness. [Online]. Available: https://en.wikipedia.org/wiki/Motion_sickness#Motion_sickness_due_to_virtual_reality


