POLITECNICO DI TORINO

Master's Degree in Computer engineering

Master's Degree Thesis

Multiplayer implementation for the remote assistance in a Virtual Reality assessment tool and preventive intervention for Alzheimer's disease

Supervisors	Candidate
Prof. Gabriella OLMO	Francesco PENENGO
Prof. Vito DE FEO	

December 2022

"Till we believe in something"
Simon Neil

Summary

Since the end of the last century, one of the main fields of interest for the world of Virtual Reality (VR) has been its application in healthcare and medicine as, thanks to its nature, VR made easy to visualize complex stimuli in a controlled and secure environment.
Recently this technology started to be more accessible to the public, gaining more and more importance and widespread availability. With this in mind, the PEDAL (Personalized Environment for Dementia Assisted Living) project has been designed in order to develop a VR tool which would help in the assessment of the symptoms of Alzheimer's Disease (AD) patients and in the subsequent intervention. AD is the most common form of dementia, millions of people in the world are affected by it, for the largest part the elderly ones, thus becoming more and more common as the life expectancy increases. The underlying idea behind the PEDAL project is to detect early signals of disorientation and challenges in wayfinding in familiar urban areas, which have been discovered to be one of the earliest symptoms in people affected by dementia. The project has been carried on by a group of researchers and students from the Politecnico di Torino and University of Essex under the supervision of Professor Vito De Feo.
Thanks to the work of my colleagues Basil John, Daniele Bigagli, Giacomo Mineo, Jenitha Martins, Joel John and Simona Potenza, the project allows the user to navigate in a virtual environment modelled to resemble the city of Colchester following a pre-determined path to reach a known destination with the help of a virtual assistant. The final end of the tool is to monitor the patient and assess its level of spatial disorientation gathering both vital parameters and data regarding his performance in the task given. The experience is available through the use of a VR headset and a stationary bike electronically connected to replicate the movement of the handlebar and the pedaling in the virtual world in a secure way.
The goal of this thesis is to present the implementation of a networked multiplayer system in order to establish a remote connection between the patient and the clinical supervisor, thus reaching a further degree of freedom and flexibility in the usage of the VR tool. During the development we explored the most common multiplayer systems that are available in the Unity ecosystem, both proprietary and

from third-party libraries, using them in different stages of the project to finally find the most suitable one for our purpose.
The first implementation has been done on a desktop-based prototype of the project in which the user could move in a fictional city. At this stage, the networking design was split in two parts: a low-level communication system using TCP sockets via two different scripts, one for the Client code, written in C# to provide full interoperability with the Unity ecosystem, the other for the Server code, written in Python, and a high-level multiplayer system using the third-part package Mirror Networking. Mirror, a Unity package written on the basis of the now deprecated UNET, the proprietary Unity's solution for multiplayer games, enabled us to connect two remote clients running the project with a deep possibility of customizing all the aspects regarding the networked connection, from the server to the transport method to employ.
Moving on from the first prototype, the project was partially rewritten to become a VR ready tool with the Oculus Integration package; after having evaluated the networking solution with Mirror, we decided to shift to another package named Photon PUN2. The reason for this was that Mirror's principal aim is to provide a solution for Massively Multiplayer Online (MMO) games, thus tailoring their functionalities towards that world of multiplayer gaming. Besides, Photon, even in its free subscription tier, provides a hosted server and up to twenty concurrent users, more than enough for our application, facilitating and speeding up our development time and easing the costs. We also used Photon's remote procedure calls to replace and improve the previous low-level networking system based on the manually coded TCP sockets.
In the end, the current state of the project allows the supervisor and the patient to connect to the same server and start the experience in an assisted modality in which the supervisor controls the virtual assistant and can guide and monitor the patient during his navigation in the city.

In this thesis everything will be covered in depth: we'll start with an introduction on the history and motivations behind the PEDAL project and the need for a multiplayer tool in chapter 1, moving on to the analysis of some literature and background in chapter 2. Chapter 3 will be dedicated to an overview of the tools used for providing the project to the final user, while chapter 4 will provide a detailed description of how the development was carried on. In chapter 5 the workflow of the final product will be presented via some images taken from the game itself and explained. Finally, chapter 6 will be dedicated to the introduction of some of the features and improvements that are currently undergoing development or are planned for the future of the project.

Acknowledgements

Thanks to my family and friends, the most important values in life and my biggest sources of inspiration. Thanks to all my colleagues in the PEDAL project and to Professor Vito De Feo, it has been a pleasure and a valuable experience.

List of Contents

List of Figures

Acronyms

VR Virtual Reality

PEDAL Personalized Environment for Dementia Assisted Living

AD Alzheimer's disease

MMO Massively Multiplayer Online

HMD Head Mounted Devices

AR Augmented Reality

MR Mixed Reality

XR eXtended Reality

UE Unreal Engine

IDE Integrated Development Environment

VCS Versioning Control System

UNET Unity NETworking

TCP Transmission Control Protocol

UDP User Datagram Protocol

FPS First-Person Shooter

DLL Dynamic-Link Library

URP Universal Render Pipeline

Chapter 1

Introduction

1.1 PEDAL Project

The PEDAL (Personalized Environment for Dementia Assisted Living) project sits in the category of serious gaming: the goal of the researchers is to develop a VR tool which would help in the assessment of the symptoms of Alzheimer's Disease (AD) patients and in the subsequent intervention. AD, the most common form of dementia, is a neurodegenerative disease which strikes millions of people across the globe. One of its first symptoms for people affected is the inability to recognize and orientate in familiar areas.
To address this the team developed a game through which, with the use of a VR headset and a stationary bike, the patient could navigate in a virtual city modelled to resemble the one he lives in with no concerns for his security and in a monitorable environment. His goal will be to reach a certain destination starting from a set of pre-determined spawning areas, with different level of difficulties and hints, while some metrics like the time spent and the choices at intersections and so on are registered and monitored by medical supervisors.

The goal of this dissertation will be to present how a multiplayer solution was found to let the medical staff be able to monitor, guide and gain information from a remote post while the final user is free to play at his home, starting from a low-level implementation of a socket connection between a C client, made to run in a Unity script, and a Python server, designed to exchange information on the game itself at any suitable moment, to the assessment and employment of two of the most well-reputed high level multiplayer packages available: Mirror and Photon. The implementation choices have always been made keeping in mind the need to make an easy-to-use game and not to differentiate too much the experience from the single-player one.

1.2 Multiplayer Gaming

The industry of video-gaming represents the most profitable source of entertainment in the world, with an estimated revenue of nearly 200 billion US dollars in the year of 2022 and constants growth projections of around 5% a year until 2025, easily overtaking two others widespread industries such as music and movies. It feels safe to say that a major part of its success its due to the possibility for the game enthusiasts to play with one another, either over the network or sitting in front of the same console or computer, giving the multiplayer games a longevity and an always-feel-new feel that they would not otherwise have.

To get back to the origin of the first multiplayer games we must go as far as the middle of the 20th century, where the first prototypes saw life at the most important fairs around the world. At this stage the games consisted of a rudimental computer everyone played against; the system would then give and record a score that could then be compared with those of past players. In the 1970s Atari was the first big company to focus solely on the commercialization of videogames. Simultaneously the first arcade machine began emerging in public areas across the globe. The following years saw the advent of the first personal computers, marking the turning point for the widespread of videogames to the public; it was however not until the recent years that the availability of cheap and fast home internet connections, along with the birth of social communities such as Steam, allowed the game manufacturers to fully concentrate on the making of multiplayer games, both creating new ones and converting mono-players titles.

Although the greatest chunk of the market is revolved around the mass entertainment, gaming has also seen a lot of practical uses for medical, rehabilitation, educational and other "serious" purposes. These are often referred to as Serious Games and have themselves benefited from the collaborative nature of multi-user game modes. [1]

1.3 Virtual Reality

Virtual reality (VR) is right now one of the most focused technology in the spectrum of computer information, even if there is not a official definition we can describe it as an immersive replica of our reality available via head mounted devices (HMD), made in three-dimensional graphics. A lot of market-leading companies, such as Meta, Microsoft and Apple are investing in those HMDs, trying not only to snatch a portion of the gaming industries from traditional consoles or PCs, but also to redesign the Web of the future. The term VR is often accompanied by other terms such as Augmented Reality (AR) and Mixed Reality (MR). All these technologies fall under the category of eXtended Reality (XR) and present some differences

between them: while VR aims at the full virtualization of the environment, AR brings virtual objects into the physical reality. MR represents a junction point between AR and VR, since the virtual objects (also called holograms) brought to our real environment will be interactable.

VR has gained a lot of attention from the world of healthcare by researchers willing to explore its functionalities and exploit possible improvements over the traditional medical treatments. A lot of research has been carried on in fields such as therapeutic support, prevention and rehabilitation. Advantages encountered during these approaches have been highlighted from the perspective of the medical staff since, thanks to its nature, VR made easy to visualize complex stimuli in a controlled and secure environment, providing a greater flexibility and a easy way to get immediate feedback.
Similar positive aspects have been observed also from the point of view of the end users: for [2], VR promises increased motivational and training effects. Other studies have pointed out how the recreation of a virtual environment can lead to a more active participation on the part of the patient. [3]

While there seems to be a lot of studies to back up the use of VR in medical fields, we had to thoroughly consider the fact that a lot of people could experience motion sickness and general discomfort while wearing a headset. With this in mind we tried to develop a solution that could comprehend both effectiveness and safety, and we did so designing the product for the use with a stationary bike that electronically transmits the movement input to the VR experience to avoid the need of making the player walk and possibly losing balance, while also the in-game avatar is displayed on a bike to make a deeper connection to the real world.

Chapter 2

State of the art

2.1 VR in healthcare

The last two decades have seen a lot of interest from the world of medicine towards VR-related technologies as a tool for simulation, treatment, rehabilitation and other healthcare fields.
Technological improvements in graphics, processing and hardware designing, along with a broader affordability and popular knowledge of VR devices have sped up significantly this process in the recent years.
The focus of this chapter will be to present some of the most remarkable applications that Immersive Virtual Reality (IVR) took part into the world of healthcare. To give a better background to the reader, some more punctual definitions of IVR and VR devices are listed in the continuation of this chapter.

2.1.1 VR devices

IVR can be defined as a VR system that provides at least 3 degrees of freedom (DOF) of head orientation tracking and uses a head-mounted display (HMD) worn by the user who will be completely separated from the surrounding environment. While, as we will do in the rest of this dissertation, people commonly refer to IVR as VR, the term has been coined to distinguish immersive VR from Semi-Immersive Virtual Reality and non-immersive Virtual Reality, both of which will be outside the scope of this paper. Examples of widely commercially available VR devices are the Meta Quest (Figure 2.1), PlayStation VR by Sony and HTC Vive.

Nearly all of them are composed of an HMD, which is basically a mobile device in charge of the computational and graphic tasks, designed to be worn around the head to provide full isolation while two adjustable lenses placed in front of the eyes render the VR experience, and two controllers through which the user can interact with the virtual world. VR experiences, especially those aiming at getting

Figure 2.1: HTC Vive Pro

a further degree of realism, can also be extended with the use of haptic technology, an interface that enables bilateral signal communications between the user and the simulation. Haptic devices often, but are not limited to, substitute the hand controllers in medical simulations to give the right stimuli associated with touch, pressing, temperature changes and so on. In figure 2.2 we can see an example of haptic glove developed by HaptX, designed for surgery simulation.

2.1.2 Main areas of interest

As stated in the previous chapter, the major points of strength of VR are that it provides a great flexibility thanks to the countless simulation we can create and that they can be easily observed providing immediate feedback to the supervisors in charge of monitoring the users. This allowed researchers to experiment and employ VR tools in various fields such as:

- **Medical staff training**: VR simulation is set to become the standard for ensuring the acquisition of a wide range of practical learning for medical professionals. The advantage of simulators in medical training is not only in the reduction of (human) costs. They allow for better planning of training and objectives, progression and tasks. Progression, in particular, is of major

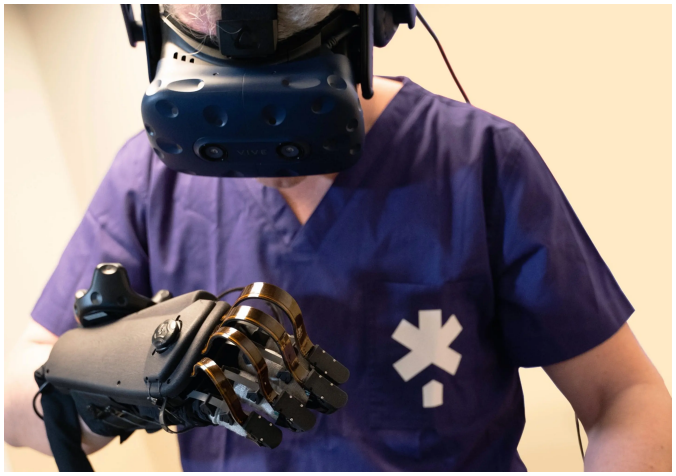

Figure 2.2: HaptX Glove

interest: with a simulator, you can start with what is simple and then move on to complexity.

- **Mental health and psychological therapy**: It's one of the first fields in which researchers have gathered information about positive aspects of VR treatment on patients, from those suffering from PTSD, who represents nearly the 3.6% of the USA population, to people with phobias and anxiety disorders. The benefits come from the possibility to face some scenarios that normally induce discomfort in a more controlled environment to help come with them in a gradually manner.

- **Rehabilitation**: VR is being currently used in rehab for diseases both in the sphere of physical injuries and mental illnesses with great results: a study from Feng H et al. [4], in which an even number of patients affected by Parkinson's disease (PD) have been divided in equal groups, one receiving a traditional treatment and the other undergoing an experimental VR training resulted in significantly better performance for the VR training compared with the conventional physical therapy group. Positive results were also yielded by VR treatment for people with upper extremities [5] and spinal cord injuries [6] The effectiveness of this kind of treatment has been quite universally associated with the fun and interactive aspects of the tools used, which often resulted in a greater engagement and overall enjoyment of the therapy compared to the traditional methodologies [7].

- **Prevention and assessment**: It's one of the latest frontiers of VR application in healthcare, as well as the main scope of this dissertation. There are a lot of ongoing studies on the efficiency of VR as an assessment tool for people with

suspected mild cognitive impairments or dementia: [8] took into consideration nine different studies and experiments on the subject in which good results were observed in differentiating people with dementia from controls, also stating that the greatest benefits are seen when VR tools are used alongside more traditional models.

2.1.3 Background and related works

The PEDAL project follows the steps of previous studies that take into consideration the measurement of spatial disorientation in patients as a signal of dementia in its early stage. One experiment by Chimezie O. et Al. [9] saw the prototyping of a semi-immersive virtual reality system named GRAIL, consisting of a treadmill and a 180° projection surface with optical motion capturing technology. The participants were asked to walk on the treadmill while their movements were tracked so that the projected environment would move accordingly until they reached a certain destination in a known area. The goal of this experiment was to measure motion behavior responses and vital parameters while deliberately adjusting the VR landscape to introduce different degrees of disorientation.

Another study by David Howett et al. [10] implemented an immersive Virtual Reality tool with a headset in which participants had to move in different shaped paths while remaining into a small area.

For the PEDAL project we decided to implement a fully immersive VR tool in which the users, riding a stationary bike, can move in a virtual area modeled to resemble a familiar city while given the task to reach on of the checkpoints located in the proximity of real points of interest of the city itself. To reach the full flexibility of use for this tool, a multiplayer solution has been implemented to allow the remote monitoring of the patient.

2.2 Multiplayer serious games

The idea of exploiting the collaborative nature of multiplayer games to improve the effectiveness of tools designed for healthcare and education has been studied for quite some years. Unfortunately, due to the difficulty in combining the design concepts of serious and collaborative games, we do not have many examples.
According to [1] the collaborative mode can promote more social involvement when compared to traditional gaming, especially for more extrovert individuals, eliciting more social involvement and empathy.
Out of the few examples of multiplayer serious games the majority of them seems to be focused on the topics of education and learning, but we can see implementations also for other kinds of medicine-related fields.

Chapter 3

Technologies overview

3.1 Hardware

From the first steps into the development of the project one of the main things to consider was the choice of the hardware on which the finished work needed to be delivered, as different devices require different SDK integrations, code tweaks and design patterns.

The first decision that the team made was about the HMD; the factors we took into considerations were: the need for a good and reliable product, availability and affordability for the end user, comfort, ease of use and an active community of developers who could provide help if needed during the development. In the end the choice fell on the Meta Quest 2 from Meta, company previously known as Facebook, while the main alternative was the HTC Vive Pro, a device of the same quality range but with a higher price and probably less known by the public.

Later we also decided to adopt a different strategy from the treadmill employed in similar studies, designing a custom stationary bike electronically connected to a computer, helping to tackle the need of giving the user the ability to move in every direction, while also addressing some safety issues that could present themselves when people wearing an HMD must stand and walk.

3.1.1 Meta Quest 2

The Meta Quest 2, previously known as Oculus Quest 2, is an Android-based stand-alone (does not need connections with a PC to work) head mounted device developed by Meta. Its easy integration and development with Unity on any kind of operating system, along with a lightweight and cozy design (Figure 3.1), made it the perfect fit for our use-case.

It also presents a fairly high resolution of 1832x1930 pixels per lens, 6BG of RAM and the Qualcomm Snapdragon XR2 chipset, features that allows to achieve

a better realism in the simulation.

Figure 3.1: Meta Quest 2

3.1.2 Stationary bike

The first prototype of the stationary bike (Figure 3.2), developed by the electronics engineers, is composed of a common bike connected to a bike trainer, an apparatus that fixes the bike in place with a clamp and a roller that increases the friction on the rear wheel. Thanks to electronics connections and Arduino chips, the movement of the pedals and the steering of the wheel are transmitted to the PC to enable the movement of the character.

Figure 3.2: Stationary bike prototype

3.2 Software

3.2.1 Unity

Unity, along with Unreal Engine (UE), is one of the most popular game engines available; it has been chosen for the development of the project thanks to its cross-platform nature, less steep learning curve compared to UE and affordable price for small organizations. It is also perhaps the game engine that has made

the greater steps towards XR development, with plenty of available resources and third-party libraries.

Unity relies on the C# programming language for the scripting part, allowing the developer to use its preferred Integrated Development Environment (IDE) like VS Code or JetBrains Rider, while everything else is demanded to the Unity Editor itself. Unity uses a component-based system, where every element you place into a scene, which can be considered as an independent part of the whole game, is a game object with its own components that define its physics, animation, behavior and look. Every game object can be converted into a "prefab" in order to be efficiently reutilized in other parts of the game.

Another important feature for those developing with Unity is the availability of pre-made scripts and game objects that can be integrated as packages for certain functionalities. One example of a package that we used is XR Interaction Toolkit: a high-level interface for creating cross-platform VR and AR experience. Along that, we also imported the Oculus integration package, the official SDK by Meta which provides a way to access nearly all the functionalities of their HMD.

3.2.2 Unity Collaborate / Plastic SCM

Working as a team on the project, one of the main concerns was to find an efficient and simple way to share and work simultaneously from different locations while keeping track of the changes and transmitting them to the others.
Unity Collaborate was our first choice: it provided an effective, while basic, versioning system with limited functionalities and, being a proprietary solution, worked fine with Unity: not only it was integrated into the editor but also provided a built-in system to synchronize and store over the network only the needed files, which would have been a harder task using the most common tool Git LFS.

Collaborate was discontinued and gradually replaced by Plastic SCM, a git-like versioning control system (VCS) with full functionalities which addressed the limitations of the former tool. It is also integrated in the Unity editor for the most basic and used functionalities of pulling (getting changes from the network) and pushing (committing your local changes so that they are available to the others), while it needs its own free to download desktop program to access more in-depth features like branching, conflict resolution and so on.

3.3 Networking

To provide a fast, reliable and secure way of allowing a remote connection for our VR tool we evaluated and used different networking libraries. At the time of the first integration in the project of the multiplayer implementation there was no maintained proprietary solution by Unity, as Unity Networking (UNET) was

deprecated and later removed while a new tool wasn't released until the second half of 2022. As a result, our focus was pointed to two third-party libraries: Mirror and Photon PUN 2.

Before diving into some of the most important features of Mirror Networking and Photon PUN 2, the remainder of the chapter will be dedicated to present the basic concepts of a networked game. Most games present a client/server architecture:

- The server is an instance of the game where all players connect to and is in charge of controlling and managing various aspects such as communication, hit validation, spawning and many others.
  If a game has no instances that only acts as a server (dedicated servers), one of the clients will fulfill both roles, becoming a host server.

- The client is an instance of the game which connects to the server via a local or internet network.

The transmission of data across the net is generally ruled by one of the two most common protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP):

- TCP is the protocol used by HTTP, SSH and many other internet applications. It needs a connection to be opened between client and server in order to make possible the exchange of data and it has built-in reliability, since packets who arrive at their destination are controlled and if there are some missing, TCP will resend them. This comes with the trade-in of a higher latency when compared to UDP.

- UDP is used for real-time communication, where low latency is the key, and a missing packet is not something that could break the state of the application. UDP offers also an in between solution where packets can have different implementations to gain reliability.

3.3.1 Mirror

Mirror Networking is the first networking tool we investigated and used in our project. It is based on UNET, being born with the idea of tackling the bugs and limitations that later lead to its deprecation.
Mirror provides a client/server architecture with a simple interface through which you can pick the role you want as soon as you launch an instance of your game. It is also transport-agnostic as it let you choose between several different protocol implementations, both based on TCP and UDP.
Mirror enforces server authority, meaning that the control over the behavior of

all the networked game objects is, by default, demanded to the server, allowing greater control and avoiding the possibility of cheating.
Game objects that are synchronized over the network needs to have a special component that comes with the Mirror package, so that their position, rotation and animation state can be correctly updated and rendered to all the connected clients.
Finally, the scripting part is facilitated by the possibility to extend a base class that provides a series of common functionalities: spawning a game object, getting info about the current number of connected clients and so on. It also allows a bi-directional communication between the server and the clients.

Mirror's biggest aim is to provide a solution for MMO games, providing specific functionalities and optimization at the cost of a more difficult code and handling of first-person shooter (FPS) games. For this reason, it was not the best choice for our application.

3.3.2 Photon PUN 2

Photon Unity Networking (PUN) is the second unity networking package we imported in our game. It comes in different versions, both free and paid; one of those, Photon PUN 2 free, is the one we chose for our project, as it provided more than sufficient bandwidth and concurrent users.

The main difference with Mirror is that PUN enforces a lobby system, in which all the clients must first connect to a specific room until the master, generally the host, starts the game by loading a "level", a networked scene of our game. This lobby implementation has been very useful in the development of the application since it allowed to create a system where the medical supervisor could create a room, wait for the patient to connect and then start the simulation.

Similarly to Mirror, PUN provides a series of component to synchronize the game objects over the network as well as a set of expandable base functions for the scripting part.

3.3.3 ParallelSync

Parallelsync is a handy Unity package that mitigates the development and testing time for networked games, since it allows to make a copy of the current project that is automatically updated when the original one is modified. Having two different Unity editors running at the same time and with the same content allows the developer to quickly test the game by clicking "Play" on both editors, with the traditional alternative being to build an instance of the game every time a modification is made and then run it along with the Unity editor.
Parallelsync is available at https://github.com/VeriorPies/ParrelSync.

Chapter 4

Development

The development of the PEDAL project saw different phases during which different strategies were employed and several new functionalities added to reach a finished product. In this chapter we'll analyze in depth the implementation choices and the reasoning behind them, both technically and in the context of the integration with the work of the colleagues.

4.1 First prototype

The first implementation of the PEDAL project was that of a desktop-based application featuring a fictional squared city in which the user could move in order to reach a building marked as his house.
Graphically the player character was modelled to show only its arms, while the camera was placed in such way to show its point of view. A black arrow pointing towards the direction of the house and an indication of the distance in meters were the hints the player needed to follow to complete the task. (Figure 4.1)

Figure 4.1: Capture of the first prototype

As simple as it was, this prototype established the foundation of the whole project and allowed the team to consider possible improvements, pros and cons with a solid basis.

From the perspective of the multiplayer implementation, the first steps we took were to find a way to send pieces of information from the player to the supervisors at a low level at pre-determined points of the game and to connect the two instances across the network: the solutions we considered are listed in the following chapters.

4.2 Low-level socket communication

The first implementation for the low-level communication system we designed was composed of two different scripts needed to establish a socket connection between the user and the supervisor:

- The client, written in C# in order to grant full interoperability in the Unity ecosystem.
- The server, written in Python to run on the remote computer of the medical staff.

The initial aim was to provide a way to monitor, guide and gather data about the final user performance throughout his gameplay.
Information exchanged at that level should not be critical for the performance and continuation of the gameplay, since there is no way to grant a connection fast and reliable enough to match the high-level one provided by the Unity packages we were planning to use. Rather, info related to time spent, disorientation measurements at critical points like intersections and other useful insights were tracked, written on a file and sent in a pre-defined structure via the socket connection.

4.2.1 Client implementation

The client script has been written in order to establish a TCP connection between client and server where the former could send a fixed number of bytes to the latter at any time. Since there was no need of a bi-directional communication system, the client was only designed to send data and not to listen to server responses.
The main two methods exposed by the script were named ConnectResult and SendData:

- ConnectResult was written to create a TCP socket at a specifiable address and port number, it returned a string indicating if the connection process ended positively or negatively. In the latter case no further steps or actions were taken in order to re-establish the connection or showing some errors to the user since, as we already mentioned, this kind of low-level network was not essential for the usage of the VR tool.

```
public string ConnectResult
   (string address = "localhost", int PORT_NUM = "8888")
 {
     try
     {
         this.client = new TcpClient(address, PORT_NUM);
         return "OK";
     }
     catch (Exception exception)
     {
         return ("Server is not active. Please start server
                    and try again." + exception.ToString());
     }
 }
```

The values specified at the signature of the method serves the purpose of default values and are overwritten if others are specified during the invocation. The connect function was intended to be called once for every networked scene, as soon as the Game Object in charge of managing the state was created.

- SendData was used to convert into an array of bytes the string message we want to send and writing it using a Stream.

```
public void SendData(string message)
 {
     byte[] bytes = Encoding.get_ASCII().GetBytes(message);
     this.client.GetStream().Write(bytes, 0, bytes.Length);
 }
```

The method, in its first implementation, assumed a fixed structure of the message to send, therefore not including any header or other indication of the number of bytes the server needed to read.

4.2.2 Integration in Unity

To maximize the interoperability between the script and Unity, the C# solution was built into a Dynamic-link library (DLL) which was then added to the Unity

project structure to be utilized by any script without the need of rewriting the same code every time a connection or an exchange of data needed to be made.

In order to do so, the script was written as a C class library in Visual Studio, an IDE belonging to the Microsoft ecosystem, built as a DLL and then added to a specific Unity folder named "Plugin". Doing so made the aforementioned methods of the client accessible to any script in Unity that needed them just by importing the DLL as a module, as shown in figure 4.2, where at line 3 the command "using" followed by the name of the DLL itself is highlighted.

```
1  using System.IO;
2  using UnityEngine;
3  using ClientConnector;
```

Figure 4.2: Module importing

After having imported the library, you can access its functionalities by instantiating a new object of the class Connector which is, in our case, the name chosen for the class, and then invoke the needed methods. Figure 4.3 demonstrate the instantiation of the object at line 11 and its usage at lines 16-17.

```
7  public class FilePos : MonoBehaviour
8  {
9      public float samplingRate = 1f; // sample rate in Hz
10     public string outputFilePath;
11     private Connector _connector;
12     private StreamWriter _sw;
13
14     public void OnEnable()
15     {
16         _connector = new Connector();
17         var result = _connector.ConnectResult("localhost", 8080);
18         _sw = System.IO.File.AppendText(outputFilePath);
19         InvokeRepeating("SampleNow", 0, 1 / samplingRate);
```

Figure 4.3: DLL usage

As it is clear from the code, apart from reducing the repeated code, this implementation choice allowed us to split the client functionalities into different methods invokable only if needed.

4.2.3 Server script

As previously stated, the choice for the programming language of the server script has fallen on Python, since it is nowadays one of the most widespread back-end languages and is the go-to choice when dealing with machine learning (ML) systems, which could be later integrated to analyze the gathered data in search of disorientation symptoms. For its implementation the server was composed of a

TCP client, bound to a specified hostname and port number, continuously listening for message arriving from the client until the number of bytes received matched the fixed size of the packets.

4.3 High-level networking with Mirror

In conjunction with the low-level communication system, we demanded the handling of the high-level networking features, such as the synchronization of the gameplay across the network, to the Mirror Networking package.

At this stage of the project, the main goal was to find a viable way to integrate a multiplayer mode into the game without making it look too different from the solo player experience and integrating as many functionalities as possible, while also evaluating pros and cons in the making for reaching a stable implementation in future steps.
Thus, the integration of Mirror has been carried assessing step by step the functionalities it offered us and how to exploit them.

As already explained in the chapter about the technical overview, Mirror provides a series of components we can attach to game objects in our scene to make them multiplayer-ready.

4.3.1 Network Manager

The first component needed for every scene that must be synchronized is called the Network Manager; we added it to an empty game object called Network Manager for the sake of clarity, as shown in figure 4.4.
It manages various networking aspects of our project, such as:

- Game state management: keeping track of the numbers of players connected, the metrics of the network and the lifecycle of the application.
- Spawn management: The creation of game objects that are shared across the network for some or all the connected peers.
- Customization: It allows the developer to choose different networking options, which we are going to describe later in detail.

Out of the several features it provides, the ones interesting for our projects were the possibility to choose a transport method, the possibility to assign a prefab for the networked player and a quick and easy way to configure the network.

Mirror grants the possibility to choose between a series of pre-made built-in and even additional implementations, both based on UDP and TCP. Since our project doesn't have urgent needs in terms of concurrent users or reliability, we

experimented both with KCP, a fast UDP implementation without much overhead, and Telepathy, a TCP implementation. Both are built-in and written in C# for full interoperability and possibility of extension. Along with the transport method, the "Network Info" section of the Network manager component allows to specify easily the network address, max number of concurrent users and how to deal with disconnections. Lastly, in the "Player Object" section, the developer can quickly register a prefab that will be automatically assigned for every user, along with a series of prefabs that are registered for spawning when a certain condition is met.

Since everything it provides is implemented entirely using the API, Mirror allows to expand on the Network Manager component's features by using scripting to derive your own class from Network Manager and customize its behavior by overriding any of the virtual function hooks that it provides if needed for more advanced developments.

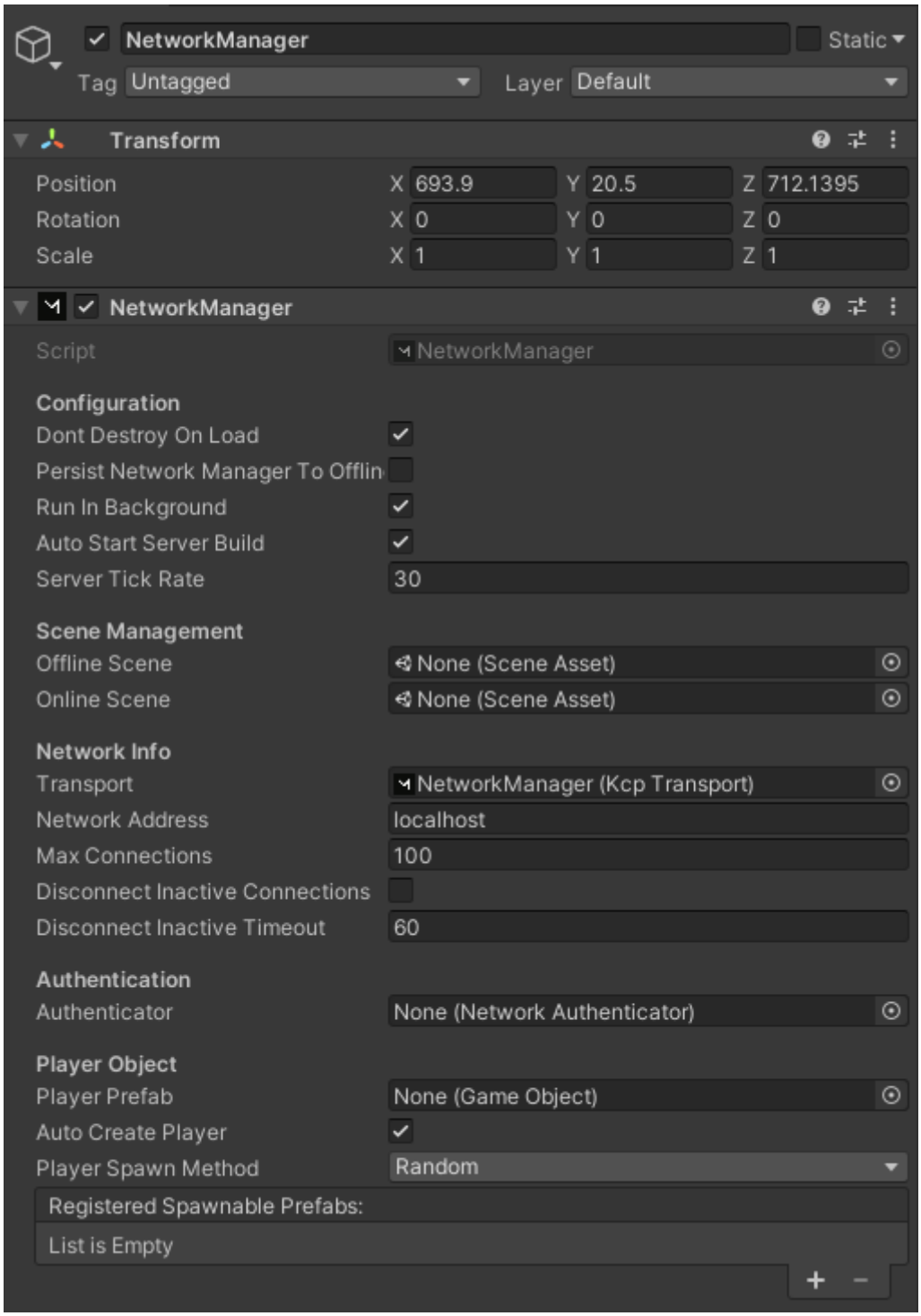

Figure 4.4: Network Manager component

4.3.2 Network Manager HUD

The Network Manager heads-up display (HUD) is a component that provides the basic functions to start hosting a networked game, or find and join an existing networked game. Unity displays the Network Manager HUD as a collection of simple UI buttons in the Game view (Figure 4.5).

The developer can use the HUD as a quick-start tool to build and test multiplayer games without the need of creating more complex ad-hoc user interfaces. This was especially useful for this stage of the project since the design of the lobby was not defined yet

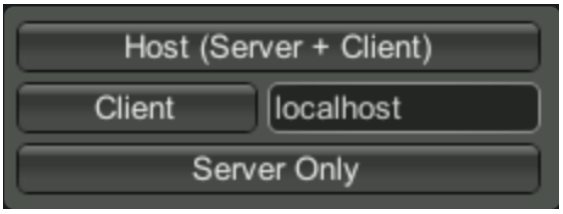

Figure 4.5: Network Manager HUD

4.3.3 Network Identity & Network Transform

The Network Identity component is at the heart of the Unity networking high-level API. It controls a game object's unique identity on the network, and it uses that identity to make the networking system aware of the game object. When we need to synchronize a game object over the network, be it the character controller or a spawnable prefab, we must add this component (Figure 4.6). Mirror leaves the possibility to check the option "Server Only" for game objects that need to be spawned only on the server and not on the clients.

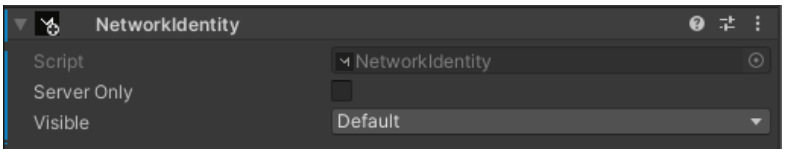

Figure 4.6: Network Identity component

Once a game object has the Network Identity component added to it, we can synchronize its position, rotation, and scale across the network thanks to the Network Transform component (Figure 4.7). The concept of transform is borrowed from Unity's built-in system, where a transform identifies the positioning, rotation and scaling along the 3 axis X, Y and Z.

Across the various settings, one of the most useful for our project has been the Client Authority enabling. As already explained in the previous chapter, Mirror enforces and uses as default server authority, meaning that only the server can manipulate the values regarding the transform of a game object.

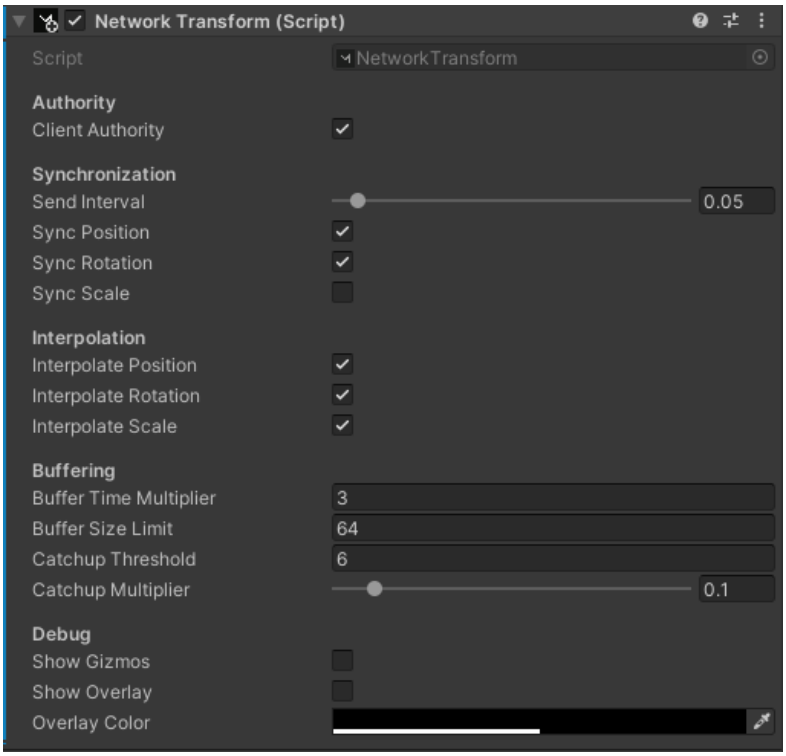

Figure 4.7: Network Transform component

4.4 Project refactoring

Once the main features and requirements were laid out thanks to the work on the initial prototype, the focus of the team moved on to the creation of a second version of the game with its full functionalities.
Although not directly related to the multiplayer implementation, all the new features were to be taken into account when assessing the most suitable networking system to use, not to mention the need to make it work smoothly with every new addition.

Arguably the most important change brought to the project was its migration from a desktop-based application to a VR one, keeping a close eye on optimization and best practices to avoid making it too resource consuming. We did so starting

a new blank project and importing from the prototype only the assets and scripts needed and also switching to the Universal Render Pipeline (URP) for maximum efficiency.

On this new semi-blank project, thanks to the work of Simona Potenza a new city, modeled to resemble Colchester, was used as the new environment for the project, providing a more realistic and complex area to explore. Along with that, Giacomo Mineo and Daniele Bigagli implemented a new checkpointing system and inserted a new interactive virtual assistant that guides the user in a more efficient way to his destination. (Figure 4.8)

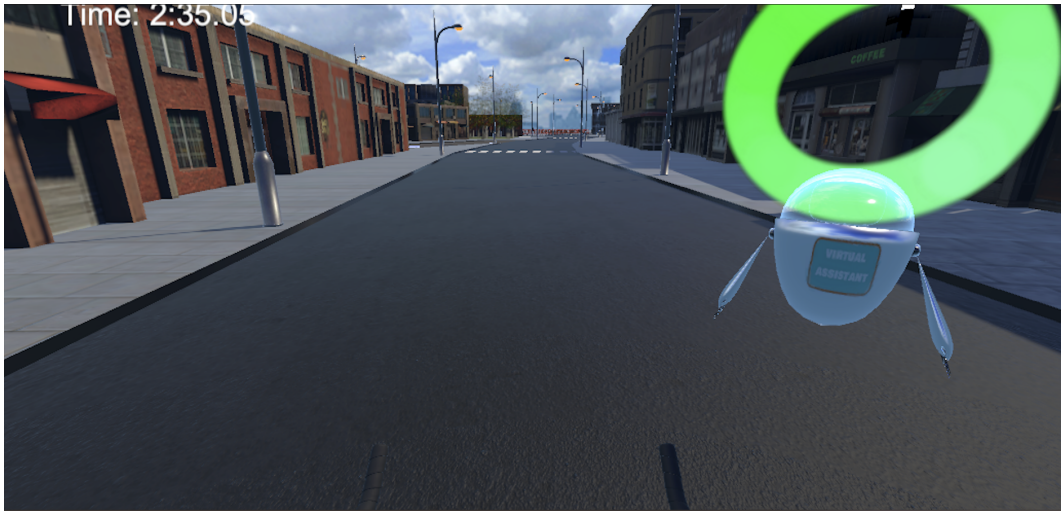

Figure 4.8: New project glimpse

Since more developments were planned along the way, we decided to work on a single scene both for the multiplayer and single-player experience, thus allowing an easier integration of the new features at the cost of carrying a little more complexity.

4.5 Switch to Photon PUN 2

After some consideration we decided to move from Mirror to Photon PUN 2 for our networking implementation. The main reasons are the following:

- PUN provides an easily configurable and hosted server, freely obtainable by creating an account on their website and generating a private API key for your game. This constitutes a great development facilitation compared to Mirror since it allows us to try our game without the need of hosting a server, which can be tricky and expensive.
- A wide online consensus on preferring it over Mirror for VR development.

- Vast documentation and availability of examples.
- A built-in lobby matchmaking structure, which makes a good fit for our project.

Before explaining how we integrated PUN into the project to exploit its high-level networking system, the next chapter will be dedicated to a discussion on the low-level communication system put on in the prototype of the project.

4.5.1 Low-level networking

Since PUN offered the possibility to use their back end as a dedicated server to host our application, we had to consider whether to maintain the client-server architecture script written for the prototype version with the downside of having to host another server just for it or to drop it and employ some other form of communication between peers and master.
Thankfully, PUN provides a way to enable a bi-directional communication and event signaling system with more than enough functionalities to replace the socket communication.
The backbone of this system is the Remote Procedure Call (RPC), essentially describable as method-calls on remote clients in the same room which accept several data types as parameter. One example of RPC is given below:

```
   [PunRPC]
void ChatMessage(string a, string b, PhotonMessageInfo info)
{
    // the photonView.RPC() call is the same as without
    // the info parameter.
    // the info.Sender is the player who called the RPC.
    Debug.LogFormat("Info: {0} {1} {2}", info.Sender,
                    info.photonView, info.SentServerTime);
}
```

In this case the info parameter collects data about the client who executed the call on the method, while the other two string parameters can contain useful messages or information that the server or other clients can immediately use or store. The [PunRPC] signature is all it's needed to mark a method as a RPC.
PUN also gives the possibility to define which clients execute the RPC in the case not everyone needs to send it.

While RPCs provide great flexibility by themselves, the main limitation is that the script they are defined into need to be attached to a networked game object.

To tackle this issue PUN exposed to the developers the RaiseEvent method: it works with every script and allows to set as content any serializable (convertible into an array of bytes) object.
An example from the official documentation:

```
private void SendMoveUnitsToTargetPositionEvent()
{
     object[] content = new object[]
              { new Vector3(10.0f, 2.0f, 5.0f), 1, 2, 5, 10 };
     // You would have to set the Receivers to All
     // in order to receive this event on the local client as well
     RaiseEventOptions raiseEventOptions =
         new RaiseEventOptions { Receivers = ReceiverGroup.All };
     PhotonNetwork.RaiseEvent
          (MoveUnitsToTargetPositionEventCode, content,
             raiseEventOptions, SendOptions.SendReliable);
}
```

We can see that we need to set up an event identifier, some options regarding who will receive the event and even a transport method and the encryption for the sent package.
In order to receive the events raised, a game object must be registered as soon as it is enabled as target for callbacks via script:

```
private void OnEnable()
{
    PhotonNetwork.AddCallbackTarget(this);
}
```

And implement the OnEvent interface to handle them:

```
public void OnEvent(EventData photonEvent)
{
    // Do something
}
```

Although having laid the ground for the usage of this functionality, our project is not currently using it as the main goal was to synchronize the state of the game objects only making them observable by the connected peers. That said, as will be explained in a later chapter, this will help in the future developments when the need for the synchronization and gathering of data will emerge.

4.5.2 Setup

Differently from Mirror, where the destination host could be specified as a parameter of the Network Manager component, PUN provides a quick setup window (Figure 4.9) directly into the Unity editor through which you can set up your project using the API key retrieved when registering and creating a new project from the Photon dashboard. Other settings can be handled directly from the Photon website.

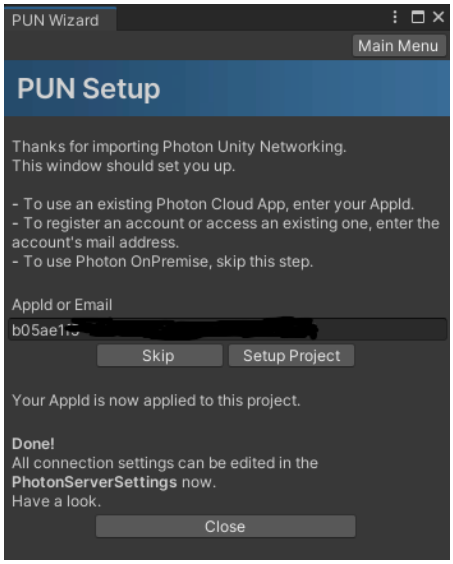

Figure 4.9: PUN wizard

4.5.3 Lobby implementation

The first different implementation choice we made between the prototype and the new implementation has been to move the players connection logic to the game mode selection menu implemented by my colleagues. Doing so provides a cleaner and more usable multiplayer implementation that can go along with the solo player one, getting close to a final product solution.

Leveraging the already mentioned lobby-based PUN matchmaking system, we added an "assisted" game mode to the start menu through which a series of

connections are made.
First one is the actual connection to the Photon server, for which a code snippet is provided below:

```
void ConnectToServer()
{
    PhotonNetwork.ConnectUsingSettings();
    PhotonNetwork.GameVersion = gameVersion;
}
```

The "ConnectUsingSettings" function allows the client to quickly connect to their dedicated host, while giving the possibility to specify further connectivity options which are, however, outside the scope of this dissertation.

Another important setting is provided by PUN by adding the following line of code:

```
PhotonNetwork.AutomaticallySyncScene = true;
```

As the name suggests, it allows to synchronize the game scene currently shown for every connected peer and it is useful for when the actual game has to start.

Once the client is connected to the server, a callback is fired: from then the players are allowed to join a room or create one if it does not exist yet.

```
public void JoinRoom()
{
    var roomOptions = new RoomOptions
    {
        MaxPlayers = 2,
        IsOpen = true,
        IsVisible = true
    };
    PhotonNetwork.JoinOrCreateRoom
       ("Room1", roomOptions, TypedLobby.Default);
}
```

In a similar way as before, once two players have joined the room one of them will have the possibility to load the game, bringing both to the Colchester map

where the assisted navigation can start. The player who started the game will take the role of the master.
The following is the code that implements the start of the experience:

```
public void LoadArena()
{
    if (PhotonNetwork.CurrentRoom.PlayerCount > 1)
    {
        Debug.Log("Number of players connected: "
            + PhotonNetwork.CurrentRoom.PlayerCount);
        PhotonNetwork.LoadLevel("New_Colchester");
    }
    else
    {
        Debug.Log("Number of players connected: "
            + PhotonNetwork.CurrentRoom.PlayerCount);
        connectionStatus.text = "Waiting for the second player";
    }
}
```

All the code presented runs both does not need to be adapted between the desktop-based and the VR application.

4.5.4 Colchester scene

To make the experience of the assisted navigation through the city as similar as possible to the solo one we decided to reuse the same game objects in a slightly different fashion: while in the solo-player mode the character controller is represented by the front part of a bike and is followed and guided by a floating robot, in the multiplayer mode the user controls the bike, and the supervisor controls the robot.

From a technical point of view, PUN, just like Mirror, requires adding a component to every networked game object. This component is called Photon View (Figure 4.10)

Other than assigning a unique View ID which will identify the game object across the network, this component lets the developer specify some options about the ownership, an important concept when dealing with networked objects.
The owner of a game object is the peer who controls it, by default the ownership is assigned to the one who instantiated it, like in the code below:

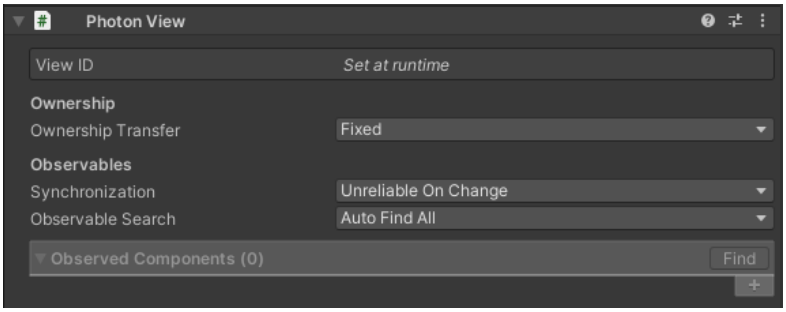

Figure 4.10: Photon View

```
if (PhotonNetwork.IsMasterClient) //Supervisor
{
    Debug.Log("Instantiating Master Client");
    player1 = PhotonNetwork.Instantiate("Virtual Assistant",
        player1SpawnPosition.transform.position,
           player1SpawnPosition.transform.rotation, 0);
}
else //Patient
{
    Debug.Log("Instantiating Patient player");
    player2 = PhotonNetwork.Instantiate("Bike",
        player2SpawnPosition.transform.position,
           player2SpawnPosition.transform.rotation, 0);
}
```

The ownership is transferred in case the owner is disconnected, but it's also possible to change it explicitly via a request to the current owner that needs to be accepted setting the "Ownership Transfer" value to "Request" or taking it without consent, when the aforementioned value is set to "Takeover".
At the current state of the project all our ownerships are fixed, meaning they cannot be changed at runtime, but it could be useful in future implementations if we need to let the supervisor take control of the patient character.

The Photon View lets you observe a series of component regarding your networked game object, like its transform or animation. While we did not need the latter, observing the transform is required in order to make the movement of one player go across the network and replicate to the other client screen: to do so the developer needs to add the Photon Transform View component (Figure 4.11).

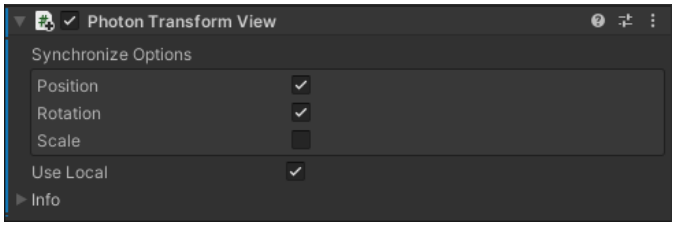

Figure 4.11: Photon Transform View

The component lets the user synchronize a combination of scale, position and rotation.

The code that controls the networked experience is simple: beside the instantiation of the characters' prefabs, we implemented a callback that listens for disconnection of either player and, in case it happens, will send the game back to the lobby menu.

```
public override void OnPlayerLeftRoom(Player other)
{
    Debug.Log("OnPlayerLeftRoom() " + other.NickName);
    if (PhotonNetwork.IsMasterClient)
    {
        PhotonNetwork.LoadLevel("Game Menu");
    }
}
```

Finally, some scripts in charge of controlling the player and camera movement were adjusted to work only on the local instance via the check

```
PhotonNetwork.IsMine()
```

4.5.5 Integrations

Once the networking scripting and configuration has been finished, the last part of the work was to integrate the multiplayer implementation with the new features introduced in parallel by the team.

The first step was to give separate controls to the bike and the robot prefabs, adjusting accordingly their connected scripts.

While part of the work did not create conflicts, we had to perform some code refactoring and assets reorganization in order to make everything work together. The main criticalities came from the fact that, while the controller game object is

inserted in the scene statically, the networked ones are instantiated at runtime and thus cannot be referenced directly by some scripts that need to do it.

For instance, the script that controls the minimap had to be adjusted in order to let it follow the correct player by dynamically getting a reference to a camera to follow and a similar work has been done for the point of interests' message box. Other scripts like the one who controls the checkpointing system did not need any modification.

Chapter 5

Connection workflow

This chapter will be dedicated to the presentation of the final connection workflow developed for the PEDAL project using some screenshots taken from the Unity Editor in play mode.

As already explained the starting point for the connection to the Photon Servers is commanded by the Lobby of the game (Figure 5.1).

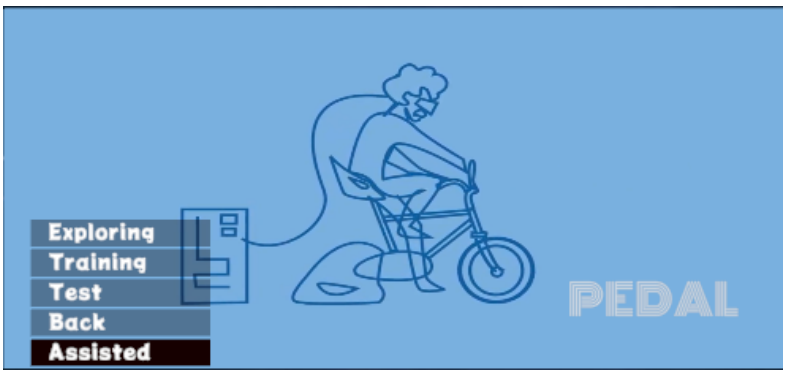

Figure 5.1: Game Menu assisted navigation

Once the "Assisted" menu voice is selected a script starts to listen to connections callback, highlighted to the user via a loading screen (Figure 5.2).

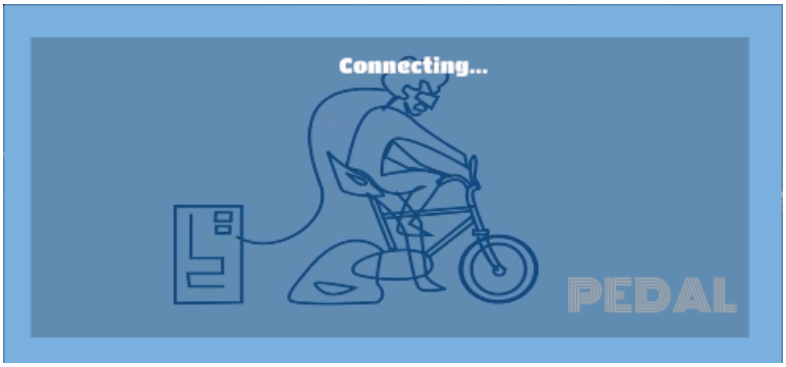

Figure 5.2: Game Menu connection

If the connection is successful, an option to join a room or to create one if it does not exist yet is given (Figure 5.3).

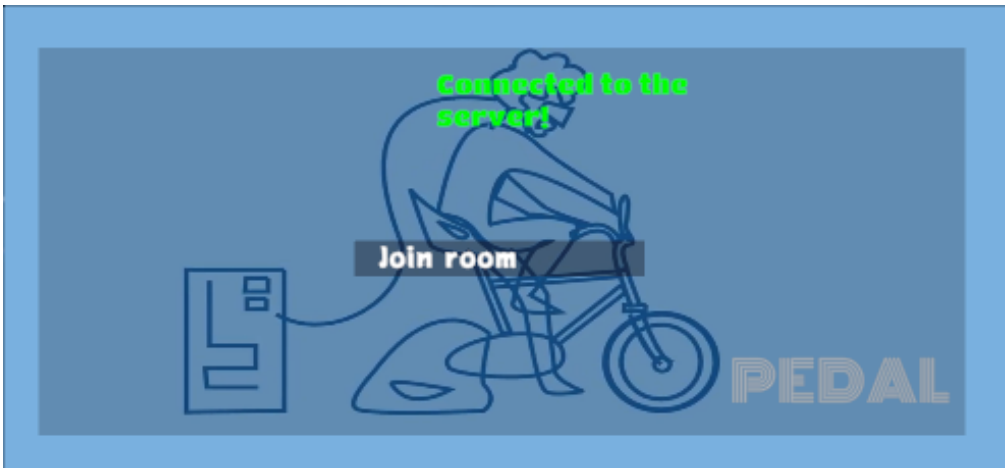

Figure 5.3: Game Menu join room

At this state of the project, the player who creates the room will be marked as the master client.
Once a room is created an indication of how many players is currently given and updated as soon as a new one enters (Figure 5.4 and 5.5)

Figure 5.4: Game Menu one player

Figure 5.5: Game Menu two players

If the master client, who is the only one enabled, tries to load the game while only one player is currently in the room, the message in Figure 5.6 is shown

Figure 5.6: Game Menu wait for second player

The load game option synchronizes the state on both clients putting them in a loading state to start the real experience. Below are two figures that shows the mutual view from the supervisor client who is controlling the robot (5.7A) and from the patient who is controlling the bike (5.7B)

Figure 5.7: View from the virtual assistant

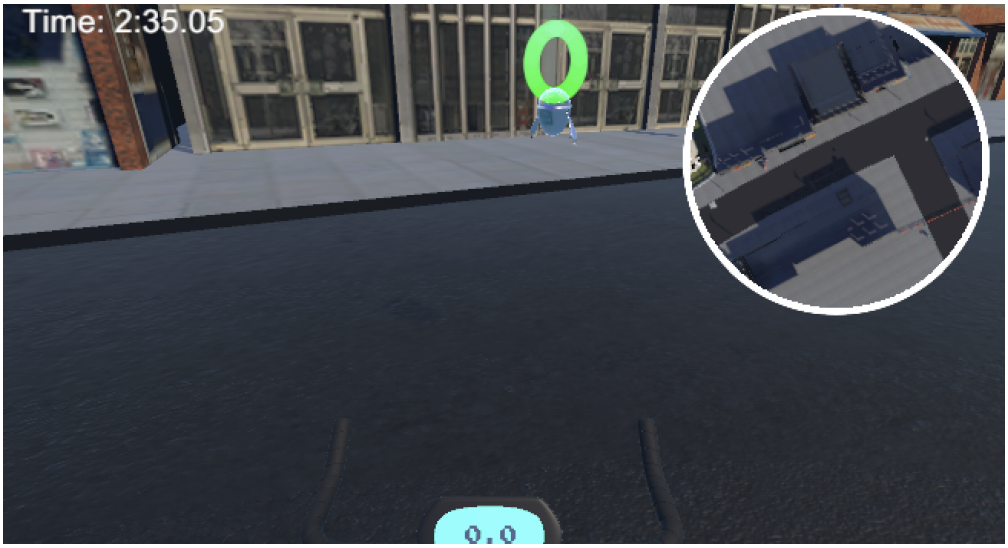

Figure 5.8: View from the bike

As we can see the patient's point of view is slightly offset to offer a view from above and capture the front part of the bike with the animated handlebar to achieve a greater sense of realism. The supervisor's perspective, instead, is a first-person one, as he only needs to focus on the movement of the patient.

Chapter 6

Conclusions and future steps

This thesis presented how a networking system was integrated into the PEDAL project for the remote assistance of patients by a medical staff allowing a more flexible and precise use of the tool.
During all the project development, from the prototype to the current state, the implementation choices and employed tools were all evaluated thoroughly to provide a final experience as useful and enjoyable as possible. To do so, the path marked out by the PEDAL team with the solo-player implementations has been followed, granting a continuative experience in both modalities and allowing a smooth integration for the upcoming features.

After having assessed various networking tools and strategies we chose Photon PUN 2 as the functionalities it provides, along with the relative ease of use and availability of documentation made it the best fit for the project. PUN also allowed to host for free our project on their dedicated servers speeding the process of development and testing while granting up to twenty simultaneous connections in its free tier.

Currently, the multiplayer implementation regards only the synchronization of position and rotation of the two controllers for player and supervisor, thus no criticalities were found in terms of bandwidth or resources usage.

As we moved on to the prototype to the current implementation, we were able to drop the low-level communication system made with TCP sockets, which were limited in usage and unsecure, in favor of the more exploitable Photon's built-in RPC system. While not currently used on the project, it could be the basis of several new functionalities.

At the state of art, the project provides an essential but complete experience to the user thanks to the modelling of a realistic copy of the city of Colchester, the introduction of a Virtual Assistant and a checkpointing system along with the possibility to being monitored remotely during the navigation.
Since our fucus has always been on producing an incremental project, current

and future developments are going to expand the experience with new features or improvements to the existing ones:

- Different navigation modalities are going to be introduced: from the first exploration of the city to the proper training in which the evaluations will be carried on. Each modality can be performed on the same environment or on different ones and with different levels of difficulty in terms of disorientations and other variables.
- Several new ways to interact with the city are going to be added: some, already mentioned in previous chapters, regards the possibility to explore some points of interest in the city through the use of dialogue boxes that will pop up once the user approaches them.
- The Virtual assistant can be improved adding other visual and auditive clues, like for instance voice feedbacks.
- Another multiplayer modality, in which the supervisor can monitor the experience from the point of view of the patient, can be quickly integrated following the already existing basis.
- Real time analysis through machine learning tools of gathered data regarding the performance of the user during the navigation for medical records and even for introducing an adaptative level of difficulty based on them.
- The low-level networking system provided by Photon thanks to RPCs can be exploited for a real time interaction in the user experience. For example the number of disorientation's instances can be adjusted according to the performance of the patient while the navigation is still being performed.

Bibliography

[1] Fábio Pereira, Sergi Bermúdez i Badia, Rúben Ornelas, and Mónica S. Cameirão. «Impact of game mode in multi-user serious games for upper limb rehabilitation: a within-person randomized trial on engagement and social involvement». In: *Journal of NeuroEngineering and Rehabilitation* 16.1 (Aug. 2019). DOI: 10.1186/s12984-019-0578-9. URL: https://doi.org/10.1186/s12984-019-0578-9 (cit. on pp. 2, 7).

[2] Andreas Halbig, Sooraj K. Babu, Shirin Gatter, Marc Erich Latoschik, Kirsten Brukamp, and Sebastian von Mammen. «Opportunities and Challenges of Virtual Reality in Healthcare – A Domain Experts Inquiry». In: *Frontiers in Virtual Reality* 3 (2022). ISSN: 2673-4192. DOI: 10.3389/frvir.2022.837616. URL: https://www.frontiersin.org/articles/10.3389/frvir.2022.837616 (cit. on p. 3).

[3] «Immersion of virtual reality for rehabilitation - Review». In: *Applied Ergonomics* 69 (2018), pp. 153–161. ISSN: 0003-6870. DOI: https://doi.org/10.1016/j.apergo.2018.01.009. URL: https://www.sciencedirect.com/science/article/pii/S0003687018300176 (cit. on p. 3).

[4] Hao Feng, Cuiyun Li, Jiayu Liu, Liang Wang, Jing Ma, Guanglei Li, Lu Gan, Xiaoying Shang, and Zhixuan Wu. «Virtual reality rehabilitation versus conventional physical therapy for improving balance and gait in Parkinson's disease patients: A randomized controlled trial». en. In: *Med. Sci. Monit.* 25 (June 2019), pp. 4186–4192 (cit. on p. 6).

[5] Pinar Tokgöz, Susanne Stampa, Dirk Wähnert, Thomas Vordemvenne, and Christoph Dockweiler. «Virtual reality in the rehabilitation of patients with injuries and diseases of upper extremities». en. In: *Healthcare (Basel)* 10.6 (June 2022), p. 1124 (cit. on p. 6).

[6] Amanda Vitória Lacerda de Araújo, Jaqueline Freitas de Oliveira Neiva, Carlos Bandeira de Mello Monteiro, and Fernando Henrique Magalhães. «Efficacy of virtual reality rehabilitation after spinal cord injury: A systematic review». en. In: *Biomed Res. Int.* 2019 (Nov. 2019), p. 7106951 (cit. on p. 6).

[7] Rachel Kizony, Noomi Katz, and Patrice Weiss. «Adapting an immersive virtual reality system for rehabilitation». In: *Journal of Visualization and Computer Animation* 14 (Dec. 2003), pp. 261–268. DOI: 10.1002/vis.323 (cit. on p. 6).

[8] Felix Clay, David Howett, James FitzGerald, Paul Fletcher, Dennis Chan, and Annabel Price. «Use of immersive virtual reality in the assessment and treatment of Alzheimer's disease: A systematic review». en. In: *J. Alzheimers. Dis.* 75.1 (2020), pp. 23–43 (cit. on p. 7).

[9] Chimezie O Amaefule, Stefan Lüdtke, Thomas Kirste, and Stefan J Teipel. «Effect of spatial disorientation in a virtual environment on gait and vital features in patients with dementia: Pilot single-blind randomized control trial». en. In: *JMIR Serious Games* 8.4 (Oct. 2020), e18455 (cit. on p. 7).

[10] David Howett et al. «Differentiation of mild cognitive impairment using an entorhinal cortex-based test of virtual reality navigation». In: *Brain* 142.6 (May 2019), pp. 1751–1766. ISSN: 0006-8950. DOI: 10.1093/brain/awz116. eprint: https://academic.oup.com/brain/article-pdf/142/6/1751/28706714/awz116.pdf. URL: https://doi.org/10.1093/brain/awz116 (cit. on p. 7).