FINAL DEGREE PROJECT

Biomedical Engineering Degree

Industrial Electronic and Automatic Engineering Degree

VIRTUAL REALITY APPLICATIONS DESIGN FOR PHYSIOTHERAPY AND PHYSICAL TRAINING

Memory and Annexes

Authors: Robert Gil Martín
Eva Tinoco Mascaró

Director: Jordi Torner Ribe

Co-director: Gil Serrancolí Masferrer
Francisco Alpiste Penalba

June 2020

RESUM

Els estudis i les evidències dels avantatges que aporta la realitat virtual en teràpies de rehabilitació son cada vegada més abundants. En aquest projecte, amb la col·laboració de l'Associació de Diversitat Funcional d'Osona (ADFO), s'ha dissenyat dos jocs amb l'objectiu de crear aplicacions de realitat virtual per millorar la rehabilitació dels pacients després de patir un ictus.

El problema principal de les teràpies de rehabilitació és que es fan molt monòtones pels pacients, ja que els exercicis solen ser repetitius d'una sessió a l'altre. Gràcies a la creació d'aquests jocs, aquest problema es redueix. A més, disminueix la saturació de pacients a les consultes i, el fisioterapeuta pot tenir una base de dades amb un altre tipus de resultats, tant a nivell tècnic com visual.

Un dels jocs creats està dissenyat per fer-lo servir amb l'ordinador i el ratolí, ja que no es disposava d'equips de realitat virtual (SteamVR HTC Vive) en els domicilis pertinents. Tot i així, aquesta aplicació pot adaptar-se a una configuració en realitat virtual si en un futur es volgués utilitzar el joc en aquest sentit. L'altre joc creat està dissenyat exclusivament per ser una aplicació en realitat virtual. Per fer-ho, s'han utilitzat uns dispositius mòbils amb unes aplicacions de realitat virtual per poder simular les ulleres i els controladors. No és exactament el mateix que utilitzar un equip real de realitat virtual, però tot i així han estat de gran utilitat.

Per altra banda, les proves de les aplicacions no s'han pogut realitzar amb els pacients de l'ADFO. En un futur es té previst fer-ho per avaluar millor la funcionalitat de les aplicacions creades. Tot i així, s'ha obtingut algunes opinions de persones sanes que sí que han pogut provar aquests jocs i s'ha pogut rectificar en funció d'això.

RESUMEN

Los estudios y las evidencias de las ventajas que aporta la realidad virtual en terapias de rehabilitación son cada vez más. En este proyecto, con la colaboración de la *Associació de Diversitat Funcional d'Osona* (ADFO), se han diseñado dos juegos con el objetivo de crear aplicaciones de realidad virtual para mejorar la rehabilitación de los pacientes después de sufrir un ictus.

El problema principal de las terapias de rehabilitación es que se hacen muy monótonas por el paciente ya que los ejercicios suelen ser repetitivos de una sesión a la otra. Gracias a la creación de *serious games* (juegos cuyo propósito principal es distinto al entretenimiento) este problema se reduce. A demás, disminuye la saturación de pacientes en las consultas y, el fisioterapeuta puede tener una base de datos con otro tipo de resultados, tanto a nivel técnico como visual.

Uno de los juegos creados está diseñado para usarlo con el ordenador y el ratón, ya que no se disponía de equipos de realidad virtual (SteamVR HTC Vive) en los domicilios pertinentes. Sin embargo, esta aplicación puede adaptarse a una configuración en realidad virtual si en un futuro se quisiera utilizar el juego en ese sentido. El otro juego creado sí que está diseñado exclusivamente para ser una aplicación en realidad virtual. Para ello, se han utilizado unos dispositivos móviles con unas aplicaciones de realidad virtual para poder simular las gafas y los controladores. No es exactamente lo mismo que usar un equipo real de realidad virtual, pero aun así han sido de gran utilidad.

Por otra parte, las pruebas de las aplicaciones no se han podido realizar con los pacientes de la ADFO. En un futuro se tiene previsto hacerlo para evaluar mejor la funcionalidad de las aplicaciones creadas. Aun así, se ha conseguido algunas opiniones de personas sanas que si han podido probar estos juegos y se ha podido rectificar en función de ello.

ABSTRACT

Studies and evidence of the benefits of virtual reality in rehabilitation therapies are increasingly abundant. In this project, with the collaboration of the *Associació de Diversitat Funcional d'Osona* (ADFO), two games have been designed with the aim of creating virtual reality applications to improve the rehabilitation of patients after suffering a stroke.

The main problem with rehabilitation therapies is that they become very monotonous for the patient since the exercises are usually repetitive from one session to the other. Thanks to the creation of our games this problem has been reduced. In addition, the saturation of patients in the consultations will decrease and the physical therapist may have a database with other types of results, technically and visually.

One of the games created is designed to be used with the computer and mouse, because virtual reality equipment (SteamVR HTC Vive) was not available in the relevant homes. However, this application can be adapted to a virtual reality configuration if in the future it wants to be used in this aspect. The other game created is designed exclusively to be a virtual reality application. To make this possible, mobile device with virtual reality applications have been used to simulate headset and controllers. It is not the same as using a real virtual reality equipment, however they have been especially useful.

On the other hand, testing of the applications could not be performed with ADFO patients. It is planned to do it in the future for a better evaluation of the functionality of the applications created. However, some opinions have been obtained from healthy people who have been able to try these games and have been able to rectify based on that.

ACKNOWLEDGMENTS

During the period in which we have been working on this project, certain unforeseen events have happened that we would not have imagined at the final degree project beginning, such as a worldwide pandemic. That is why we find it very necessary to thank the help of certain people who have made our journey more pleasant and enjoyable in these difficult days. People who have allowed us to carry out the TFG in the best possible conditions.

On the one hand, we would like to thank to *Universitat Politècnica de Catalunya* (UPC) for the opportunity to offer us to work in this project. Special thanks to the teachers who have accompanied us in the last months of our degree to guide and advise us in this project: Gil Serrancolí, Jordi Torner and Francesc Alpiste.

Secondly, we would like to thank the *Associació Diversitat Funcional d'Osona* (ADFO) for collaborating with the UPC in this project and being able to give us feedback on the games created. But most of all, we are grateful that they helped us suggesting possible solutions when we had to confine ourselves to our homes to continue our project.

On the other hand, we would also like to thank the help of our colleagues, Berta Mayans and Alix González, who are carrying out a project similar to ours, because we have been able to resolve common doubts with each other. We would also like to thank Matteo Mezzanotte, an Erasmus student, for helping us unify all the games created in the existing application that past students created.

~ Eva Tinoco ~

I also want to thank my family and my partner Guillem Pétriz for their support throughout the engineering degree, and especially in these recent months when we have had to be confined at our homes. Thanks to their daily encouragement, I have been able to work in the project in a telematic way with more positivism. And thanks to the trust they have had in me, they have made me feel immensely proud of the amount of work done in this project.

~ Robert Gil ~

Personally, I would like to thank my family who had to suffer my ups and downs during the process of this project and make me feel more secure in front of adversities.

Of course, the person who listened more my failures and my successes was my partner Andrea García, and without her positivism and patience this experience could have been quite more difficult.

Finally, but not less important, I would like to thank to my university colleague Rachid Boukir. He has been my travel companion during these four long years where we have supported each other in difficult times and where we also celebrate our course passes. I am quite sure his career will be magnificent.

GLOSSARY

- ADFO: *Associació Diversitat Funcional d'Osona*
- CT: Computed Tomography
- CVA: Cardiovascular Accident
- Game object: Base class for all entities in Unity Scenes.
- MRI: Magnetic Resonance Imaging
- Prefab: Customized game object.
- TFG: *Treball Final de Grau* (Final Project Degree)
- UPC: *Universitat Politècnica de Catalunya*
- VR: Virtual Reality

INDEX

1. PREFACE

In the beginning, the main use of virtual reality was focused only on entertainment. Over time, and with constant improvements in both technology and medicine, great progresses have been made. This has allowed technology to share knowledge, techniques and resources with medicine and vice versa. This is where virtual reality begins to have medical applications, such as the implementation of virtual reality in the rehabilitation of people who have suffered a stroke in order to improve the motor function of their limbs; application on which this project is mainly based.

Figure 1.1: Virtual reality in the healthcare industry.
[https://frontiersinblog.files.wordpress.com]

1.1 Project origin

Recovering from a stroke has always been an enigma because some patients have a medical profile that differs from others. That is why many treatments could either work (partially work) or be ineffective. One of the most challenging aspects is the psychomotricity one, where the patient finds it harsh to recover the mobility of the muscles. However, it can be solved by applying a certain therapy to the subject. There are conventional physical therapies, but this project implements the physical therapy with virtual reality (VR). And here is where this project was born. Being a continuation of previous TFGs, this project consists on implementing VR as a rehabilitation therapy to the patient.

In addition, due to the confinement for the COVID-19, it was forced to restructure the focus of the project because the corresponding equipment to create the games was at the university and it was not possible to use it. However, following the recommendations and conditions proposed by the *Associació de Diversitat Funcional d'Osona* (ADFO) the project was carried out. ADFO proposed the implementation of the use of the mouse in the game in order that patients could make certain movements following trajectories dragging the mouse with their hands and consequently moving the upper limbs.

1.2 Motivation

There are several reasons why so much interest and motivation was in the creation of two games with the goal of helping people who suffered a stroke to rehabilitate.

The first motivating reason is that using VR can be challenging, as this kind of technology is under development, so there is not much research that has been done about the subject. And everything that is challenging, is encouraging. It is so satisfying working with a really innovative and developing technology such as VR.

Another reason is being able to help people with physical disabilities recover more quickly and effectively. The main idea of this whole project is to create a VR application that helps the patient to do specific movements in order to recover the sensitivity of the muscles. Moreover, this would be an advantage for the physiotherapist. It could reduce their amount of work and avoid saturation of people in consulting rooms.

Another reason is that conventional therapies may be hard to follow, as one of the most challenging aspects of the disease is the difficulty of remembering how to do the most basic and essential movements on daily basis. Using VR will make the interface of the activity look more intuitive so many patients may find it really helpful and easy to get through the therapy. Furthermore, in many cases when the conventional therapies are used, the patient loses motivation or interest due to the exercises become too repetitive and it can also be frustrating not seeing the progress done. Through this project, it will be possible to assess how much progress has been made from one therapy to another, and it will also have a pleasing and entertaining environment.

Lastly, the aim is to improve not only the rehabilitation processes but also the collection of data for subsequent analysis.

1.3 Prerequisites

On one side, there are some necessary tools that have been used to carry out the project:

- **PC:** running Windows 7 or higher version.
- **Unity:** video game graphic design platform.
- **Visual Studio:** code editor (C# in this case).
- **Electronic devices:** three Android mobile phones.
- **Computer accessories:** mouse, keyboard.
- **Plugins:** Steam VR and VRTK Toolkit.

On the other hand, and the most important thing for the correct development of the two games of the application that is created in this project, a previous knowledge is needed: a good command of the use of the Unity video game creation program, which also means that the user has a good level of the C# programming language, which is the one used to be able to program the fundamental scripts for the Unity project.

2. INTRODUCTION

This work explains how the process has been from the beginning to the end of the design and creation of virtual reality applications based on rehabilitation exercises for patients who have suffered a stroke. To do this, steps have been followed that were stipulated at the beginning of the work in order to carry out all the objectives stablished. The following of the different work sections shows how it has evolved.

The first part is based on an information searching phase where all the theoretical concepts to apply in a practical way throughout the project are found. In this way, it will have a good academic base to carry out this work.

Secondly, there is a more detailed explanation about these two applications related with the patient’s rehabilitation who have suffered from stroke disease. All this information is reproduced in a very illustrative way to have a clearer idea of how each part of the games are made.

And finally, a series of tests and analyses are carried out to verify that our project has managed to meet the desired minimum objectives. Due to the state of alarm, caused by COVID-19, the trials will be carried out by relatives of the authors of this project in their respective homes, instead of ADFO patients.

At the end of the project are the annexes A, B, C and D where all the scripts have been attached, an explanation of how the games are run and a survey template that was used to ask to the family how was their experiences.

2.1 General objectives

As it has been mentioned, this project's main objective is to help those people with motor difficulties to exercise certain movements with a methodology that is more enjoyable for the patient and is not a simple conventional rehabilitation.
Seeing as the technological world advances and the applications that are developed for multiple applications, our contribution with this work enters this creative and innovative way. By complying with the prerequisites for this project, it is possible to carry out this objective through a correct work methodology.

On the other hand, it is expected that this work will fulfil the objective of creating a greater ease of work, by the ADFO professionals, in analysing some patient results. Having a greater field of work, thanks to these applications, this will serve them to keep tracking the state of the patient's mobility.

2.2 Specific objectives

Going more deeply, to carry out these applications, there are these following points as specific objectives:

- Have the main idea of the purpose of both applications. That is, how the idea of adapting specific physical movements to a virtual reality game is going to be faced. In this case, these movements can be chosen before carrying out the exercise by the user.
- Design and create the previously stipulated environments, with a logical and not ostentatious sense. To avoid the patient having a feeling of a highly charged environment. And being as realistic as possible.
- Create an avatar or hands that simulate their movements within the game, through the controllers and the headset.
- Create scripts with C# language oriented to the objects programming, to introduce some mechanics and animation to those game objects that will be relevant in the course of the activity.
- Create sequences and certain events that are caused due to the patient's actions inside the game.
- Collect information extracted at the end of the game process, by exporting the reports, for subsequent analysis.

2.3 Scope of work

The final scope of this project is to carry out two therapeutic applications in order to break with the traditional methodologies. That means that our goal in this final work degree is to be able to make the same kind of rehab exercises in a more pleasant way, in this case playing with some fun VR games. To accomplish that, at the end of this project these two applications need to satisfy the stipulated requirements, commented previously.

This work will be finished when the applications work perfectly and they are playable, proving with some tests with our families. One more thing to check, before finishing the work, is to be able to extract valuable information so that the applications make sense and have a purpose that is not just playing a game.

2.4 Contingency plan

It was necessary to create a contingency plan given the situation of COVID-19. A contingency plan is a plan designed for a different outcome than the usual plan in order to manage an exceptional risk situation. The identified risk of this contingency plan is a global pandemic. The contingency plan:

Threat: global pandemic (COVID-19).

Impact:

- ➢ Impossibility of teamwork.
- ➢ Compulsory teleworking.
- ➢ Impossibility of using the devices of the university laboratory. This includes the following devices: Kinect sensor, virtual reality equipment (Steam VR HTC Vibe), powerful computer to design games in good conditions, a large computer screen for greater work comfort.
- ➢ Need to redo the objectives and methodology of the project.

The contingency plan will contain the following **countermeasures**:

- ➢ Technical measures:
    - o New replacement equipment in order to carry out the work. These new devices are: one computer for each of the members of the work team (a total of two computers) and three Android mobile phones to simulate the headset and drivers of the virtual reality equipment Steam VR HTC Vibe.

- Organizational measures:
  - Telecommuting.
  - Repetitively backups to avoid losing the work done. Working on the platform of Unity causes the saturation or blocking of those less powerful computers. This causes the crash of the computer or the shutting down of the Unity program. For this reason, it is necessary to make different backups of the work.

- Human measures:
  - Additional training to learn the appropriate C# programming language in order to be able to design the application towards the new goal.
  - New assignment of roles and responsibilities to the members of the working group.

Recovery plan:

- Fix of the possible errors in the scripts of both games.
- Adaptation of the games created to the new incorporation of virtual reality equipment Steam VR HTC Vibe. This includes:
  - Modification of the corresponding scripts.
  - Modification of the position of the objects in the scene. Situate them in a comfortable place for the player's view when he is using the headset. Otherwise, if the objects are too far apart, the player will have to move his head from side to side to be able to see all of them.

3. STATE OF THE ART

3.1 Virtual Reality

Virtual Reality (VR) is a computer technology that simulates an environment, previously created, by a 3D interface in which the user interacts. Inside those interfaces, the user experiences new sensations which are caused by these high-quality applications. These applications can be like realistic or fantasy worlds, depending on the developer goal. Visual simulation is not the only which this technology can reach, it can also simulate sounds, touch, and smell effects.

3.2 Virtual reality history

The first time that the virtual reality concept appears was in 1935, however Sir Charles Wheatstone was the first person to describe stereopsis concept in 1838 [1]. Stereopsis is a term which describes the perception of depth in a 3-dimensional environment. This happens when using a binocular vision projects different images in your eyes with another perception. So, this technology allowed Wheatstone to combine the reflection of two images in a single image. His prototype was made by two mirrors inclined 45 degrees from the user’s eyes and they were reflecting an image from the side, creating a new perception.

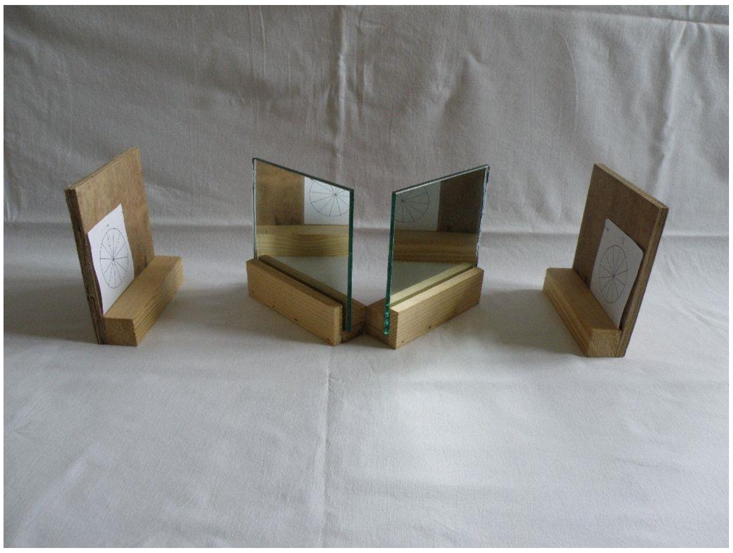


Figure 3.1. The Wheatstone mirror stereoscope.
[https://artefactual.co.uk/]

As mentioned, in 1935 the first fictional VR model was presented for a movie. This movie was based on a googles invention by a professor who said that you could be immersed in a movie when you wear them. You could have new senses when you try that invent like sights, sounds, tastes, smells, and touches. You could interact with new characters as well. Even without knowing what Virtual Reality is, this creative story was well on the way.

Once seen how a theoretical concept of virtual reality was conceived, let us move on to the first physical appearing of this concept. This moment appears in 1956 in EEUU by the cinematographer Morton Heilig. He created the first VR machine which was combined by different types of technology to simulate as much senses as possible. This machine was like a huge booth where was able to fit up to four people. It combined a 3D video, audio, vibrations, smells, and atmospheric effects. The screen that Heilig used to create this machine was a 3D stereoscopic scene. He used the theoretical concept that Wheatstone introduce in his moment. For the other effects, he used stereo speakers, a vibrating chair and scent producers to simulate the smells [2].

Heilig said "Sensorama is the cinema of the future", he wanted to immerse people on his films and create new experiences. As you can see in figure 3.2, the concept of the cinema exists in this product, but this is in a smaller scale if it wants to apply it to involve more people.

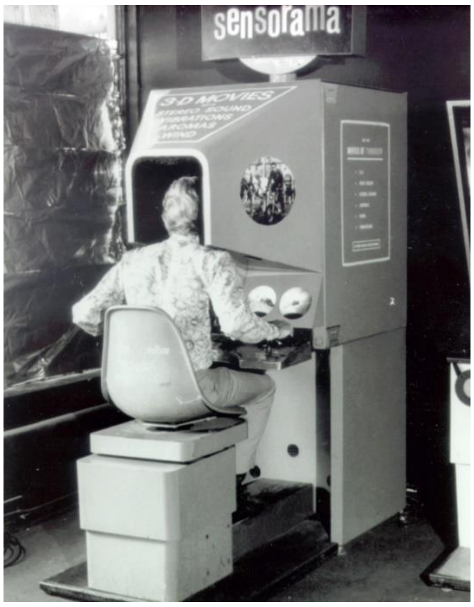


Figure 3.2 The Sensorama VR machine.
[https://www.researchgate.net/]

In 1960s, Heilig patented this VR machine, but this was not the only thing he made. He also, patented the first head-mounted display (HMD) that everyone all knows nowadays as the main device to virtual reality immersions. That device allowed people to get into a 3D world just putting on the headset, however there was not a tracking motion yet.

Initially, the headtracking was implemented in 1961 by Comeau and Bryan, two engineers who implemented this technology to a military application [3]. Using a camera that simulates the head movement provided the military to look around and detect possible dangers.

It would not be until 1965 when Ivan Sutherland create the concept to include a computer hardware to the virtual reality system and in this way allow running the applications in real time [3]. Adding this technology, it will be possible to be more realistic in these virtual environments to a point that cannot be differentiated from reality. As Sutherland said: "With appropriate programming such a display could literally be the Wonderland into which Alice walked" [3].

It was in 1968 when Sutherland and Bob Sproull, his student, create physically the first virtual reality HMD. However, it was not too comfortable for the people who had to test its functionality. As you can see in figure 3.3 there is a man testing this headset which is connected to a computer from that time. Obviously, that VR system was not the optimal to work with, but it was a good start.

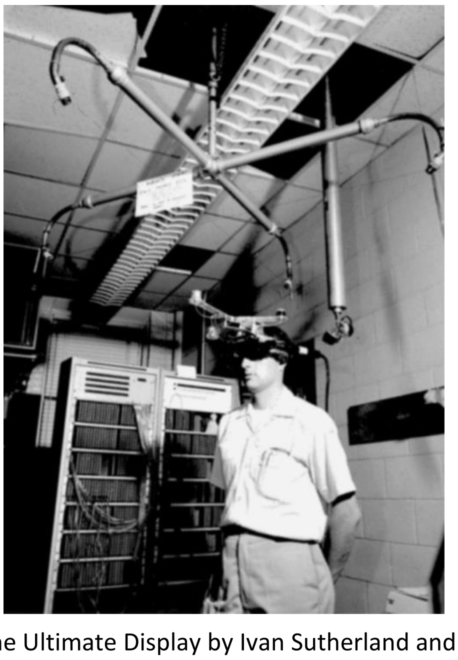


Figure 3.3. The Ultimate Display by Ivan Sutherland and Bob Sproull. [https://proyectoidis.org/]

During the 60s and 70s decades some simulators were invented with this VR system to provide to military to recreate real cases like a plane flight.
Also, General Electric Corporation implemented this flight simulator using more than one screen to surround the user in a 180-degree configuration.
The inflection point was produced in 1975 when it was created the first VR platform where it simulated the user's avatar with the same size and performed the same moves that he or she was doing. This platform was called VIDEOPLACE by Myron Krueger [3]. He was a computer artist who developed these artificial environments using computers and video systems. This platform is composed by computer graphics, video projectors, cameras, screens, and some position sensors to detect the user moves. All this setup was inside a dark room like a cinema room to be able to show all the images by the projector to the screen. Figure 3.4. shows how was it mounted and how it would look like.

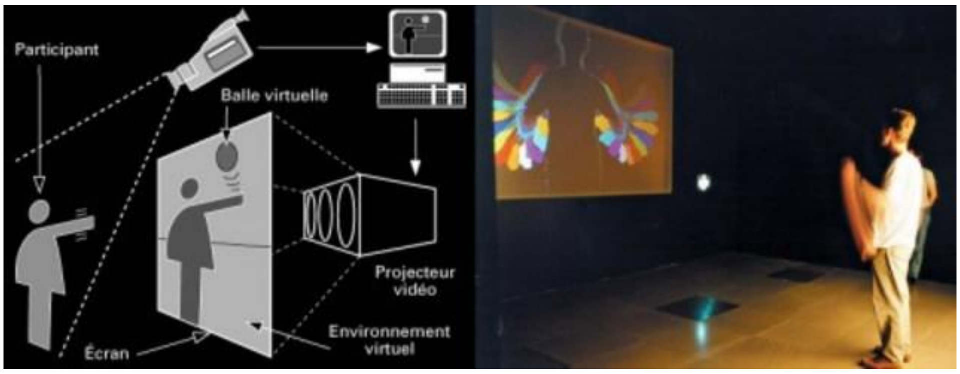


Figure 3.4. The Krueger's VIDEOPLACE platform.
[https://www.timetoast.com/]

The MIT used this previous first-person interactivity to travel through the Aspen city streets using images taken from Google Street View. Today, we all know that this interaction is possible in Google Maps pressing a simple button.

Once invented the head tracking, what should be next? The hand tracking with wired gloves would be the next step and Sandi and Defanti found it the way to achieve that. Using gloves with photocells in the fingers that they generated electrical signals with the light variance coming from lights emitters. Due to these signals, it was possible to track the user's hand gestures. The first application was also in a flight simulator by the military forces. However, the first company who started joining both technologies, HMD, and the gloves, was VPL Research, Inc. They started developing this kind of VR equipment and selling them as a unique setup.

In 1980s there were multiple new applications for these setups, but there is one that stands out more than the others. This application was a simulator for the astronauts training. Scott Foster was hired by the NASA to develop the 3D audio processing in real-time. In Figure 3.5. it can be observed how the headset were becoming more sophisticated.

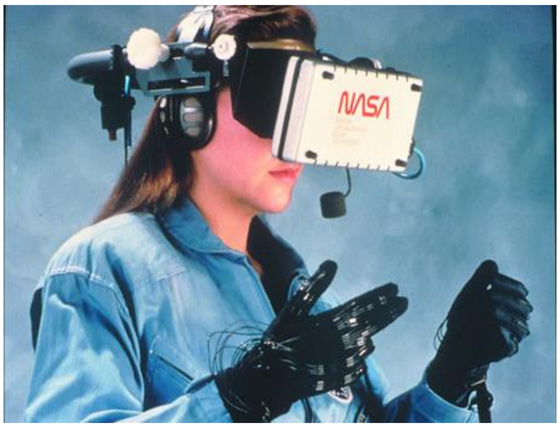


Figure 3.5. NASA's astronaut simulator.
[https://www.pulseheadlines.com/]

For the moment, all the VR applications were based on simulators the most. But, what about VR games which is commonly known by most people? That would not be until 90s decade where Jonathan Waldern showed a VR arcade machine at the Computer Graphics in London [4].

The videogames pioneer company to start designing VR game cockpits was SEGA. SEGA had tons of famous arcade games whose could be translated to VR games. So, this company released their first arcade simulator in 1994 called SEGA VR-1 and the headset CyberMaxx. These setups were placed basically in gaming rooms because it was too expensive and too big to buy it as a private equipment. However, the company Nintendo started developing in this new world technology creating the first VR portable gaming console, called Nintendo Virtual Boy. Although, there were bad news for the Nintendo people cause this product was not good enough, it was too weak in 3D colour graphics, software support and it was not too comfortable.

Videogame industry got stacked for a while and it would not be until 2010 when the first Oculus Rift headset prototype appeared. It was created by Palmer Luckey an 18-year-old entrepreneur and it was used to test the Google Street View in a stereoscopic 3D mode. It only had a 90-degree field of view and it works attached an image processor by a computer which it was delivering the images with much power than previous cases.

This guy earned $ 2 billion dollar selling this product to Facebook, and when that happened all the Facebook competitors wanted to get in this market. And using the famous reverse engineering some important companies appeared with their own products. Sony announced Project Morpheus, a headset to use it with PS4, which was a new product in that moment as well. Also, Samsung showed us the Samsung Gear VR to use it with their new mobile phone device the Galaxy. Google created the cheapest stereoscopic viewer called Cardboard which is made with cartoon and two glasses [5].
Between 2014 and 2016 there were lots of companies who started developing VR products, a part of these famous companies commented, and there were more fields where this technology could be used, for example, the BBC created a 360-degree video which allowed see Syrian migrant camps [6].

In 2016 there was a company that develop a revolutionary VR product which it allowed users to be detected when they were moving freely in a specific space. The headset and the hand controllers were using some tracking sensors which sent position information and that information was processed and refreshed to the headset. That company was HTC, and the product today is known as HTC VIVE and they collaborate with one of the biggest known videogame's platforms, Steam. As you can see in the image, the two circled components are the responsible to track the whole room and, in this way, they also verify in real-time the user's position.

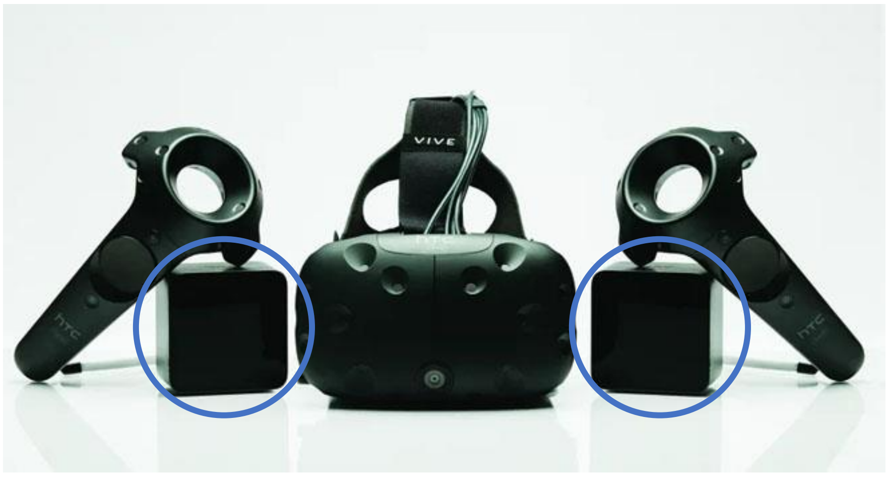


Figure 3.6. HTC VIVE SteamVR.
[https://www.geektopia.es]

At the present, mobile VR is declining rapidly because the devices such as the Oculus GO are very affordable and there are more benefits. And the newest update that from this sector is that now it is possible to track the hands without gloves or controllers, just with a camera in front of the headset, like Kinect from Xbox does.

3.3 VR modalities

This section introduces the existing modalities in VR. Nowadays, it is possible to find five different VR formats, which are: non-immersive, Fully Immersive, Semi-Immersive, Augmented and Collaborative.

3.3.1 Non-immersive Virtual Reality

Its name defines completely the meaning of this modality. This is based on virtual applications where the user is not immersed in them. This kind of VR is already known all over the world because everybody can use it in their houses just using a screen, headphones, and controllers (like the mouse or a console). The user interacts with the characters inside the virtual environment, but they won't feel any sense like a fully immerse VR application. Commonly, the most used applications are the conventional videogames that everybody can play in their houses with a PC or a console like PS4 or Xbox.

One example of these videogames is Call of Duty, a shooter game where the user is controlling a soldier in a war situation. This kind of games are useful for the army as well to implement new strategies and how to tackle some situations before experiencing this in real life.

3.3.2 Fully immersive Virtual Reality

Totally opposite to the previous case, there is the fully immersive virtual reality which it will give a realistic experience as if the user were physically inside a virtual world. Here there is a situation where it needs to use a different kind of equipment, like the helmets, gloves or a hand-tacker and some position sensors which will determine our movements. Obviously, a good computer is needed in this case, because these apps require good graphics to be in a more realistic environment.

In recent years, this type of VR has been applied in virtual media rehabilitations and trainings. Before becoming a surgeon or any other medical specialist these trainings with VR applications will be so useful for them to take less risks in a real intervention and it will also be a cheaper way to improve these skills.

3.3.3 Semi-immersive Virtual Reality

In this case, there is an intermediate point between both previous modalities, which is the semi-immersive virtual reality. Here there are some mixed qualities from non-immersive and fully immersive VR applications. The user is immersed in the virtual world like the fully immersive mode, but there is no motion tracking, i.e., there are not the physical moves from the user represented in the virtual world. This feature comes from the non-immersive mode, using any controller, with a gyroscope support, to direct the character and perform any action. So, this is another way to get into a virtual world without setting a whole space or room with the position sensors to detect the moves. It is obviously a cheaper setup and most commonly used among the rest of modalities after the non-immersive.

This is the modality that this project uses in our games to prove them with patients who suffered a stroke. HTC VIVE SteamVR is made up by a helmet where the user gets immersed into the virtual world and two hand-controllers as the motion detection.

3.3.4 Augmented Virtual Reality

Maybe this is the most curious VR mode because the interactions are done in the real world. Using a screen and a camera (e.g. a mobile phone) there is the possibility to include objects, characters, or simulate environmental changes in a real place. This VR mode is used to interact with another people which is around you. For example, one of the most revolutionary games for that moment was Pokémon Go. It was possible to catch the Pokémon seeing them in your mobile phone screen and throwing a pokeball to it.

3.3.5 Collaborative Virtual Reality

This modality is like the semi-immersive one, it is based to collaborate with other players in the same virtual world using VR equipment but using hand controllers. The users can control their characters with the controllers, and they can speak, see, hear, and interact through different actions between them. There are companies that use this method to create meetings with the employees or subsidiaries due to the distance. It is also used for video games like the very known shooter game PUBG (Players Unknown Battle-Ground), which is a battle royal game where there are 100 players and only 1 can reach the victory, so they meet in a virtual world and try to eliminate their characters to survive.

3.4 VR applications

Nowadays, the VR application field is very extensive, but it can highlight different fields where this technology is used the most. These VR applications are VR in military, VR in sport, VR in mental health, VR in physical-motor rehabilitation, VR in medical training, VR in education and VR in fashion. VR military and health applications are already commented in the previous points, but there are other fields where their applications are quite interesting.

3.4.1 VR in Sport

For all the people who does not like train in a gym or do not have enough time during the day, this is another variant to do some exercise. Using this technology, it is possible to train with a personal trainer without being in the same room. It also helps to follow your physical and mental process during these sessions, for example, an athlete which is injured can use this VR applications to improve his/her cognitive abilities.

Now there are some broadcasters, who stream in live some sport events by VR games to get the attention of more people who are interested in participating. So, the people who do not want to spend too much money to be a part of these kind of events this is another way to enjoy them. For example, there is the possibility to be sit in a soccer stadium with no need to pay the whole ticket price, as you can see in this next image.

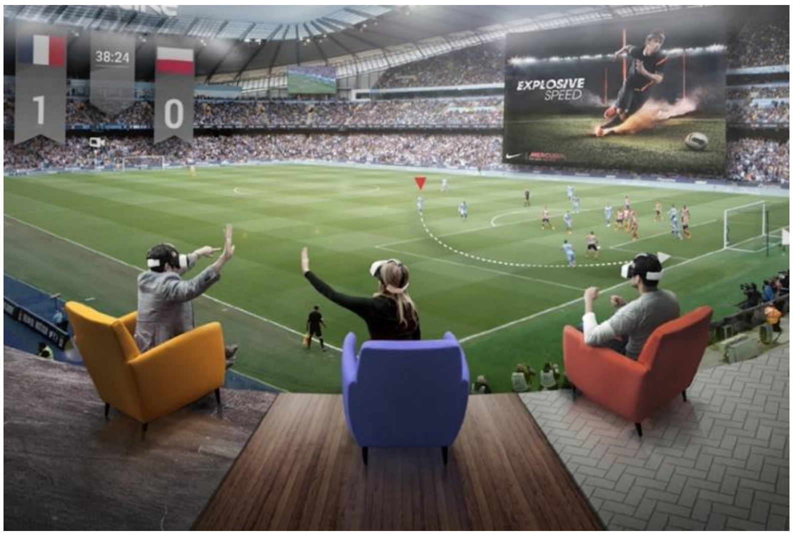


Figure 3.7. Live soccer match in a virtual reality application.
[https://www.cbronline.com/]

3.4.2 VR in Education

This technology can help people to learn new skills, respectively. So, schools have adopted these VR applications to provide another kind of education to the students. Now, there is the possibility to visit a museum staying in the classroom, for example. Using a collaborative method in VR allow students to interact with each other.

This is a teaching methodology that improve the interest from the student to learn more about new things and they can visualize in a better way the subjects they are learning. Sometimes, there are subjects that can be too abstract and difficult for the student to understand, for example when the teacher is talking about the space, the planets and so on. In figure 3.8 there is a kid using a VR app which allows him to travel through the space.

Figure 3.8. A kid learning about the space in a virtual reality application. [https://medium.com/]

3.4.3 VR in fashion

It has arisen a new way to visit a clothing store in a virtual way because the retailers want to show their products and reach more people. In this manner, there is no need to pay for advertising and spend too much money in resources.

This kind of applications are based in a 360-degree experience to make sure that everything is shown. Big fashion companies like Tommy Hilfiger has used this technology allowing people to participate in its fashion shows.

3.5 Description of stroke pathology

Stroke is a cerebrovascular injury. It occurs when there is a rupture or obstruction in a blood vessel, reducing the flow of blood to the brain. When the blood does not reach the brain in the necessary amount, the nerve cells do not receive oxygen and they stop working. Moreover, the function of the part of the brain that has been affected can be temporarily or permanently altered.

Figure 3.9. Example of a part of the brain affected by a stroke.
[https://www.clinicbarcelona.org/]

3.5.1 Types of strokes

There are two types of stroke depending on the mechanism of the injury:

- **Ischemic stroke:** caused by obstruction of blood flow, usually by a blood clot or thrombus. The thrombus partially or totally limits blood flow, decreasing the amount of oxygen that reaches the brain. This problem is usually caused by the development of fat deposits on the vessel walls, which is called atherosclerosis. The ischemic stroke, which is the most frequent of the cases, affects approximately 80% of the cases.
- **Hemorrhagic stroke (or cerebral hemorrhage):** the rupture of a vessel causes the outflow of blood and compression of structures of the central nervous system. The hemorrhagic stroke is the least frequent of the cases, affects between 10 and 15% of the cases.

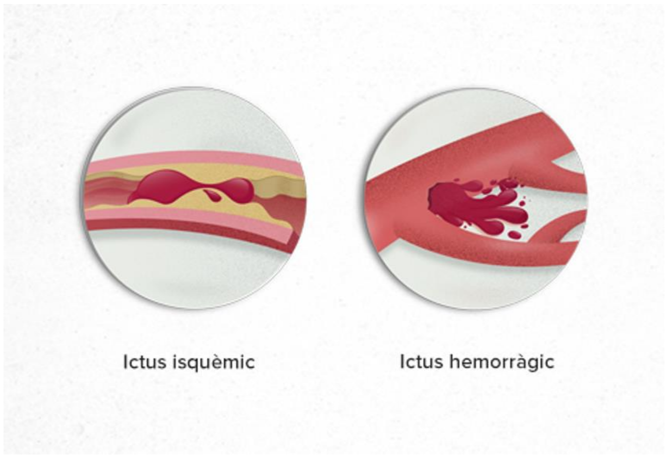


Figure 3.10. Types of strokes. Ischemic stroke (left) and hemorrhagic stroke (right). [https://www.clinicbarcelona.org/]

3.5.2 Causes

There are risk factors that lead part of the population to have a higher risk of suffering a stroke. Some of them cannot be controlled but there are others that can be changed, processed, or modified.

Some of the main risk factors are:

- Being over 55 years old. The risk is higher as age increases.
- Family medical history. Genetics is an uncontrollable risk factor.
- The same number of strokes occur in both genders. However, more than half of the deaths are in women.
- Having recently suffered a stroke. Once you have had a stroke, your chances of having another stroke increase considerably.
- Arterial hypertension.
- Mellitus diabetes.
- Having a high level of cholesterol.
- Consumption of alcohol, tobacco, or drugs (amphetamines, cocaine, etc.).
- Sedentary lifestyle.
- Obesity.
- High red blood cell count. The reason is that red blood cells cause the blood to thicken, which an increase in red blood cells can cause clots more easily.
- Present heart disease: Heart rhythm disorders (arrhythmias, and especially atrial fibrillation), dilation of the heart cavities, or alterations in the heart valves, can cause blood clots inside the heart. The thrombus can travel through the arteries to the brain and once there obstruct one of the arteries and cause a stroke.

3.5.3 Symptoms

Most brain functions are well located in different areas. Therefore, the symptoms of a stroke depend more on the location of the alteration in the blood supply than on the cause that generated it. A stroke is likely to occur if you experience the following symptoms:

- Loss of strength in one half of the body (face, arm, and leg on the same side).
- Difficulties in speaking.
- Loss of sensation or tingling in one half of the body.
- Sudden loss of vision in one eye or double vision.
- Very intense headache other than usual.
- Loss of balance and coordination.

Some ways of knowing when a person will have a stroke have been developed. There is, for example, the Cincinnati scale, which consists of three checks:

- **Facial asymmetry:** The patient is made to smile to check if both sides of the face move symmetrically. If abnormal, one side would show difficulties to move.
- **Arm strength:** The patient is instructed to stretch the arms for 10 seconds. In an abnormal case, one arm does not move or falls in comparison to the other arm.
- **Language:** The patient is instructed to speak. If abnormal, it drags the words, has trouble speaking, or does not speak.

If any of these three tests obtain the abnormal result, it is possible that the patient will suffer a stroke.

3.5.4 Diagnosis

The arrival of the patient at a hospital center in the first 6 hours after the start of the stroke is essential to reduce complications by 25-30%.

Usually, the doctor can diagnose a stroke through the history of the events and physical examination. The physical examination helps the doctor to determine where the brain injury is located.

Imaging tests such as computed tomography (CT) or magnetic resonance imaging (MRI) are also often performed to confirm the diagnosis, although these tests only detect strokes after a few days have elapsed.
A CT or MRI is also effective in determining whether a stroke has been caused by hemorrhage or a brain tumor. An angiogram may be performed by the physician in the unlikely event of surgical intervention.

The doctor tries to establish the exact cause of the stroke. It is especially important to determine if it was caused by a clot that lodged in the brain or by the obstruction of a blood vessel due to atherosclerosis.
If the cause is a clot or an embolism, another stroke is highly likely to occur, unless the problem is corrected. In this situation, the doctor usually performs an electrocardiogram (to detect an arrhythmia) and may also recommend other heart study tests such as Holter.

Although other laboratory tests are not very useful, they are also done to confirm that the stroke was not caused by a lack of red blood cells (anemia), an excess of red blood cells (polycythemia), a cancer of the white blood cells (leukemia) or an infection.

Figure 3.11 and 3.12: Computed Tomography Scan (left); Magnetic Resonance Imaging Scan (right).
[Left: https://en.wikipedia.org/wiki/Main_Page ; Right: https://www.sah.org.au/]

3.5.5 Treatment

In the acute phase, the patient with stroke should be attended in a Neurology service, preferably with a Neurorehabilitation Unit. This decreases mortality and improves evolution.

During the acute phase, the treatment involves dissolving clots that have formed. It can be done in different ways:

- **Pharmacotherapy:** Fibrinolytic (rt-PA) drugs are applied venously and sometimes arterially. This drug aims to dissolve the thrombus that has caused the stroke.
- **Surgical treatment:** Occasionally, surgical intervention will be necessary to remove the formed atheroma plaque or dilate the artery using stent angioplasty. A catheter is inserted, the tip of which ends in a small inflatable balloon that, when swollen, compresses the plate against the arterial walls.
- If the stroke is hemorrhagic, the appropriate treatment is embolization of the aneurysm with coils, substances that plug the damaged arteries and prevent it from breaking again.

It will also be necessary to treat and prevent the risk factors mentioned above to avoid the appearance of new episodes (arterial hypertension, heart disease, diabetes mellitus, etc).

3.5.6 Rehabilitation

Once the acute phase is over (although it is desirable to start during the acute phase) rehabilitation treatment comes into play, and it is designed according to the consequences of each patient. This process can last days, weeks or months, depending on the starting situation and the objectives to be achieved in a given patient.

Rehabilitation seeks to minimize deficits or disabilities experienced by the patient who has suffered a stroke, as well as facilitating their social reintegration by improving their autonomy and self-esteem.

These are some possible types of rehabilitation after suffering a stroke:

- **Rehabilitation with physical therapy:** Technology and recovery programs provide encouraging results for all extremities, but especially arms and hands. Extremities that are affected in 85% of the survivors of a stroke. In our TFG we focus on this type of rehabilitation.
- **Occupational therapy:** This type of stroke rehabilitation focuses on getting the patient back to being able to practice daily activities, exercising their motor skills and adapting to the environment again.
- **Speech therapy:** One of the usual effects of stroke is language difficulty that can manifest itself as the inability to emit or understand any coherent language (aphasia), the difficulty in articulating words (dysarthria) or the inability to speak. In these cases, the speech therapist is in charge of helping the patient regain language and communication skills.
- **Rehabilitation of dysphagia:** There is a possibility that the patient who has suffered a stroke may have difficulty swallowing. Dysphagia can be rehabilitated with a modification of the diet, as well as with safe feeding techniques so that the patient does not suffer from malnutrition, dehydration, or pass food to the lung.

3.6 Stroke rehabilitation with VR

Nowadays, the rehabilitation of the motor function of stroke patients using VR technology is an increasingly common fact. It is a sector where more and more applications are being developed.

In the last ten years, there have appeared the largest number of publications on the use of VR for the rehabilitation of people with strokes. A selection and research of scientific articles and papers where VR stroke rehabilitation is introduced to see the advances in this sector and to know the current status of this technology. Most of the articles found aim to improve the motor function of the upper limb, something that is already going perfect for this project because it also focuses on improving the functionality of this part of the body.

Numerous studies have been found which affirm that the use of VR is beneficial in the rehabilitation of stroke patients. Some of these articles are explained below:

In an article published in 2015 by S. Viñas-Diz and M. Sobrido-Prieto [17], the authors studied the first publications that appeared in the field of rehabilitation of stroke patients with VR. His study, "Virtual reality for therapeutic purposes in stroke: A systematic review", reviews some scientific articles from that time to do a study on the effectiveness of the use of VR to improve the motor function of patients who have suffered a stroke. The results of the different tests carried out were successful because they prove that the use of VR in the rehabilitation of patients who have suffered strokes brings great benefits. Nevertheless, they propose to study it in more detail; for example, to find out if the effects last in the long term, which VR system is better or to define which intensity of treatment is the most appropriate.

Another article published in 2017 in Brazil [18], entitled "Efeito de um programa de reabilitação usando realidade virtual na função do membro superior em pacientes com acidente vascular cerebral," also aims to study the effects of VR therapy on the function of the upper limb in the recovery of individuals after stroke and uses neuroimaging characteristics in the acute phase as a predictor of a better response to this therapy. The results were that functional independence showed an improvement after the intervention, functional characteristics and mobility of the upper limb showed an improvement after the intervention, as well as the quality of life of the patients.

These conclusions are also reached in more recent articles such as the article published in 2018 "Effects of virtual reality therapy for the upper limb in stroke patients: a systematic review" [21]. The objective of this study was to update and compare VR therapy with conventional therapy. Studies comparing the two therapies were consulted in different databases: Medline (Pubmed), PEDro, Cochrane Central, CINAHL and Web of Science, and a manual search was also done. The conclusion was, once again, that mobility, strength, quality, and quantity of movement of the upper limb improve more after the application of VR therapy compared to conventional therapy.

Finally, and more recently, we have found and studied the article published in April 2020 "Rehabilitation through virtual reality therapy after a stroke: A literature review" [22]. This publication analyzes the efficacy of VR as rehabilitation therapy for improving movement in adults after suffering a cardiovascular accident (CVA).

This is a more interesting study for the project because the authors made a date restriction of the last five years in their bibliographic review, therefore, its conclusion is also more up to date. The conclusion they have reached has been that VR-based therapy can be effective in improving movement in post-stroke patients, either in isolation or as a complement to conventional therapy.

Then, there are two studies on that area from the Universitat Autonoma de Barcelona (UAB). The first one, "Rehabilitation of the upper limb in chronic stroke patients with Virtual Reality: A Systematic Review" [19], makes a study similar to those discussed above comparing different articles. The conclusion reached is that VR has a positive impact on the recovery of stroke patients: it induces cortical changes with repercussions on motor function, motivates the patient, provides constant feedback, and can contribute to a discharge from rehabilitation centers.

In the other article from the UAB, called "Rehabilitación del miembro superior parético en pacientes con ictus: Eficacia del empleo de Entornos Virtuales, Soportes Robóticos y Retroalimentación Visual Con Espejo" [20], an interesting example of an application of VR called Umbrella was found. Umbrella consists of a depth sensor (Kinect), a projector, and a table where the scenarios in which the patient interacts are projected.

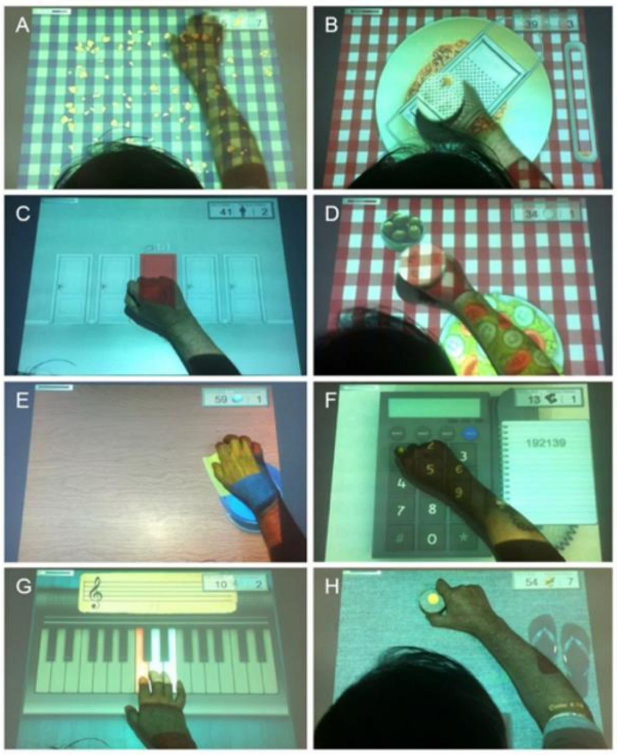


Figure 3.13. Images of different exercises of the Umbrella application. [https://ddd.uab.cat/]

There are more recent examples of applications that use VR to rehabilitate stroke patients. For example, a mobile application designed by Ezequiel Hidalgo, a rehabilitation doctor at the *Hospital Universitario Ramón y Cajal* and a researcher at the *Instituto Ramón y Cajal de Investigación Sanitaria (IRYCIS)* [23]. In this application, patients have access to videos with series of exercises that they can follow from their own home using virtual reality. The application simulates the same activities that patients would do in a traditional gym and, in all cases, there is a playful component to try to improve adherence to treatment.

Figure 3.14. VR application for srtoke rehabilitation designed by Ezequiel Hidalgo. [https://www.infosalus.com/]

Rehabilitation Gaming System (RGS) was also created, which is an initiative of the *Hospital Universitari Vall d'Hebron*, *Universitat Pompeu Fabra*, *Hospitals del Mar i de l'Esperança*, *Universitat Henrich Heine*, *Guger Technologies OEG*, *TMotion* and the *TicSalut Foundation* [24]. The *Rehabilitation Gaming System* is a virtual reality tool aimed at motor rehabilitation of the upper extremities in patients who have suffered a stroke brain injury. It works thanks to a personalized training program that combines the execution of movement with the observation of the same action performed by virtual arms.

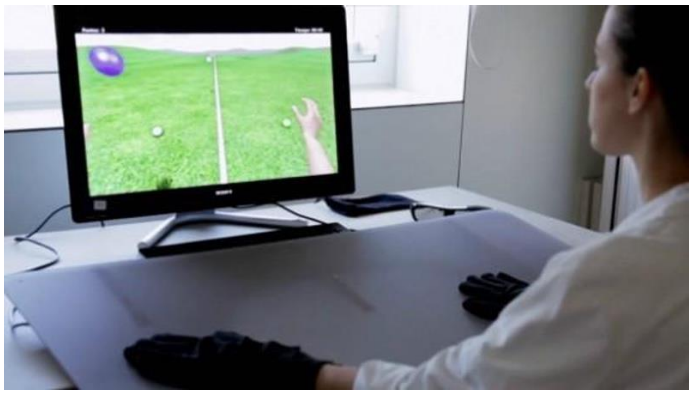


Figure 3.15. The Rehabilitation Gaming System. [https://www.vallhebron.com/ca]

To summarize, using virtual reality in the rehabilitation of patients who suffered a stroke is becoming more and more popular because there is supporting evidence of its benefits. This is the reason why there are many new projects and startups creating new applications in this field in today's world.

The rehabilitation application created in this project incorporates some features that can be highlighted: Activities focused on movement control and adaptability.

On one hand, the accuracy of the actions, the speed and the time available for doing the exercise can be easily configured providing a useful tool to stimulate active repetitive movements. In addition, our project reports the progress done between successive sessions. The therapist can use this information to assess and influence the progress of movements. This is a feature sought in some VR rehabilitation systems. [19]
On the other hand, this application has great adaptability in the presentation of movements as a reproduction of the movement produced in the real world by the patient. This feature allows the customization of the experience and can help the improvement of the therapeutic function of the system by providing graduated rehabilitation activities that can be individualized. [20]

Last but not least, the benefits of VR applied to rehabilitation can be summarized as follows:

- Non-invasive.
- Low-cost.
- Self-efficacy.
- Increase in participation and motivation of the patients.
- Telerehabilitation: treatments at home providing information to patients at the time. [21]
- Collecting data by therapists.
- Ecological validity.
- Hope for change.

4. PROJECT PLANNING

4.1 Workplan

These two applications are full new, so the workplan is represented from the beginning to the end of two VR application creation. So, it is necessary to implement some tasks to make this project become a good project. This modus operandi is quite common nowadays in every single company when they work in any project, so this project follows the same strategy that they usually do.

It is important to take care about the planning before starting the project, the control during the process and the presentation at the end. To make that possible there are some requirements that must be completed:

- **Optimize times:** do not lose too much time in a single section that you have stuck, just skip it and it will be fixed in the future.
- **Efficient work:** being well organized between both work member is quite necessary to create more content and do not make duplicates.
- **Meetings:** this part is the most important thing in a control process. If there is no communication between both sides nothing will go right.
- **Presentation:** the final result needs to be well presented to the audience. There is when all matters and when everybody will judge this work. So, it is essential to show this work as well as it can be, getting a good explanation in writing and orally.

Figure 4.1 shows the general sequence of how the whole process has been carried out.

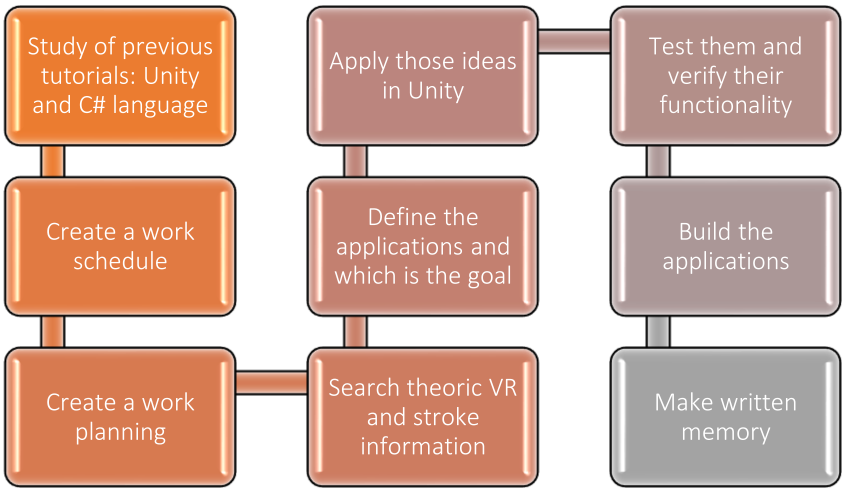

Study of previous tutorials: Unity and C# language
Create a work schedule
Create a work planning
Search theoric VR and stroke information
Define the applications and which is the goal
Apply those ideas in Unity
Test them and verify their functionality
Build the applications
Make written memory

Figure 4.1. Workflow during the final work degree.

4.2 Gantt Chart

We have planned all tasks of our project with the Gantt chart tool. This graphic tool has provided a general view of the programmed assignments. It shows all the activities realized, the start date, the end date and the duration of each activity measured in days.

Activity	Start date	Duration (in days)	End date
Project assignment	29-ene	1	30-ene
Unity online courses	30-ene	17	16-feb
First meeting with our tutors	7-feb	1	8-feb
Planning the first tasks	17-feb	1	18-feb
Reading previous TFGs	19-feb	5	24-feb
First contact with Unity program	30-ene	26	25-feb
Game development	25-feb	118	22-jun
Find assets for each game	25-feb	10	6-mar
Creation of the *horse game* environment	6-mar	26	1-abr
Creation of the *washing face game* environment	6-mar	26	1-abr
Second meeting with our tutors	8-abr	1	9-abr
Project reorentation due to COVID-19	9-abr	4	13-abr
Planning the new tasks of the project	13-abr	7	20-abr
Download VR simulation tools (VRTK, Steam VR)	20-abr	7	27-abr
Brainstorming of the game's new mechanical additions (mouse trail)	20-abr	14	4-may
Development of the whole activity scene of the *horse game*	27-abr	35	1-jun
Development of the whole activity scene of the *washing face game*	27-abr	35	1-jun
Export the nº of repetitions, time, accuracy and level in a database	18-may	28	15-jun
Third meeting with our tutors	21-may	1	22-may
Corrections and improvements	15-jun	7	22-jun
Project report	20-abr	63	22-jun
Write the project report	20-abr	49	8-jun
Sampling with volunteers to extract data	1-jun	7	8-jun
Homogenization of work	1-jun	14	15-jun
Individual review of possible errors	15-jun	1	16-jun
Common review of possible errors	16-jun	1	17-jun
Modification of possible errors	17-jun	5	22-jun
Hand in the project report	22-jun	1	23-jun
Preparation of the presentation	23-jun	8	1-jul
TFG presentation	1-jul	1	2-jul

Figure 4.2. Activities schedule.

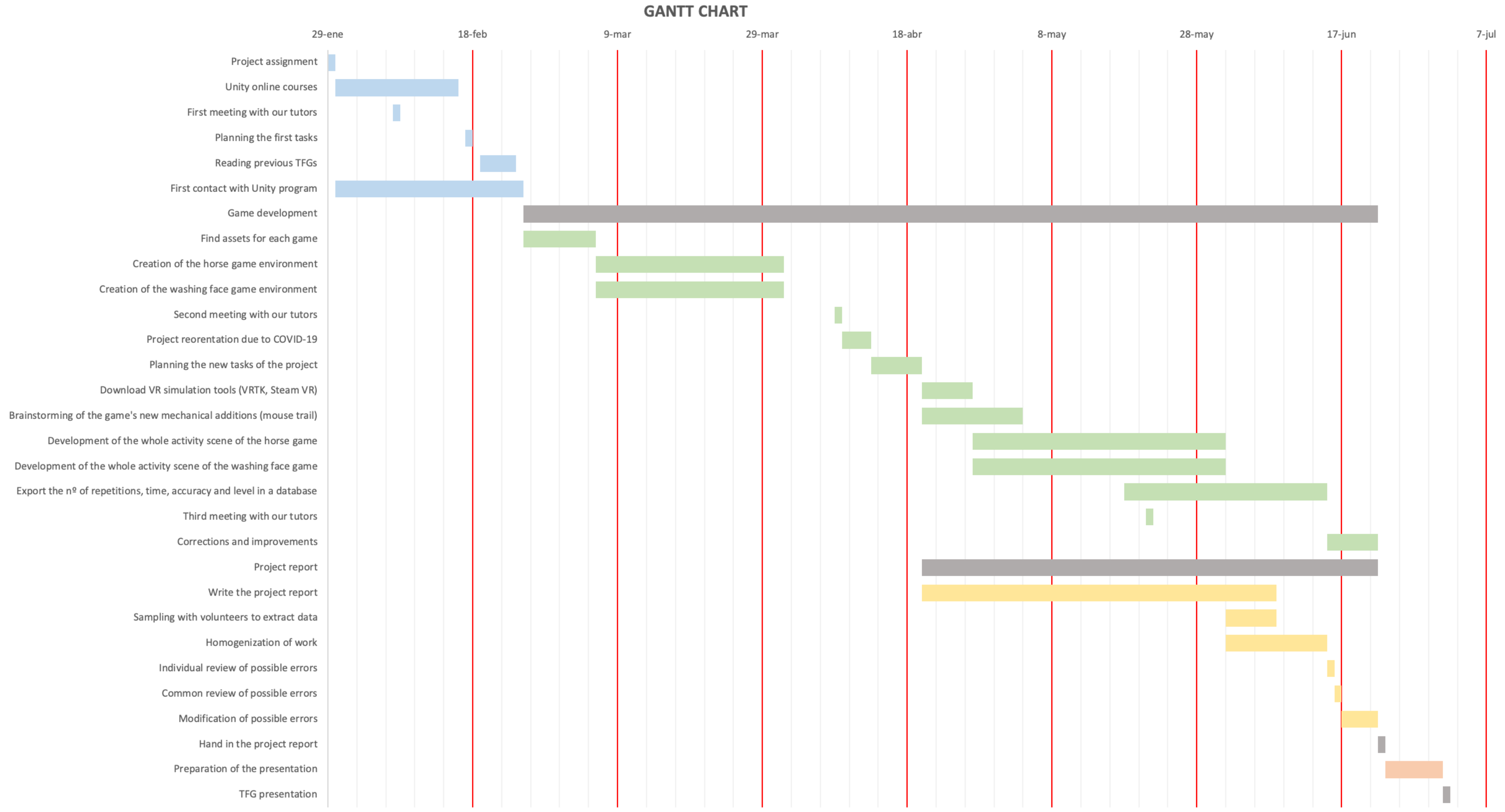


Figure 4.3. Project Workplan: Gantt Chart.

4.3 Functional and non-functional requirements

A requirement describes the services to be provided by the system and its restrictions. It is a characteristic that the system must have or a restriction that the system must satisfy to be accepted by the client.

The functional and non-functional requirements related with project must accomplish some factors. The needs of customers and other stakeholders should be reviewed, the current system, and the current organization too. Also, know the current system version and review existing documents and personal background.

4.3.1 Functional requirements

A functional requirement is what we expect from a system to do. The interaction between the system and the environment is described in functional requirements, as well as its state and functioning. So, the services or functions that the system must provide are detailed, as well as the inputs and outputs of the processes, and the data to be stored in the system. These inputs may be from users or may be the result of interactions with other systems.

In some cases, it is also desirable that the functional requirements detail what the system should not do.

The specific functional requirements for this project are:

- Being able to choose the exercise you want to realize.
- Being able to choose the desired level depending on the degree of rehabilitation that the patient needs.
- Have access to an instruction panel to know the rehabilitation goal in each game.
- Have access to an instruction panel on how to move and interact with the objects of the game.
- Possibility to go back at any time to end the exercise.
- Timing how much it takes the patient to perform the exercises.
- Calculate the number of repetitions that the patient does each exercise.
- Calculate the accuracy with which the exercises are performed.
- Export the values we are interested to a .csv file.

4.3.2 Non-functional requirements

A non-functional requirement defines how should the system be. Non-functional requirements are restrictions of the services or functions offered by the system. They are all the requirements of quality that are imposed on the project. The non-functional requirements refer to restrictions of some system properties such as reliability, response time, security, storage capacity, performance, availability, budget, or delivery time. There are different types of non-requirements and they are classified according to their implications:

- **Product requirements:** Requirements on the speed of the system execution and the amount of memory required, the reliability requirements that establish the failure rate for the system to be acceptable, the portability requirements and the usability requirements.
- **Organizational requirements:** Standards in the processes, implementation requirements such as programming languages or the design method, and delivery requirements that specify when the product will be delivered and its documentation.
- **External needs:** Include the requirements that define how the system interacts with the other systems in the organization, the legal requirements that must be followed to guarantee that the system works within the law, and ethical requirements.

The specific non-functional requirements for this project are:

- The software needed: Unity and Visual Studio.
- The system must have well-formed graphic interfaces.
- Quiet and relaxing game environment, without sudden movements that may affect the patient experience.
- Accessibility criteria: do not base the functionality of the game on the colours because sometimes the stroke patients cannot distinguish them.
- Application compatible with Windows 7 or higher versions.
- Possibility of doing the exercises without net connexion.
- Good application performance, with an acceptable response time.
- Hardware needed: mouse and mobile phones.
- Use of VR plugins such as VRTK or Steam VR.
- Sounds in the game to make the scene more realistic.

5. TOOLS USED

5.1 Unity: Videogame motor

Nowadays, there are two well-known videogame motors in the market giving good graphic benefits to game developing. These motors are Unity and Unreal Engine; however, sometimes there are some companies using their own motors.

Unity's motor has been chosen in this project to bring these applications to life, using the free version. Unity provides lots of graphic tools and uncountable assets to design and create whatever you want.

5.1.1 Unity's interface

When a new project is created, the interface appearance is literally empty (figure 5.1). The application just generates a new scene without any asset and any script. As you can observe in figure 5.1, there are different fields and tabs and each one has its own function. But, those five fields are used the most.

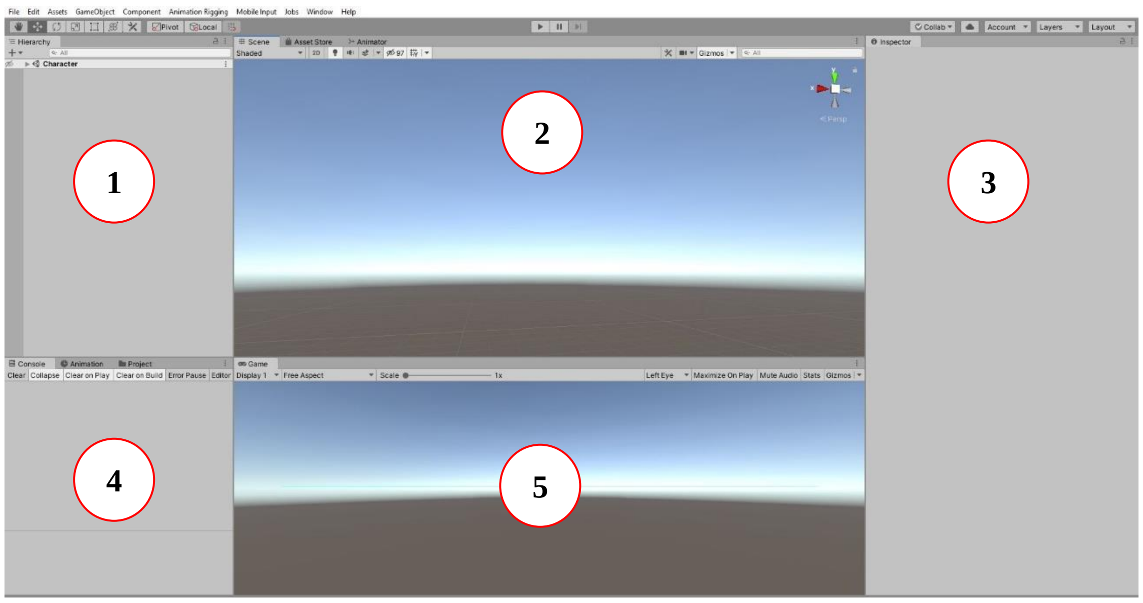

File Edit Assets GameObject Component Animation Rigging Mobile Input Jobs Window Help
Pivot
Local
Collab
Account
Layers
Layout
Hierarchy
Character
Scene
Asset Store
Animator
Shaded
2D
Gizmos
Inspector
Console
Animation
Project
Clear Collapse Clear on Play Clear on Build Error Pause Editor
Game
Display 1
Free Aspect
Scale
1x
Left Eye
Maximize On Play Mute Audio Stats Gizmos
1
2
3
4
5

Figure 5.1. Unity's interface: new project.

These fields have their particularities, but they are very intuitive to use because the environment is graphically clear. Here is the explanation of each one:

- **1- Hierarchy:** this field is constituted by all the assets, prefabs, effects, UI, particle systems and any element that the user can drag in the main scene. It is recommended to have everything organized in groups to work more efficiently without getting lost among all those game objects. There is also a useful feature which consists of hiding the game object in case there are too many things on the scene editor, and it makes it difficult to work properly.
- **2- Scene tab:** here is where the creativity is reflected, the editor field. It has different shading mode, angle perspective and other features to work with.
- **3- Inspector:** that is the field which relates the object with the script. There is where it gives life to the objects adding certain components like colliders, rigid bodies, materials, animations, etc.
- **4- Console and Project tabs:** they are together in the same corner, but there is the possibility to customize it as the user wishes. The console is the debugger, the visualizer or basically the instrument that tells the user what is happening in every single game frame. This is essential when there is a programming process involved because it indicates where the errors are. The other tab is the project folder where all the scripts, assets, animations, and so on, are located.
- **5- Game tab:** when the game starts running this is the scene where the users interact. It is helpful to customize the unity interface like this, to see both scenes (Editor and Game), and to check how the game looks like when the game is running. What the users see is what the camera, placed in the editor, is looking.

5.11 Compatible code languages

Unity games can be scripted in several languages like C#, JavaScript or Boo, but the most common language used is C#.

C# was created by Microsoft to orientate the object programming in a more standard language not as complex as C++. Visual Studio is considered one of the main IDE to code Unity scripts in C#. When game developers design videogames, they usually contain a lot of classes, so Visual Studio has good code navigation to find these class references.

5.2 Hardware

In the VR laboratory at the university, there is the SteamVR equipment to test the applications. However, this VR equipment has not been used on this project due to the confinement. So, this section contains all the workarounds and alternatives made to take this project forward.

5.2.1 Computer

Of course, the main electronic device to carry out this project has been a computer. Particularly, for this kind of projects it requires to be a graphically a powerful computer. Depending on some VR applications, they could consume a lot from the graphic's card. It also needs a good processor (CPU) to take the most graphic card efficiency.

5.2.2 Cardboard and mobile devices

Nowadays, technologies advance and everybody can perform VR activities without buying expensive electronic devices. This project has been affected by the pandemic virus COVID-19, so there is the need to find other alternatives to probe VR applications. A cheaper way to make that possible is using mobile phones and their respective applications.

There are some companies who bet for this new point of view and create their own cardboards to introduce a mobile phone inside them. Thanks to them, everybody can get immersed into a virtual world. Google Carboard (figure 5.3) is the most famous mobile adapter in market right now.

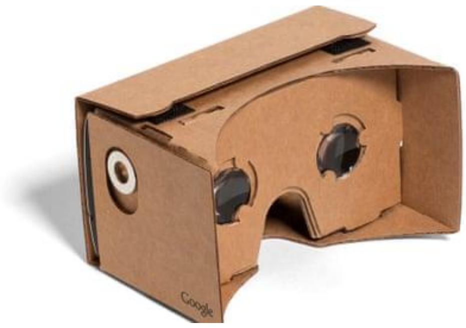


Figure 5.2. Google Cardboard
[https://www.whatvr.co/].

This Google product is made by cardboard, as its name indicates, and two stereoscopic glasses, to create the three-dimensional illusion.

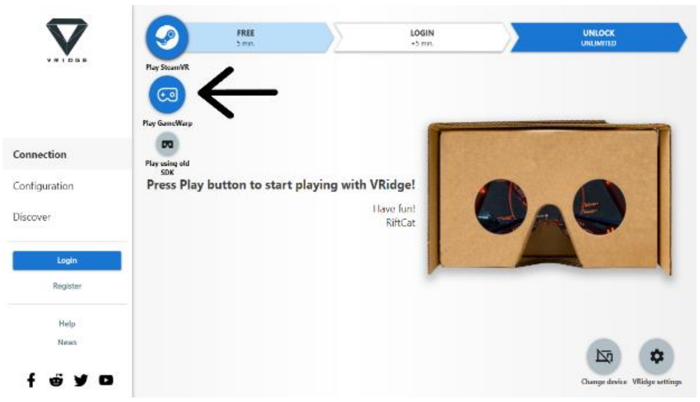
5.3 Software

Not all the merit is for the hardware devices. To accomplish all the objectives in this project, it is necessary to resort to the applications that will be potentially useful for the VR rendering.

5.3.1 Vridge and Riftcat

To reproduce a real VR experience with a mobile phone setup, these applications for Android, iOS and Windows are highly effective. There is the possibility to get a VR equipment using just three mobile phones, one as a headset and the rest for the right and left controllers. However, it is necessary to buy a cardboard to get immersed into the game. This application uses "GameWarp" technology to reproduce a high-quality game resolution.

To setup this it is necessary to synchronize the mobile phone with the computer. So, first the Vridge mobile app must be installed in both controllers and then install Riftcat to the computer. After that, it is recommended to register a Riftcat account to provide 5 more minutes of testing.



Press Play button to start playing with VRidge!

Have fun!
RiftCat

Figure 5.3. Rifcat application synchronized with Vridge headset and controllers. [https://support.riftcat.com/]

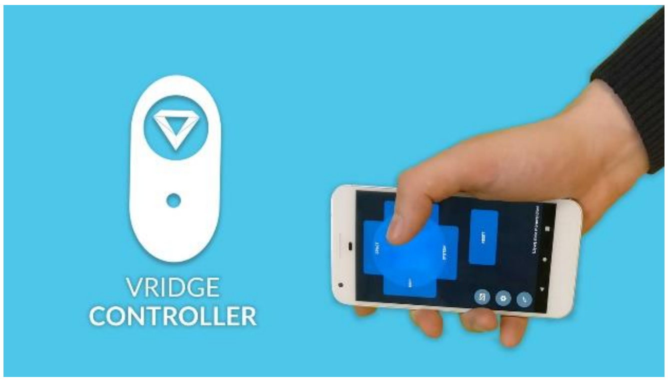


Figure 5.4. Vridge controller application. [https://skarredghost.com/]

5.3.2 SteamVR plugin

Steam has the VR equipment and, obviously, it has the plugin to work with. It can be found in the Unity Asset Store or in its official website. This package contains prefabs, scripts, examples and so on. Just with its content it is possible to create a VR game perfectly.

This option has been taken to create the “Cleaning Mirror” game, using its camera rig prefab to play in VR mode. Together with Vridge compatibility, all the tests could be positives and satisfactory.

The way to use it consists of installing SteamVR application from the Steam platform and calibrate the position of the user. There are two possibilities available to configure the room where the games will be tested:

- **Room scaled:** that gives the possibility to move throughout the room standing or sitting.
- **Stand still at one point:** interact with the game without walking or moving around the room, just using the controllers and the headset in a single place.

In the “Cleaning Mirror” game case, the second option has been chosen. So, the last step to configure the room is to set the user’s height to calibrate the headset.

When the game starts running, it appears at the left bottom corner a SteamVR tab telling which devices are connected at that moment. In this case, it shows two controllers and the headset icons in a green state (figure 5.5). If one device icon is in a gray state means that the device is not connected yet (figure 5.6).

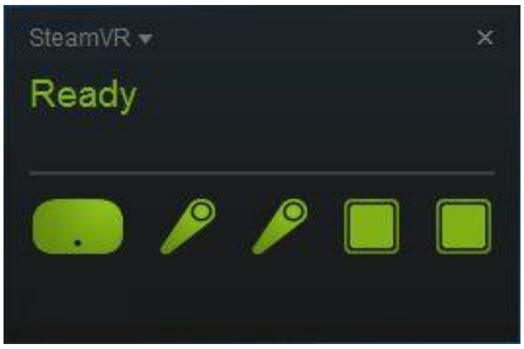


Figure 5.5. SteamVR Connected Devices: Green colour [https://www.reddit.com/].

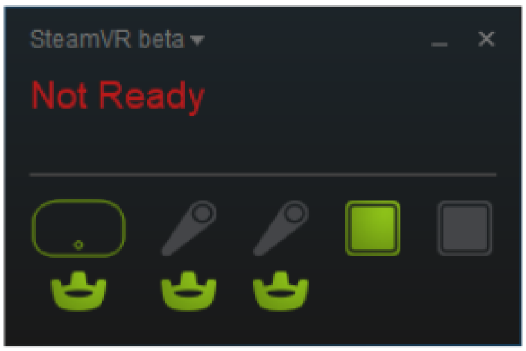


Figure 5.6. SteamVR Disconnected Devices: Gray colour [https://www.reddit.com/].

5.3.3 VRTK toolkit

If the purpose is to create a virtual game without the VR equipment, this toolkit is the solution. This package of scripts and scene examples is available with no costs in the Unity Asset Store. It offers the opportunity to simulate a VR situation using PC accessories like mouse and keyboard. It also have some prefabs like the camera rig or controller models to introduce into the projects.

All its attributes can be used with the SteamVR or the Oculus Rift set ups. That is a good advantage at the moment of using this toolkit. This plugin has been used to test the "Clean the Horse" game, and it gave all the facilities to carry out this project.

Figure 5.7. VRTK toolkit logo.
[https://vrtracker.xyz/]

6. APPLICATIONS DESIGN

This section explains how the games have been designed and programmed. It explains how the environment of the scenes was created and every script that the games contain.

6.1 Horse cleaning game

6.1.1 VR application idea

This game consists on cleaning a horse with a hose by following the trajectory that appears in the body of the horse. Initially, we thought about the idea of petting the horse doing circles in its body because the game goal was that the patient could be able to do circular movement with their arms. Later, with the arrival of COVID-19 and the confinement situation, we changed the goal of the game because we did not have the material to detect the patient's movements. Then, with the ADFO collaboration, we decided that the patient could follow different trajectories with the mouse. So, we used the environment we had already designed, and decided that the main idea would be to clean the horse with a hose following different trajectories using the mouse of the computer.

ADFO proposed this idea because there exist some rehabilitation therapies that consist on moving the upper limb in a horizontal plane at table height. So, with different exercises that include the movement of the mouse, the patients will be able to practice this kind of exercises in a more entertaining and useful way.

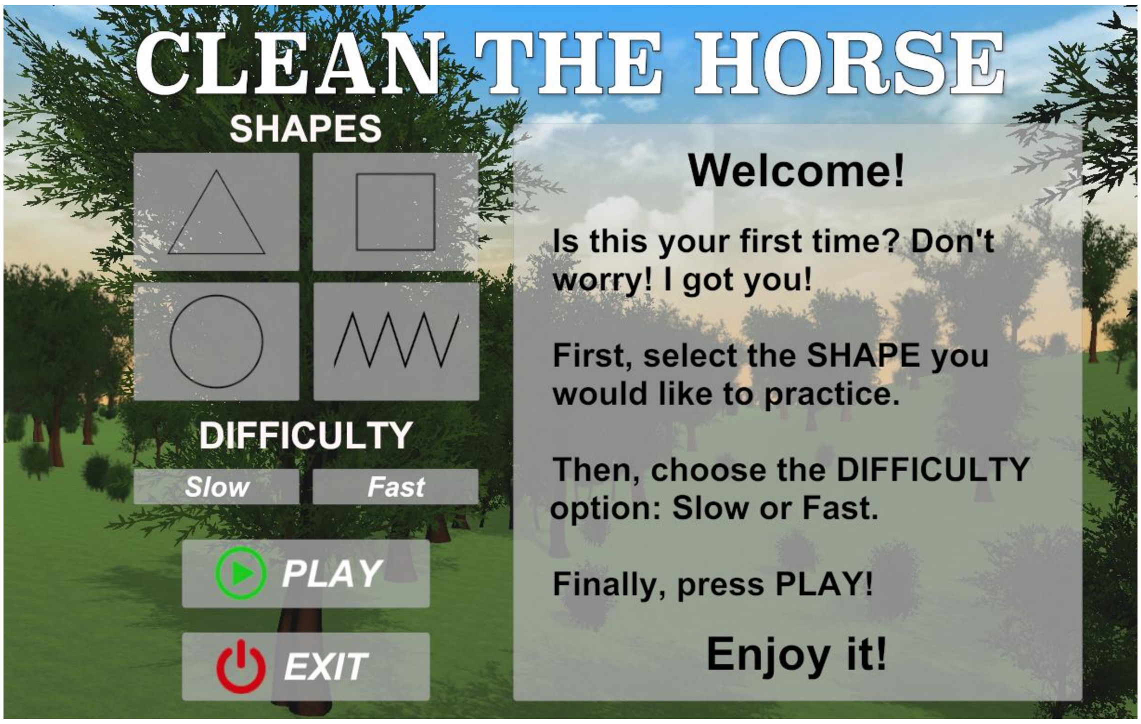
6.1.2 Graphic environment design

This game consists on cleaning a horse with a hose by following the trajectory that appears in the body of the horse with the mouse of the computer. In this game, two scenes can be found. One scene is for the menu and the other one is the game.

On the one hand, to design the environment of the first scene (the Menu Scene) there are different game objects attached to the scene. There are two directional lights to make a pleasant lighting in the scene. An asset from the Asset Store of Unity named Nature Starter Kit 2 has also been downloaded.

This asset has a game object called Terrain that is a forest plenty of trees. The Terrain is placed in the Menu Scene in order to be the background of the menu.

Furthermore, there is the canvas of the Menu Scene (figure 6.1). The canvas is shown as a rectangle in the Scene View and it is the area where all UI elements are located inside. This canvas has an image for the main title "Clean the horse" and two text objects for "Shapes" and "Difficulty" titles. At the left of the canvas, there are eight panels with a button component that are the four buttons for selecting the shape, the two buttons for selecting the difficulty, the PLAY button, and the EXIT button. These panels can have an image inside, a text, or both. On the other side, at the right of the canvas there is an instruction panel.



Welcome!

Is this your first time? Don't worry! I got you!

First, select the SHAPE you would like to practice.

Then, choose the DIFFICULTY option: Slow or Fast.

Finally, press PLAY!

Enjoy it!

Figure 6.1: Menu Scene of the game "Clean the horse".

On the other hand, the Game Scene has been designed. This scene has two-point lights and one directional light to make a pleasant lightning in the scene (yellow square in figure 6.2). The assets downloaded in the pack of Nature Starter Kit 2 are also used in this scene. This pack contains different scripts that can be attached to the main camera (green square in figure 6.2) to animate the scene. Some of them have been used, such as antialiasing which uses a technique to smooth otherwise jagged lines or textures by blending the color of an edge with the color of pixels around it. The result is a more pleasing and realistic appearance. The main camera has other scripts from the Nature Starter Kit 2 for instance for creating sun shafts or vignette and chromatic aberration.

Three game objects have been taken from the Nature Starter Kit 2 that were already created (red square in figure 6.2). They consist of two different terrains and a pack of different trees and bushes. A game object called "WindZone" has also been incorporated to move the leaves of the trees and make it more realistic (orange square in figure 6.2).

Another asset called "Horse Animset Pro Riding System v4.06" has been downloaded too. From this package, a horse asset (pink square in figure 6.2) has been incorporated in the Game Scene. This horse already had different animations and audios incorporated to simulate the noise of the horse and movements to make a more realistic environment.

Moreover, this scene has different game objects (purple square in inside the blue square in figure 6.2) with an image component to indicate the shape chosen to follow the trajectory, and other images that indicate the number of repetitions the patient has done.

Inside the blue square in figure 6.2, there are different game objects that are the key of the game. It includes an asset to represent the hand of the patient and a hose. Moreover, it has the scripts to simulate the headset view and the controllers with the mouse of the computer. It also includes a particle system component that collides with different colliders placed in the shape selected. In this way it can measure if the patient follows the trajectory correctly. All these game objects are explained with their corresponding scripts more deeply in section 6.1.3.

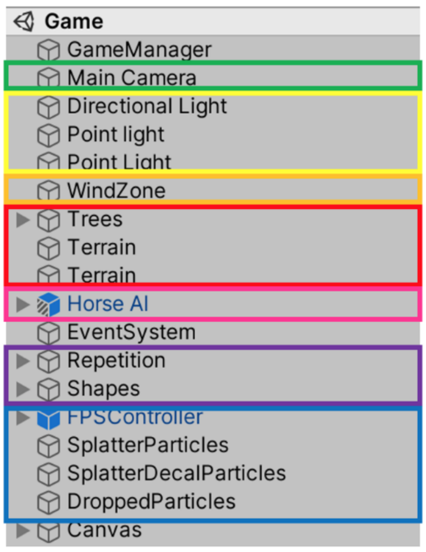


Figure 6.2. Hierarchy of the Game Scene from "Clean the horse" game.

Lastly, there is the canvas of the Game Scene. Inside the canvas there are different panels.

First, there is the Top Panel that consists of an instruction panel (blue square in figures 6.3 and 6.4). Then, we find the Chronometer (pink square in in figures 6.3 and 6.4) that is activated when the patient starts following the trajectory. There is also a red time bar (yellow square in figures 6.3 and 6.4) which is also activated when the patient starts following the trajectory and it is a countdown timer. As time goes on, the red bar is being gradually consumed.

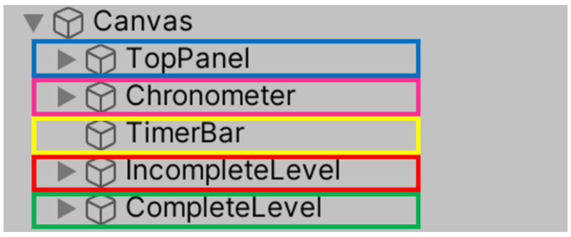


Figure 6.3. Hierarchy of the Game Scene from “Clean the horse” game. Specifically, the game objects of the canvas are shown.

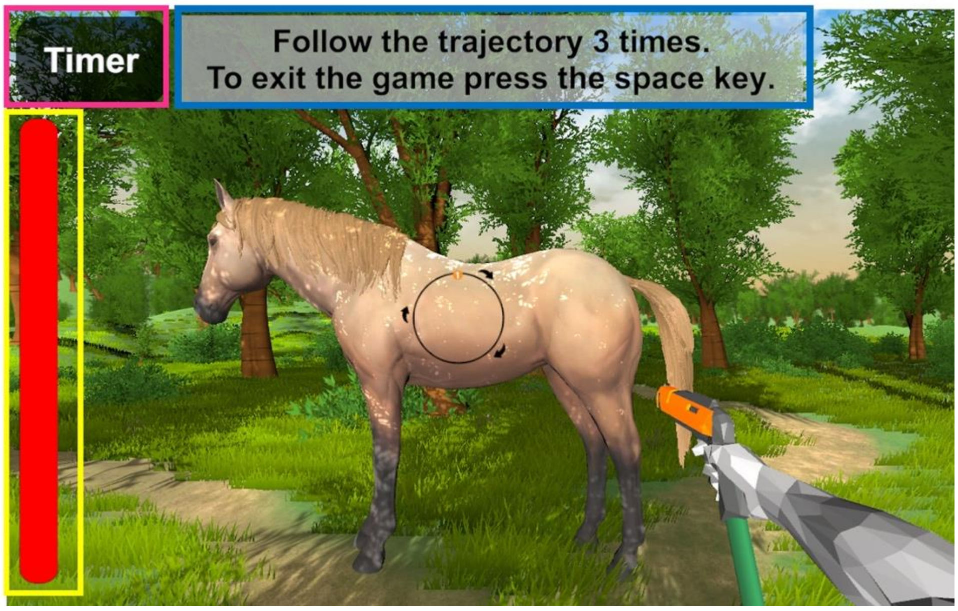

Timer
Follow the trajectory 3 times.
To exit the game press the space key.

Figure 6.4. Game Scene with the circle shape selected.

If the red bar is consumed completely before the patient can manage to do three repetitions of the same trajectory it appears the panel called “IncompleteLevel” from the canvas because the patient has run out of time.

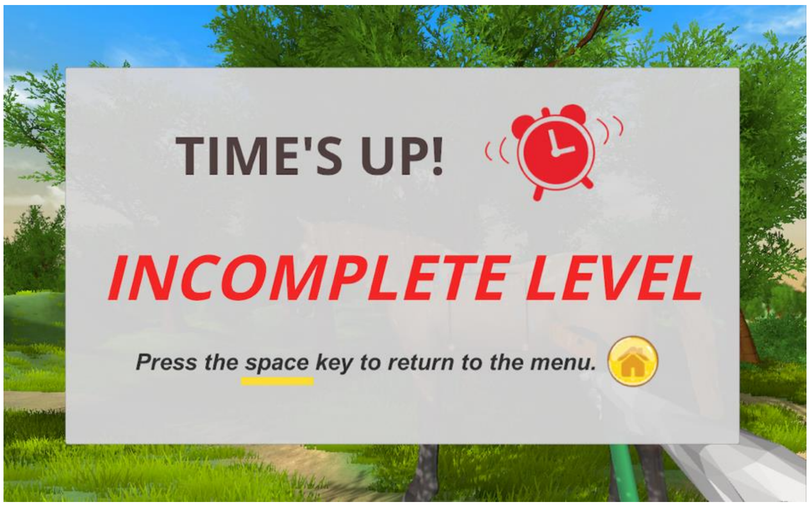


Figure 6.5. "IncompleteLevel" panel from the Game Scene.

Otherwise, if the patient can manage to do the three repetitions of the same trajectory, the panel called "CompleteLevel" appears from the canvas. This panel includes the accuracy result and the time taken by the user to do the exercise. In the example below (figure 6.6), that patient has made the exercise in 45 seconds and with an 82% of accuracy.

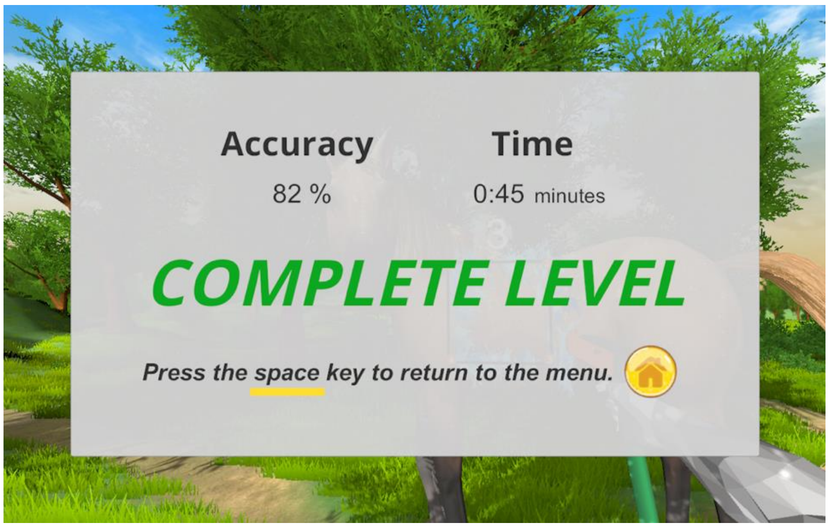


Figure 6.6. "CompleteLevel" panel from the Game Scene.

6.1.3 Game Objects programming

As commented, there are two scenes in this game: the Menu Scene and the Game Scene. To a better understanding of the programming of this game, note that the patient has three options to end the exercise: to perform the exercise correctly before the time runs out, to finish the exercise because the time has run out, or to get out of the game in the middle of the exercise. The block diagram in figure 6.7 shows the different options inside this game.

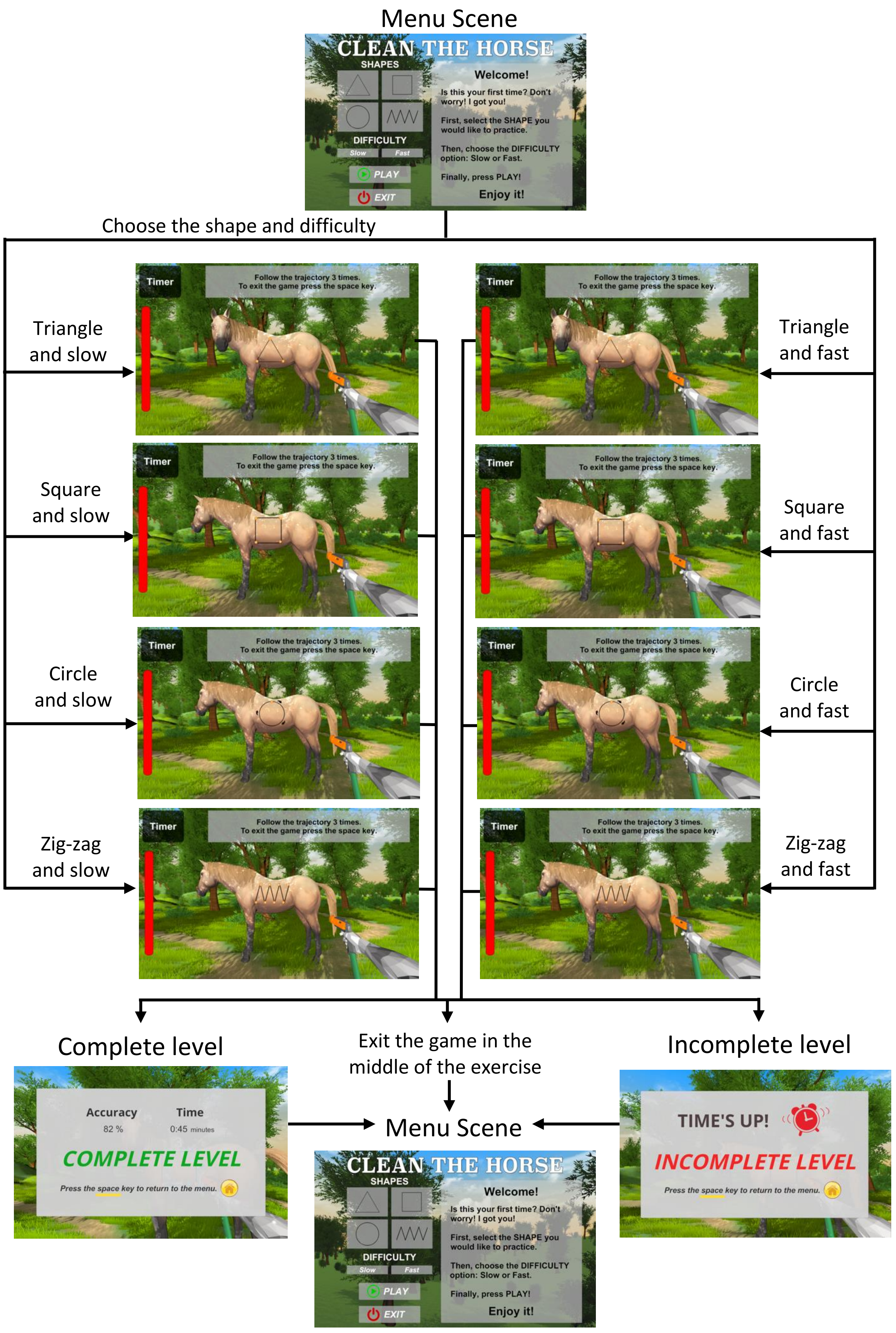

Menu Scene
CLEAN THE HORSE
SHAPES
DIFFICULTY
Slow
Fast
PLAY
EXIT
Welcome!
Is this your first time? Don't worry! I got you!
First, select the SHAPE you would like to practice.
Then, choose the DIFFICULTY option: Slow or Fast.
Finally, press PLAY!
Enjoy it!
Choose the shape and difficulty
Timer
Follow the trajectory 3 times. To exit the game press the space key.
Triangle and slow
Square and slow
Circle and slow
Zig-zag and slow
Triangle and fast
Square and fast
Circle and fast
Zig-zag and fast
Complete level
Accuracy
Time
82 %
0:45 minutes
COMPLETE LEVEL
Press the space key to return to the menu.
Exit the game in the middle of the exercise
Menu Scene
Incomplete level
TIME'S UP!
INCOMPLETE LEVEL
Press the space key to return to the menu.

Figure 6.7. Block diagram of the game "Clean the horse".

In the Menu Scene there is only one script. This script has different functions with the goal of changing the scene. First, when the scene starts (green square in figure 6.10) the PLAY button is not enabled. This is because we want the user to select this button after choosing the shape and difficulty of the game. It also makes the cursor of the mouse visible to be able to correctly select the menu options.

Moreover, this script has different functions (orange square in figure 6.10) that are going to be activated depending on the shape and difficulty that the patient chooses in the menu. The different panels in the canvas of shape and difficulty have a button component where each one activates a different function. For example, if the patient clicks the triangle shape, the Triangle function is going to be enabled (figure 6.8). First, the shape (triangle, square, circle or zigzag) has to be selected and then the difficulty (slow or fast). Once the difficulty is selected, the PLAY button is enabled.

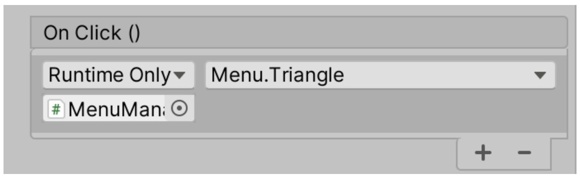

On Click ()
Runtime Only | Menu.Triangle
MenuMan:

Figure 6.8. Button component attached in the panel where the triangle shape is placed.

Then, there is the “ChangeScene” function (purple square in figure 6.10). This function is activated when the PLAY button is pressed or when the EXIT button is pressed. As you can see in the figure below, the PLAY button allows to change from the Menu Scene to the Game Scene. In the case of the EXIT button, it allows to change from the Menu Scene to the main menu where all the games are placed (figure 6.9). The “ChangeScene” function also sets the “timescale” to 1 because if in other scenes the “timescale” is set to 0, the game does not work correctly.

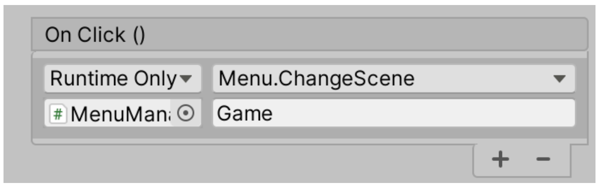
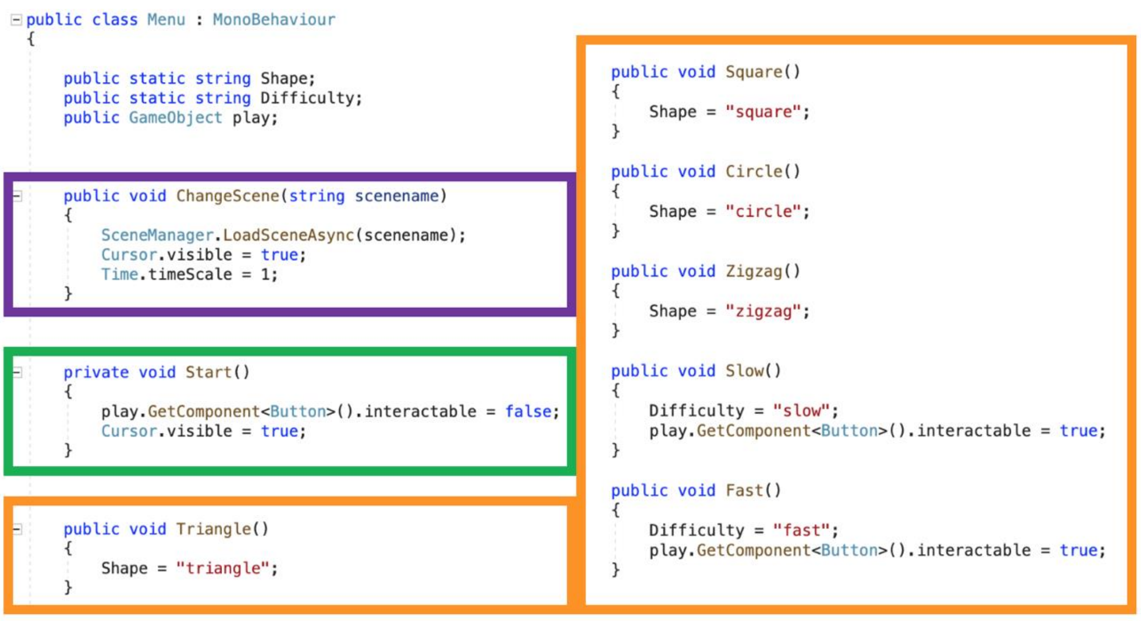

On Click ()
Runtime Only | Menu.ChangeScene
MenuMan: | Game

Figure 6.9. Button component attached in the PLAY panel.



```
public class Menu : MonoBehaviour
{

    public static string Shape;
    public static string Difficulty;
    public GameObject play;

    public void ChangeScene(string scenename)
    {
        SceneManager.LoadSceneAsync(scenename);
        Cursor.visible = true;
        Time.timeScale = 1;
    }

    private void Start()
    {
        play.GetComponent<Button>().interactable = false;
        Cursor.visible = true;
    }

    public void Triangle()
    {
        Shape = "triangle";
    }
```

```
public void Square()
{
    Shape = "square";
}

public void Circle()
{
    Shape = "circle";
}

public void Zigzag()
{
    Shape = "zigzag";
}

public void Slow()
{
    Difficulty = "slow";
    play.GetComponent<Button>().interactable = true;
}

public void Fast()
{
    Difficulty = "fast";
    play.GetComponent<Button>().interactable = true;
}
```

Figure 6.10. Menu script in the “Clean the horse” game.

In the other scene (the Game Scene), there is more than one script. First, there is the “GameScript” (figure 6.12). The Start function of this script makes the cursor not visible because it is easier to play the game. The script also creates a variable called “LevelResult” that in the Start function, the first value that it gets is “Level not finished”. In further scripts this variable will change depending if the level is complete or incomplete. If the patient wants to skip the game in the middle of the exercise, this variable will remain “Level not finished”.

Then, the Start function also enables the image of the shape selected in the menu (defined in the trajectory variable), the colliders of the shape desired (defined in the *objecte* variable), the *MiddleCollider* and the *EndCollider* (defined in the other variable), and the *maxTime* variable.

- **Colliders:** the colliders are game objects placed all over the shape chosen to follow. When the trace, that the patient does with the mouse, collides with a collider, the collider is destroyed. In this way, the game is able to calculate the accuracy of the patient’s trajectory.
- **MiddleCollider:** It is a collider placed in the middle of the path of the shape chosen to follow.
- **EndCollider:** It is a collider placed at the end of the path of the shape chosen to follow.
- **maxTime:** time allowed to do the exercise. (Figure 6.11).

These last three colliders change in different shapes and difficulties chosen.

maxTime	**Triangle**	**Square**	**Circle**	**Zigzag**
Slow	2 min	2 min	2 min	3 min
Fast	1 min 30 s	1 min 30 s	1 min 30 s	2 min

Figure 6.11. Time allowed to do the exercise in each case.

You can see the full “GameScript” code in Annex C. This script also has an Update function that allows the patient to exit the game at any time during the exercise by pressing the space key. Doing so, the Menu Scene will be opened again.

```
public class GameScript : MonoBehaviour
{
    private GameObject[] trajectory; //to activate the shape
    private GameObject[] objecte; //to activate colliders
    private GameObject[] other; //to activate MiddleCollider and EndCollider
    public static float maxTime;
    public static string LevelResult;

    // Start is called before the first frame update
    void Start()
    {

        Cursor.visible = false;
        LevelResult = "Level not finished";

        if (Menu.Shape == "triangle" && Menu.Difficulty == "slow")...

        if (Menu.Shape == "triangle" && Menu.Difficulty == "fast")...

        if (Menu.Shape == "square" && Menu.Difficulty == "slow")...

        if (Menu.Shape == "square" && Menu.Difficulty == "fast")...

        if (Menu.Shape == "circle" && Menu.Difficulty == "slow")...

        if (Menu.Shape == "circle" && Menu.Difficulty == "fast")...

        if (Menu.Shape == "zigzag" && Menu.Difficulty == "slow")...

        if (Menu.Shape == "zigzag" && Menu.Difficulty == "fast")...
    }


    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            SceneManager.LoadScene("Menu");
        }
    }
}
```

Figure 6.12: Part of the Game script.

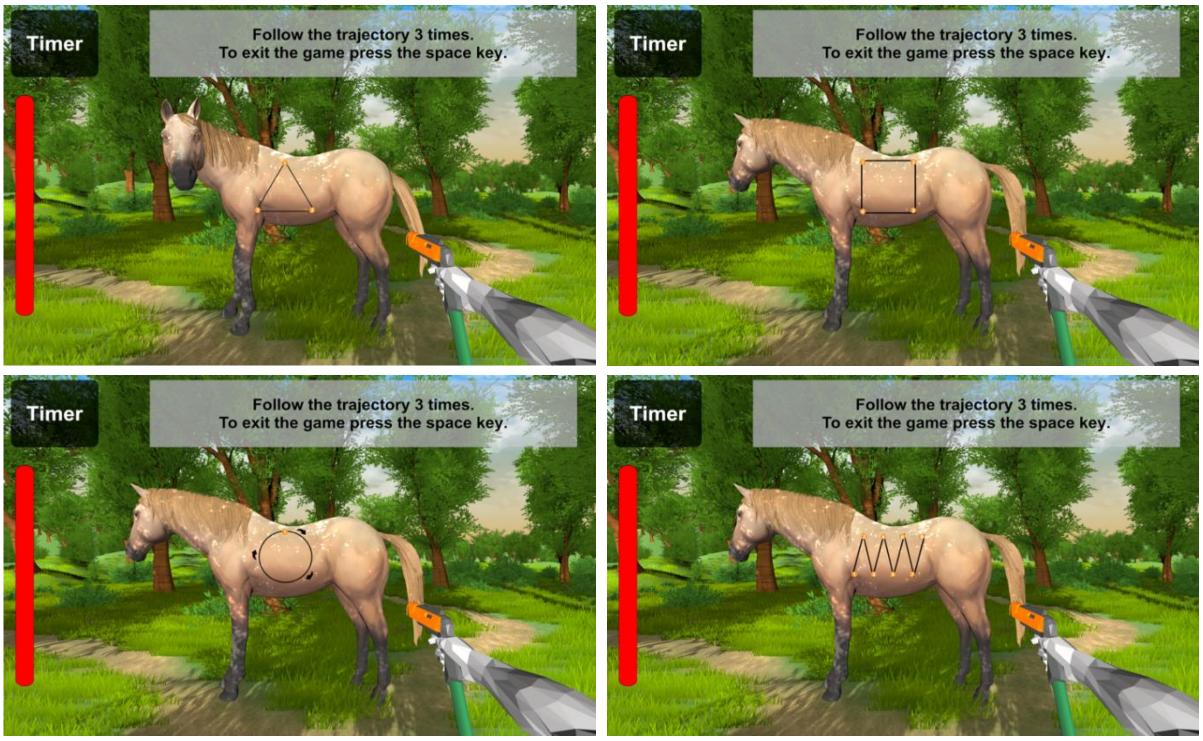

Timer
Follow the trajectory 3 times.
To exit the game press the space key.

Figure 6.13. Clean the Horse Game: the four possible Game Scene options.

Then, an online video tutorial called "Controlling Particles Via Script" was followed. Thanks to this tutorial, an asset called "AssetParticleScripting" was downloaded, in which there is a prefab called "FPSController" that has the children shown in the figure 6.14.

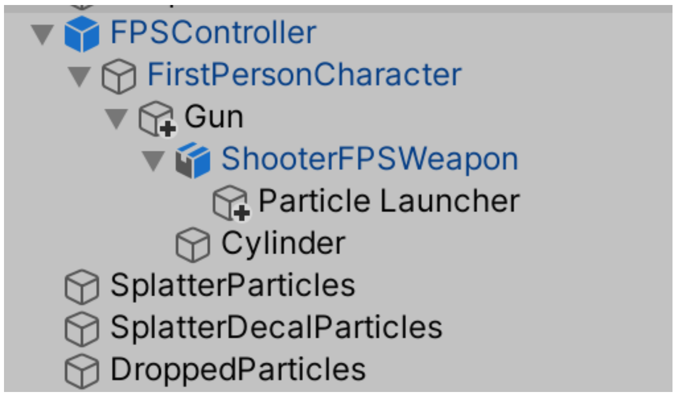


Figure 6.14: Part of the hierarchy of the Game Scene

The "FSPController" and "FirstPersonCharacter" game objects have different scripts that are able to move the camera view following the rotation of the mouse and move the character throughout the game scene with the arrows of the keyboard. The "ShooterFPSWeapon" and the Cylinder game objects are the assets for the arm and the hose. All the way down the "FSPController", there is the Particle Launcher game object

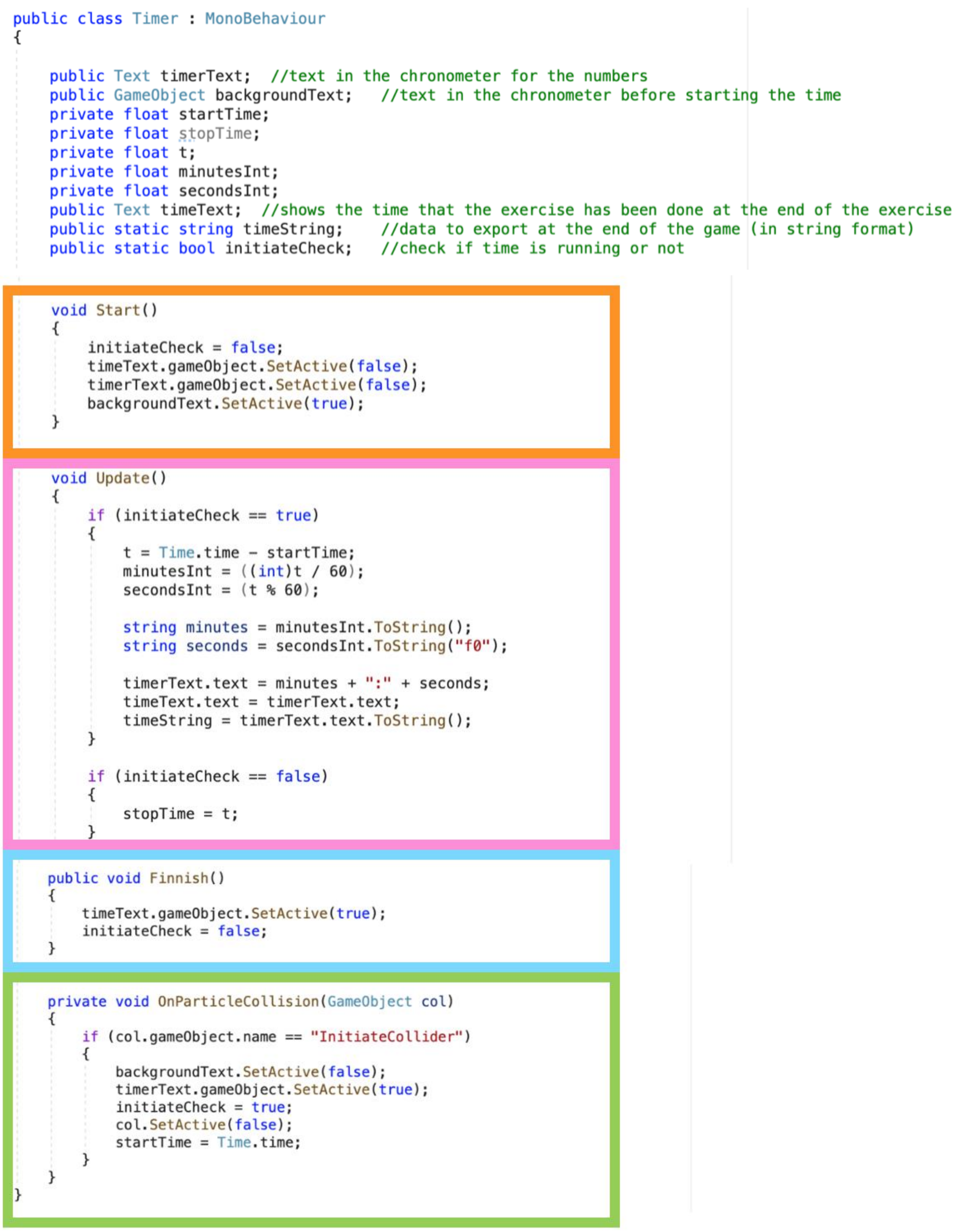
that has a script attached to it called Particle Launcher which allows the player to emit particles when he presses the mouse left button.
The “SplatterParticles” game object is the particle system that is emitting the droplets when the water beam collides with a collider in the scene. And then, there are these two splatter decal particle systems that are “SplatterDecalParticles” and “DroppedParticles”. In turn, they have the “ParticleDecalPool” script, which is what is being used to spawn all of the splatters and droplets on the colliders to display those.
All this scripts (FirstPersonController, ParticleLauncher, SplatOnCollision, ParticleDecalPool, ParticleDecalData) are shown in the Annex C.

There are two scripts (“Timer” and “TimeBar”) to control the chronometer and the time bar of the Game Scene. The Timer script is attached to the Particle Launcher game object. It has a Start function (orange square in figure 6.15) that mentions that the time is still not running (initiateCheck = false) and do not set the text game objects of the time to active yet. It only activates the background text of the chronometer that has the word “Timer”.

The Timer script also contains the information (green square in figure 6.15) that when the particle system collides with a collider called “InitiateCollider” (it is the first collider of the trajectory to follow with the mouse). The background text of the chronometer becomes not active and the text of the numbers becomes active. It also disables the “InitiateCollider” because we want this function to happen only once. It sets the “startTime” variable for starting to count and contains the information that the time is running (initiateCheck = true).

When the time is running (initiateCheck = true), the Update function (pink square in figure 6.15) updates the time of the chronometer. It also updates the value of the “timeText” variable (time result that will be displayed on the canvas when the exercise is over) and the “timeString” variable that transforms this value into a string so that it can be exported to a .csv file.

If the Finnish function is called (blue square in figure 6.15) the “timeText” (time result that will be displayed on the canvas when the exercise is over) becomes active and the “initiateCheck” variable becomes false. So, in the Update function, the time stops running.

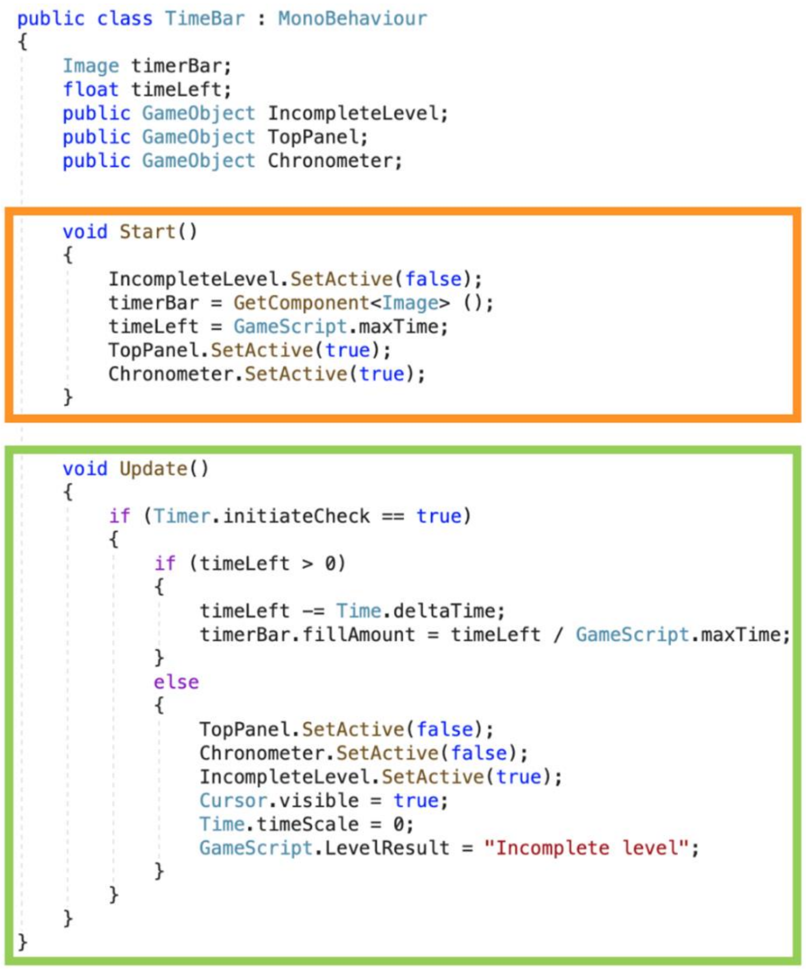


```
public class Timer : MonoBehaviour
{

    public Text timerText;  //text in the chronometer for the numbers
    public GameObject backgroundText;   //text in the chronometer before starting the time
    private float startTime;
    private float stopTime;
    private float t;
    private float minutesInt;
    private float secondsInt;
    public Text timeText;  //shows the time that the exercise has been done at the end of the exercise
    public static string timeString;    //data to export at the end of the game (in string format)
    public static bool initiateCheck;   //check if time is running or not

    void Start()
    {
        initiateCheck = false;
        timeText.gameObject.SetActive(false);
        timerText.gameObject.SetActive(false);
        backgroundText.SetActive(true);
    }

    void Update()
    {
        if (initiateCheck == true)
        {
            t = Time.time - startTime;
            minutesInt = ((int)t / 60);
            secondsInt = (t % 60);

            string minutes = minutesInt.ToString();
            string seconds = secondsInt.ToString("f0");

            timerText.text = minutes + ":" + seconds;
            timeText.text = timerText.text;
            timeString = timerText.text.ToString();
        }

        if (initiateCheck == false)
        {
            stopTime = t;
        }

    public void Finnish()
    {
        timeText.gameObject.SetActive(true);
        initiateCheck = false;
    }

    private void OnParticleCollision(GameObject col)
    {
        if (col.gameObject.name == "InitiateCollider")
        {
            backgroundText.SetActive(false);
            timerText.gameObject.SetActive(true);
            initiateCheck = true;
            col.SetActive(false);
            startTime = Time.time;
        }
    }
}
```

Figure 6.15. Clean the Horse Game: “Timer” script

Then, there is the TimeBar script to control the time bar of the Game Scene. The Start function (orange square in figure 6.16) of this script activates or inactivates the desired game objects. It also sets the timeLeft variable the value of the maxTime specified in the script GameScript. Depending on the shape or difficulty selected, the maxTime (time allowed to do the exercise) is higher or lower.

The Update function (green square in figure 6.16) is initiated when the chronometer is running (Timer.initiateCheck = true). If the time bar has not run out yet, it decreases. If the time bar arrives to its end because it has run out, the time stops, the IncompleteLevel panel of the canvas appears and the LevelResult variable defined in the script GameScript changes from "Level not finished" to "Incomplete level".

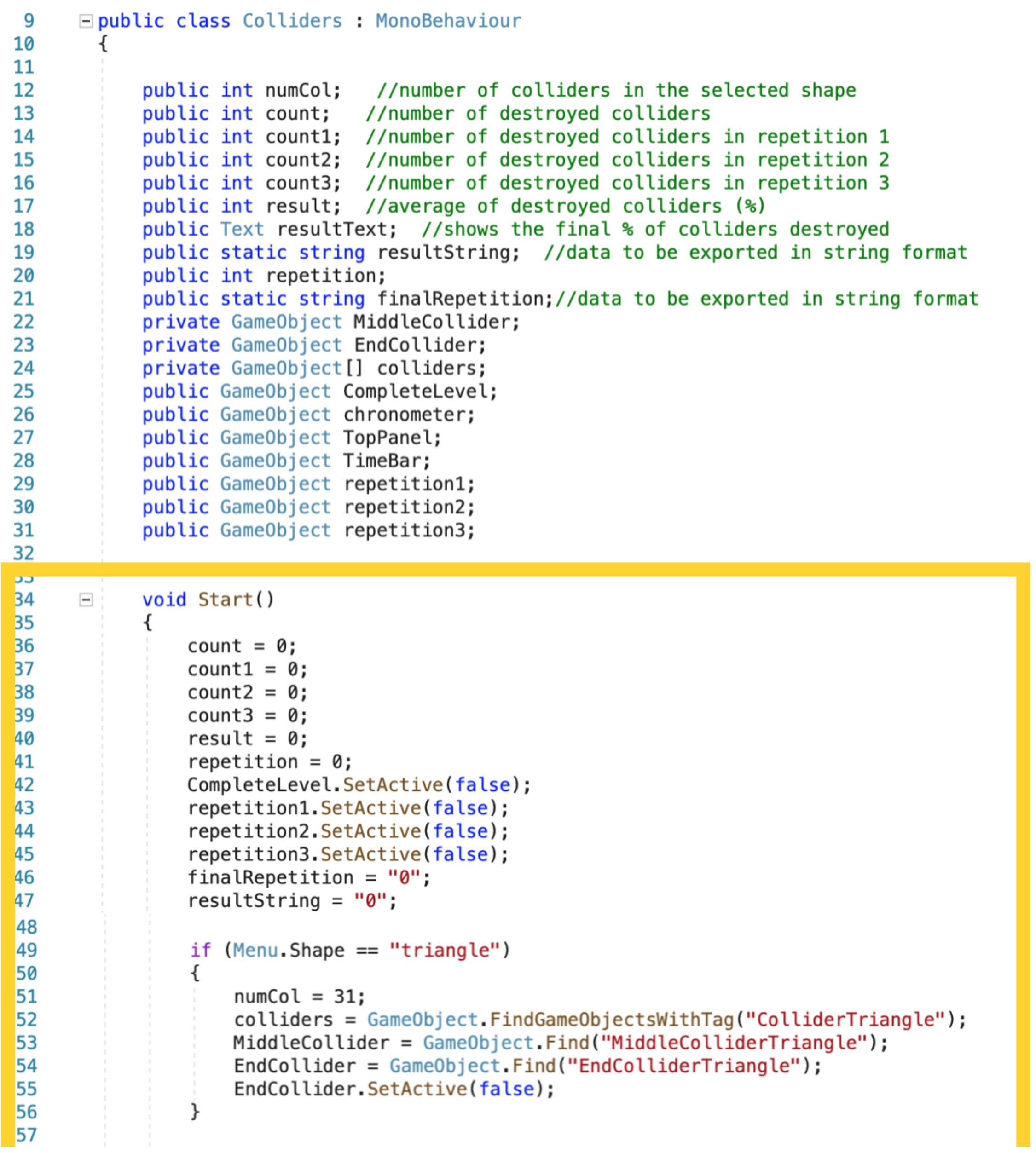


```
public class TimeBar : MonoBehaviour
{
    Image timerBar;
    float timeLeft;
    public GameObject IncompleteLevel;
    public GameObject TopPanel;
    public GameObject Chronometer;

    void Start()
    {
        IncompleteLevel.SetActive(false);
        timerBar = GetComponent<Image> ();
        timeLeft = GameScript.maxTime;
        TopPanel.SetActive(true);
        Chronometer.SetActive(true);
    }

    void Update()
    {
        if (Timer.initiateCheck == true)
        {
            if (timeLeft > 0)
            {
                timeLeft -= Time.deltaTime;
                timerBar.fillAmount = timeLeft / GameScript.maxTime;
            }
            else
            {
                TopPanel.SetActive(false);
                Chronometer.SetActive(false);
                IncompleteLevel.SetActive(true);
                Cursor.visible = true;
                Time.timeScale = 0;
                GameScript.LevelResult = "Incomplete level";
            }
        }
    }
}
```

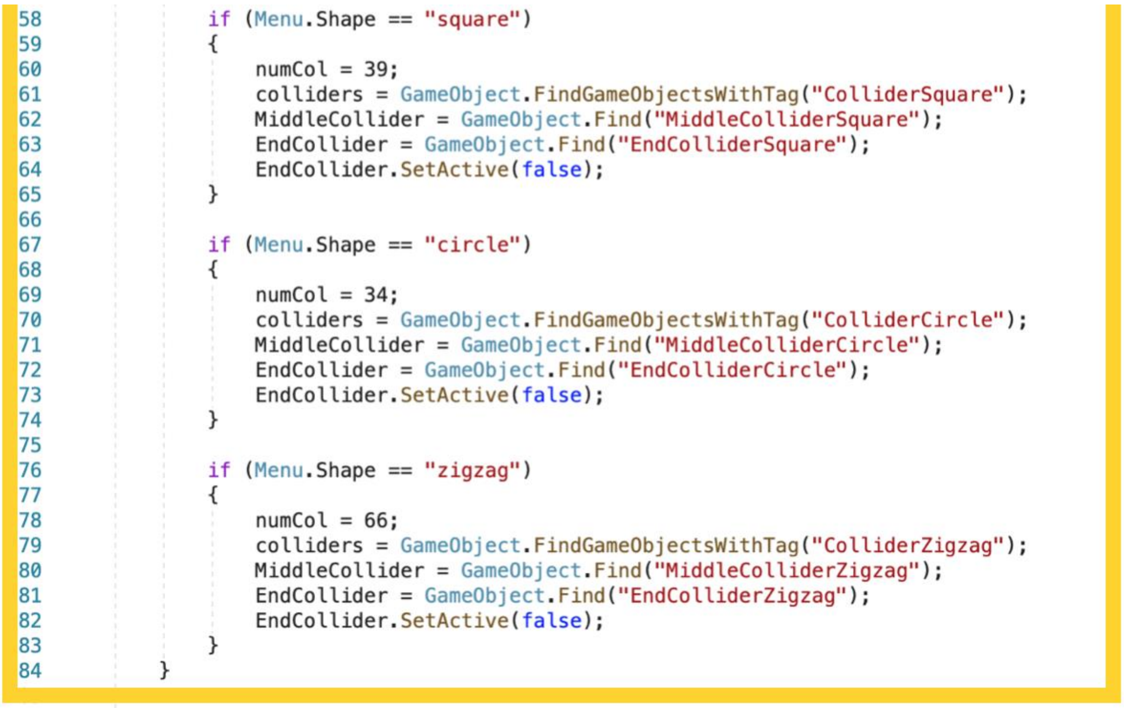
Figure 6.16. Clean the Horse Game: “TimeBar” script

Furthermore, we have the Colliders script that is attached to the Particle Launcher game object. This script contains the basics of the game. The Start function (yellow square figure 6.17) sets to zero the value of some variables and disables certain game objects. It also assigns the value to the following variables depending on the shape chosen in the menu and disables the *EndCollider*. The variables are:

- **numCol:** number of colliders that has the chosen shape.
- **Colliders:** group of colliders that form the path to follow the shape chosen.
- **MiddleCollider:** the collider placed in the middle of the shape chosen to follow.
- **EndCollider:** the collider placed at the end of the shape chosen to follow.



```
public class Colliders : MonoBehaviour
{

    public int numCol;   //number of colliders in the selected shape
    public int count;   //number of destroyed colliders
    public int count1;  //number of destroyed colliders in repetition 1
    public int count2;  //number of destroyed colliders in repetition 2
    public int count3;  //number of destroyed colliders in repetition 3
    public int result;  //average of destroyed colliders (%)
    public Text resultText;  //shows the final % of colliders destroyed
    public static string resultString;  //data to be exported in string format
    public int repetition;
    public static string finalRepetition;//data to be exported in string format
    private GameObject MiddleCollider;
    private GameObject EndCollider;
    private GameObject[] colliders;
    public GameObject CompleteLevel;
    public GameObject chronometer;
    public GameObject TopPanel;
    public GameObject TimeBar;
    public GameObject repetition1;
    public GameObject repetition2;
    public GameObject repetition3;


    void Start()
    {
        count = 0;
        count1 = 0;
        count2 = 0;
        count3 = 0;
        result = 0;
        repetition = 0;
        CompleteLevel.SetActive(false);
        repetition1.SetActive(false);
        repetition2.SetActive(false);
        repetition3.SetActive(false);
        finalRepetition = "0";
        resultString = "0";

        if (Menu.Shape == "triangle")
        {
            numCol = 31;
            colliders = GameObject.FindGameObjectsWithTag("ColliderTriangle");
            MiddleCollider = GameObject.Find("MiddleColliderTriangle");
            EndCollider = GameObject.Find("EndColliderTriangle");
            EndCollider.SetActive(false);
        }

```



```
        if (Menu.Shape == "square")
        {
            numCol = 39;
            colliders = GameObject.FindGameObjectsWithTag("ColliderSquare");
            MiddleCollider = GameObject.Find("MiddleColliderSquare");
            EndCollider = GameObject.Find("EndColliderSquare");
            EndCollider.SetActive(false);
        }

        if (Menu.Shape == "circle")
        {
            numCol = 34;
            colliders = GameObject.FindGameObjectsWithTag("ColliderCircle");
            MiddleCollider = GameObject.Find("MiddleColliderCircle");
            EndCollider = GameObject.Find("EndColliderCircle");
            EndCollider.SetActive(false);
        }

        if (Menu.Shape == "zigzag")
        {
            numCol = 66;
            colliders = GameObject.FindGameObjectsWithTag("ColliderZigzag");
            MiddleCollider = GameObject.Find("MiddleColliderZigzag");
            EndCollider = GameObject.Find("EndColliderZigzag");
            EndCollider.SetActive(false);
        }
    }
```

Figure 6.17. Clean the Horse Game: First part of the “Colliders” script.

The *Colliders* script, apart from having the function *Start*, also has the function *OnParticleCollision* which is shown in the figure 6.18 below. This function is activated when the particle system (the water of the hose) collides with the colliders placed all over the shape selected in the Menu Scene.

If it collides with a normal collider (meaning it is not the *MiddleCollider* or the *EndCollider*), the variable *count* increases one value and the collider is disabled to avoid colliding with it another time. The *count* variable counts the number of colliders that the patients collides.

If it collides with the *MiddleCollider,* the *EndCollider,* that was disabled in the *Start* function, becomes active. Otherwise, if the *EndCollider* was active from the beginning, the patients would touch it when they start to trace the trajectory.

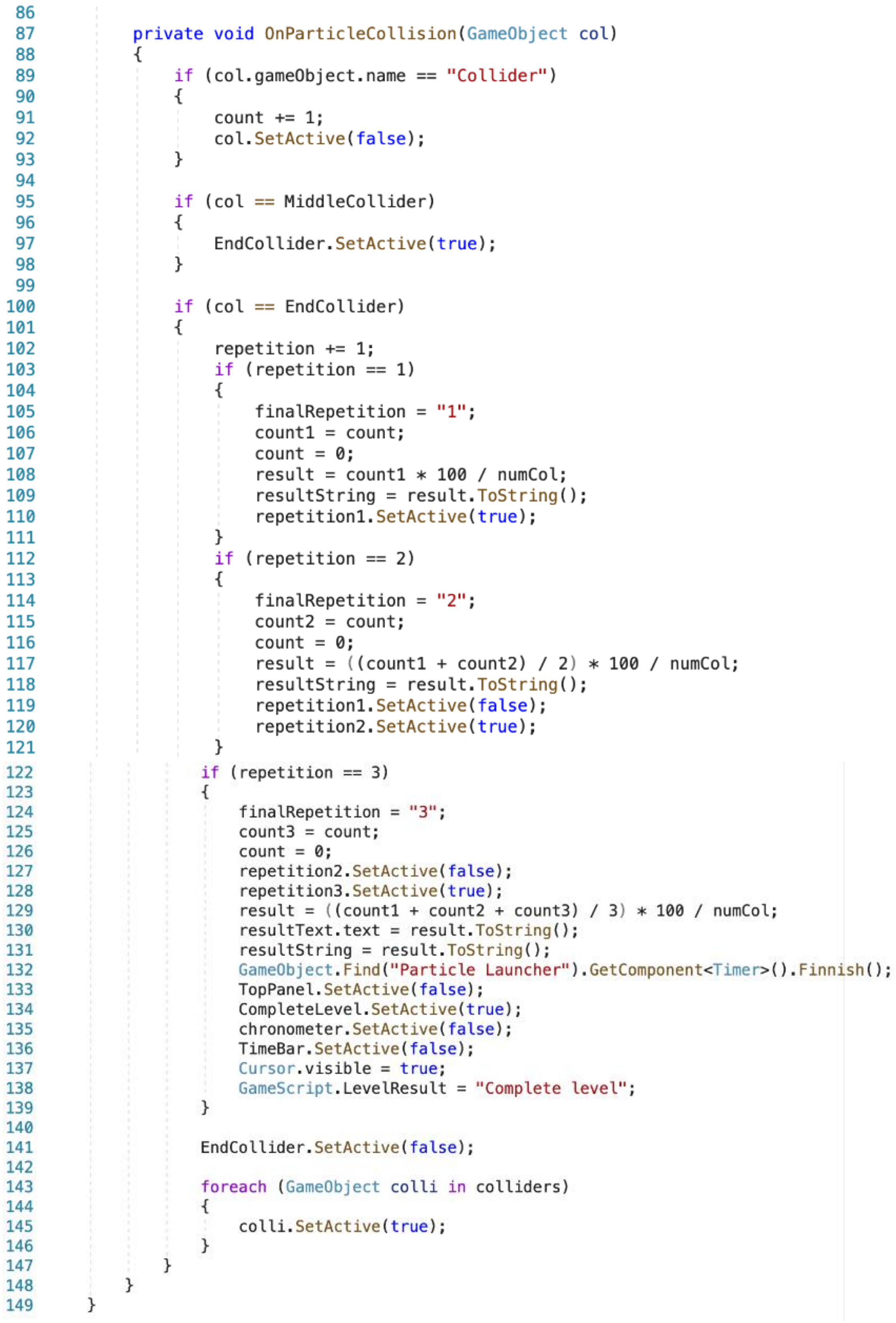
If it collides with the *EndCollider,* the *repetition* variable increases one value because it means that the patient has finished one repetition of the trajectory. If the patient has done the first repetition, the variable *finalRepetition* gets value 1. This variable is used to export the number of repetitions that the patient has done in the exercise. If the patient has done two repetitions, the variable *finalRepetition* gets the value of 2. And if the patient has done three repetitions, the variable *finalRepetition* gets the value of 3.

Then, if the patient has done one repetition the *count1* variable takes the value of the *count* variable and the *count* variable is set again to zero. The same happens if the patient has done two or three repetitions of the exercise. In this way we can know the number of colliders that the patient has destroyed in each repetition. After each repetition, the colliders are activated again and the *EndCollider* is deactivated.

In each repetition, the accuracy is also calculated in percentage of the exercise by doing the average of colliders touched. This value is saved in the *result* variable. The *resultString* variable is the same value but in a string format to be able to extract this data to a .csv document.

In each repetition, the game objects *repetition1*, *repetition2* or *repetition3* are also activated depending on the number of repetitions that the patient has done during the exercise. These game objects consists of an image of the number 1, 2 or 3 corresponding with the last repetition that the patient has done.

To end with the comments on the *Colliders* script, if the patient has done the three repetitions of the exercise, the game finishes. So, the *Finnish* function of the *Timer* script is activated, and the time stops. The *CompleteLevel* panel of the canvas is also activated and the *LevelResult* variable from the script *GameScript* changes from “Level not finished” to “Complete level”.



```
    private void OnParticleCollision(GameObject col)
    {
        if (col.gameObject.name == "Collider")
        {
            count += 1;
            col.SetActive(false);
        }

        if (col == MiddleCollider)
        {
            EndCollider.SetActive(true);
        }

        if (col == EndCollider)
        {
            repetition += 1;
            if (repetition == 1)
            {
                finalRepetition = "1";
                count1 = count;
                count = 0;
                result = count1 * 100 / numCol;
                resultString = result.ToString();
                repetition1.SetActive(true);
            }
            if (repetition == 2)
            {
                finalRepetition = "2";
                count2 = count;
                count = 0;
                result = ((count1 + count2) / 2) * 100 / numCol;
                resultString = result.ToString();
                repetition1.SetActive(false);
                repetition2.SetActive(true);
            }
            if (repetition == 3)
            {
                finalRepetition = "3";
                count3 = count;
                count = 0;
                repetition2.SetActive(false);
                repetition3.SetActive(true);
                result = ((count1 + count2 + count3) / 3) * 100 / numCol;
                resultText.text = result.ToString();
                resultString = result.ToString();
                GameObject.Find("Particle Launcher").GetComponent<Timer>().Finnish();
                TopPanel.SetActive(false);
                CompleteLevel.SetActive(true);
                chronometer.SetActive(false);
                TimeBar.SetActive(false);
                Cursor.visible = true;
                GameScript.LevelResult = "Complete level";
            }

            EndCollider.SetActive(false);

            foreach (GameObject colli in colliders)
            {
                colli.SetActive(true);
            }
        }
    }
}
```

Figure 6.18. Clean the Horse Game: Second part of the “Colliders” scripts.

Lastly, there are two scripts called *CSVManager* and *MyTools* that are the ones in charge of exporting the data. These scripts are going to be explained in section 6.3.

The other scripts that are used in the Game Scene were imported with the assets downloaded. For instance, the scripts used for the animation of the horse or the camera view. These assets and scripts are placed in the folders *AssetNatureStarterKit2*, *AssetParticleScripting* and *AssetHorseAnimsetProRidingSystem*.

6.1.4 Actions of the objects used by the patient

In the Menu Scene, the patient interacts with the button components of the scene because he/she has to choose the shape (triangle, square, circle or zigzag) and the difficulty (slow or fast) desired, press the PLAY button to start the game or press the EXIT button to return to the main menu where all the games are placed.

In the Game Scene, the patient can interact in different ways. He/she can move through the scene with the arrows of the keyboard. The patient can also rotate the camera view with the mouse position. And the patient is able to shoot water with the hose if he/she presses the left button of the mouse.

6.1.5 Data exported

There are two scripts in charge of exporting the data: “CSVManager” and “MyTools”. In the Update function of “MyTools” script, when the space key is pressed, the *AppendToReport* function from the “CSVManager” script is activated. This function exports the shape (triangle, square, circle or zigzag) and difficulty (slow or fast) selected in the menu, the result of the level (level not finished, incomplete level or complete level), the number of repetitions that the patient has done of the exercise, the accuracy of the exercise and the time it took the patient to do the exercise.

6.2 Mirror Cleaning

6.2.1 VR application idea

The application main idea is to make the user perform a circular movement with their shoulders using the Steam VR controllers. So, to make that possible this project addresses the idea of recreating a typical situation where somebody needs to clean the mirror with a cleaning spray because this is dirty. They will be immersed in the game being represented by a robot as their character. This robot will simulate their moves and they will be able perform the spray action pressing the trigger button like in a real situation.

This application is not paired with the Kinect to track the moves, so it is necessary to create a game that get some variables out to make its subsequent analysis. In this case, it has been decided to consider the cleaning speed and the accuracy. With these variables, the professional will be able to detect the improvement of their mobility over time.

As commented, this exercise is based to perform a shoulder circular movement (figure 6.19). It could be a variant making a movement with the wrist (figure 6.20). This last option is just hypothetic because there has been no possibility to test it with the SteamVR HTC Vive equipment.

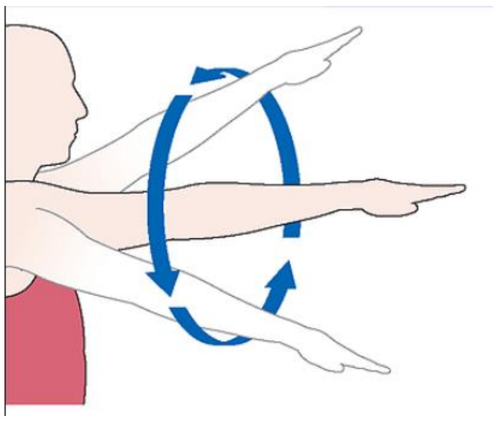


Figure 6.19. Shoulder circular movement. [https://naveedmazumder.wixsite.com/]

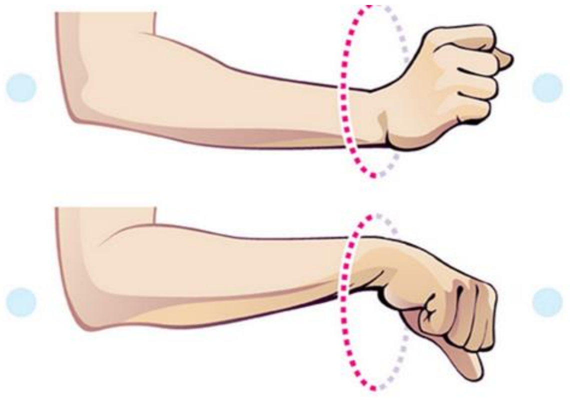


Figure 6.20. Wrist circular movement. [https://experiencelife.com/]

6.2.2 Graphic environment design

To perform a real experience based on a daily routine this application will be recreated in a conventional bathroom. This environment will make patients feel more comfortable because it has been created with realistic assets and the ambient music is so relaxing.

To make a practical bathroom it is necessary to use 3d assets in the most realistic way possible, so in the scene all the furniture has been exported from a 3d house assets website [REF]. The advantages of using this website is that it offers all kinds of designs, textures and so on. The assets found in the Unity Asset Store are too poor in graphics. However, there is a disadvantage taking those assets which is that they are heavier than Unity Asset Store assets, because of their geometry. It was thought that this could affect the FPS when the game is tested, but the applications runs perfectly. The final result is shown in the following images.

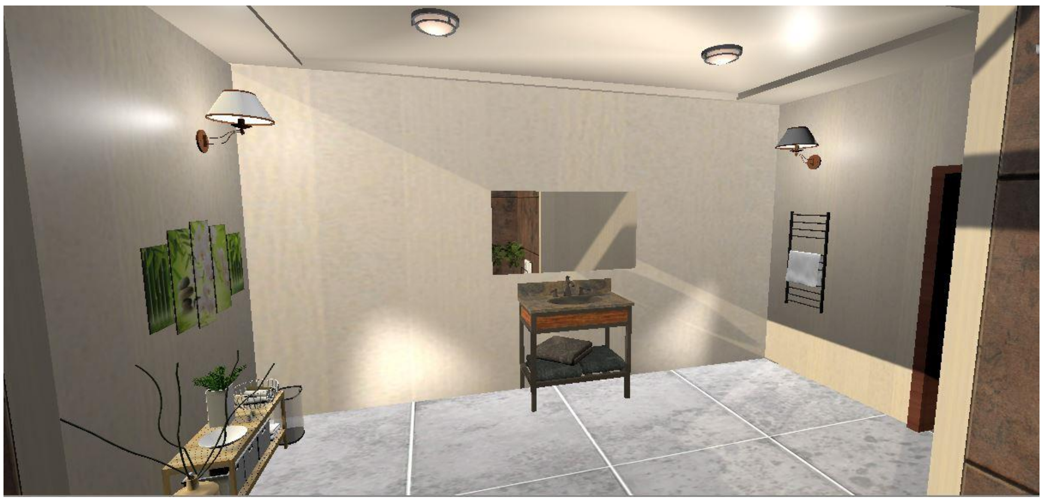


Figure 6.21. Bathroom Game environment: Mirror perspective.

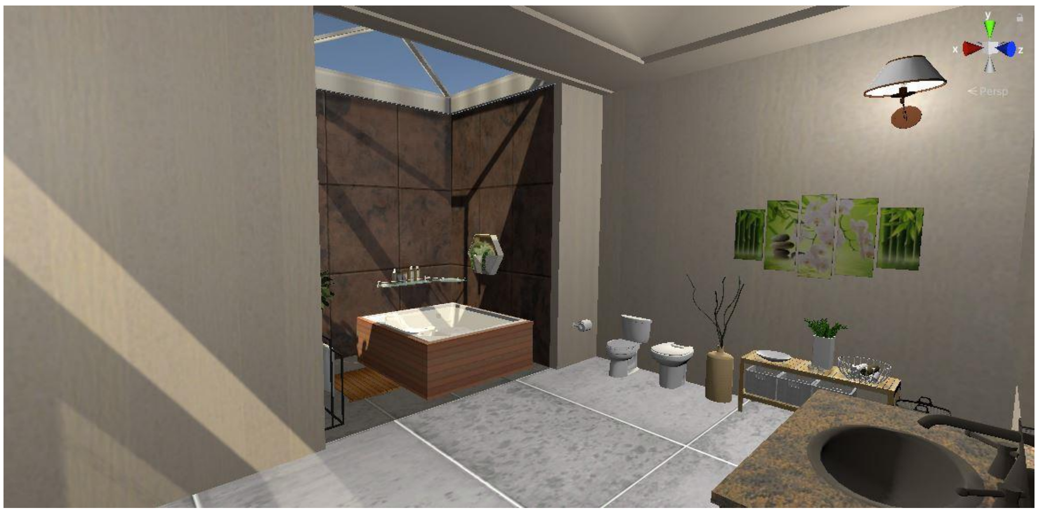


Figure 6.22. Bathroom Game environment: Assets.

The only asset that was not taken from that site was the mirror, because it needs a little bit of scripting to recreate the mirror effect. It also changes its resolution depending if the mirror is already cleaned or it is not, but this subject will be commented later in the script explanation section. As you can see, the mirror reflects perfectly the environment being this the final result after playing the game.

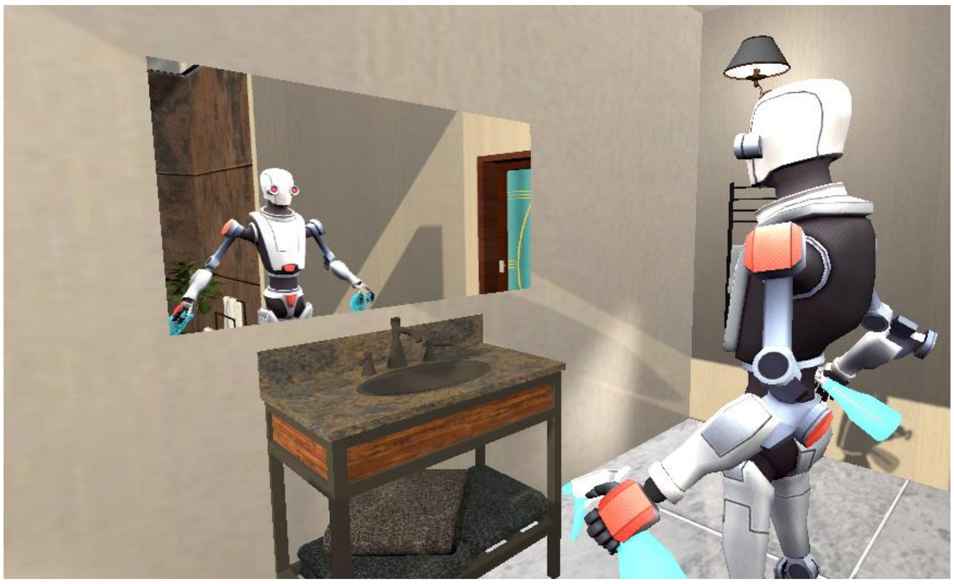


Figure 6.23. Bathroom Game environment: Mirror reflection.

6.2.3 Game Character

Usually, in VR games are made just with both hands and a head asset where there is no need to interact with the body. However, in this case it was decided to use a full body asset to be able to recognize the arms moves. So, a free full body asset that Unity offers is "Robot Kyle". This asset can be configured in such a way that allows the wrists moves to be detected (where the controllers will be attached). Using this asset, we can perform all the patient's actions that we want to.

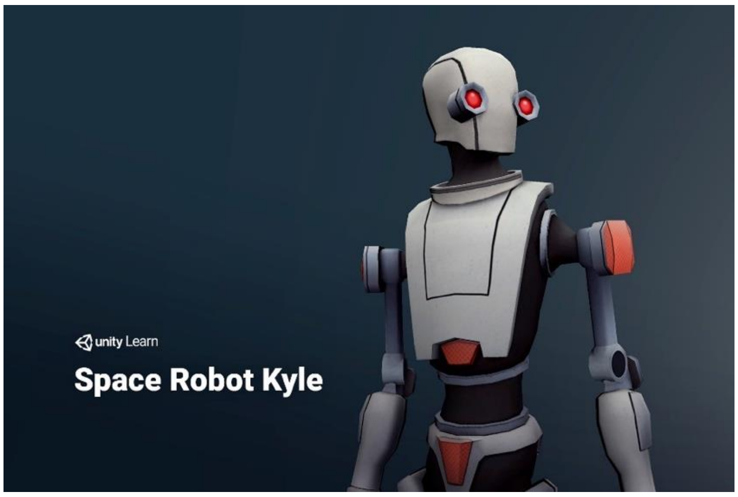


Figure 6.24. Unity full body free asset: Robot Kyle.

6.2.4 Character Setup

This section has been divided in four parts, which explains how to configure this asset to reach its objectives.

IK Constraints

Inverse Kinematics (IK) is a mathematical process which is used to give the relative direction and the orientation of the joints caused by the action of another one. The difference between IK and FK (Forward Kinematics) is that in FW there is no response from the joints when another one is moving in orientation concepts; they just stay still. In this case, that is not useful for this application, because the robot needs to move the whole arm to perform all the trajectory to clean the mirror. In this case, IK will calculate how to move the forearm and the upper arm respectively from the wrist move (where the controllers are located). These calculations are commonly used in robotic applications, and in this case the main character in this application is a robot, so it has been a little bit of a coincidence in this aspect.

There is a class in Unity that allows the tracking of all the character skeleton which is called "Two Bone IK Constraints". By default, the robot asset comes with all its skeleton divided by joints. So, applying this class into the character is possible to achieve a full setup of its joints, as you can see in figure 6.25.

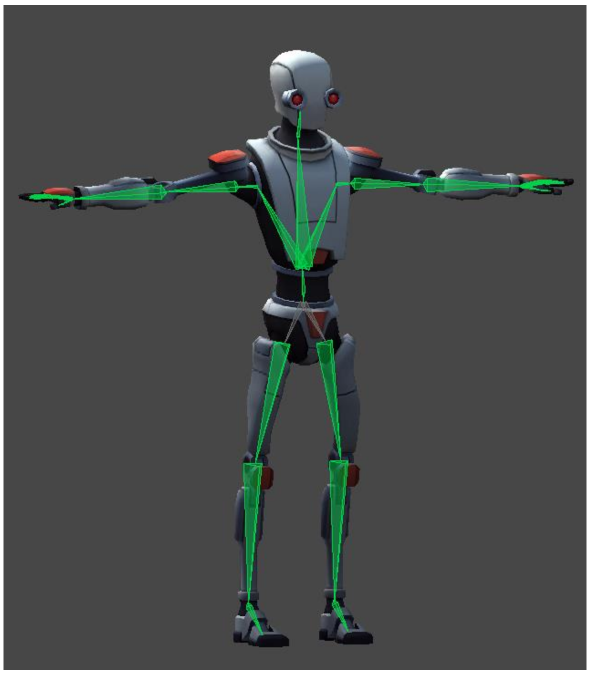
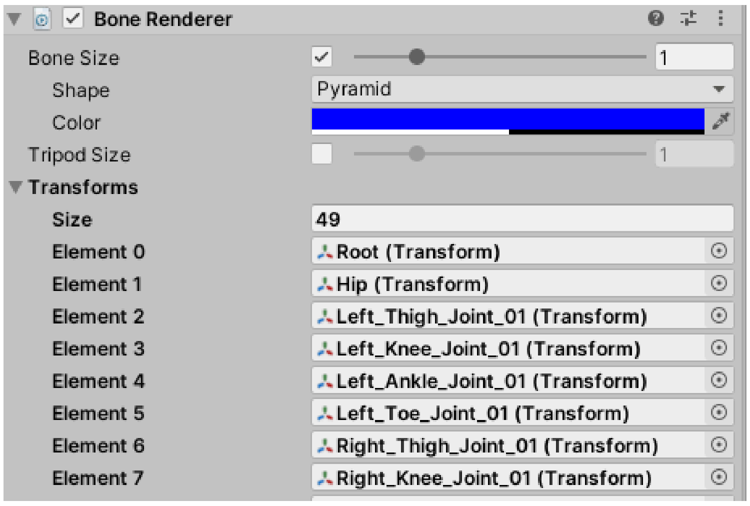


Figure 6.25. Robot Kyle: Two Bone IK Constraints.



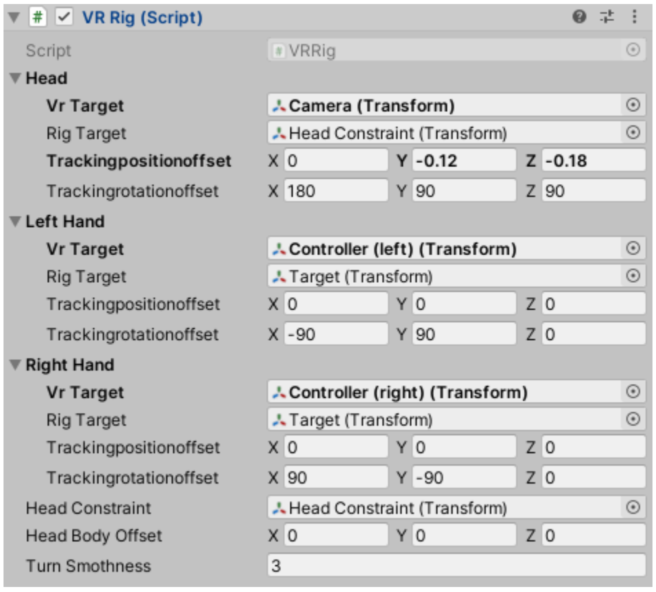
Bone Renderer	
Bone Size	✓ 1
Shape	Pyramid
Color	
Tripod Size	1
Transforms	
Size	49
Element 0	Root (Transform)
Element 1	Hip (Transform)
Element 2	Left_Thigh_Joint_01 (Transform)
Element 3	Left_Knee_Joint_01 (Transform)
Element 4	Left_Ankle_Joint_01 (Transform)
Element 5	Left_Toe_Joint_01 (Transform)
Element 6	Right_Thigh_Joint_01 (Transform)
Element 7	Right_Knee_Joint_01 (Transform)

Figure 6.26 Robot Inspector: Bone Renderer component.

Now, the next step is to make the arm bones and the head move naturally. To do that, two constraints linked to each arm have been created to allow joining the bones position and rotation relatively. Legs will not be configured because it is an upper-body exercise.

Camera Rig

To be able to represent all the movements from the patient it is necessary to use the Steam VR plugin. In this package, there is a prefab called "Camera Rig" which is constituted by the head camera and the hand controllers. The next step is to drag this prefab to the scene and link the Steam components to the script created before. To correctly position the hands and the head orientations it is necessary to change the values for each axis. Figure 6.26 shows how this script is configured.



VR Rig (Script)			
Script	VRRig		
Head			
Vr Target	Camera (Transform)		
Rig Target	Head Constraint (Transform)		
Trackingpositionoffset	X 0	Y -0.12	Z -0.18
Trackingrotationoffset	X 180	Y 90	Z 90
Left Hand			
Vr Target	Controller (left) (Transform)		
Rig Target	Target (Transform)		
Trackingpositionoffset	X 0	Y 0	Z 0
Trackingrotationoffset	X -90	Y 90	Z 0
Right Hand			
Vr Target	Controller (right) (Transform)		
Rig Target	Target (Transform)		
Trackingpositionoffset	X 0	Y 0	Z 0
Trackingrotationoffset	X 90	Y -90	Z 0
Head Constraint	Head Constraint (Transform)		
Head Body Offset	X 0	Y 0	Z 0
Turn Smothness	3		

Figure 6.27. Robot Kyle: Inspector script.

Selector pointer

UI design section shows how the menu is, however, to explain this part it will be necessary to previously comment how the UI elements interact. In this case, they will work by a VR pointer actions. This VR pointer is made by a line renderer (attached to a Raycast) and a small sphere to help users to see what they are aiming to. The pointer range is determined by the "Raycast". When this collides with the UI elements and other game objects, the sphere object will appear over them, limiting pointer line. That will be useful to not select another object behind the real target. The scripts related with this pointer are attached in Annex D. The pointer will be controlled by the right controller which is aligned with the robot right hand too.

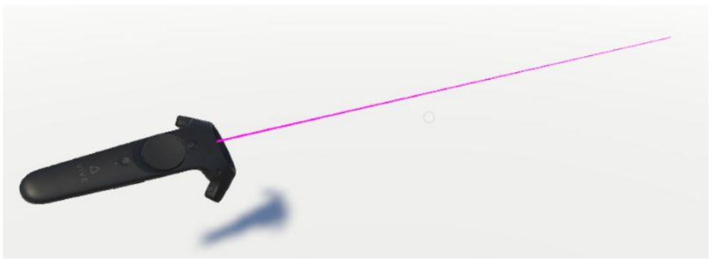


Figure 6.28. Pointer attached to the SteamVR right controller.

6.2.5 Game Objects attached to the character

Obviously, as mentioned in the idea section, the purpose of the game is to clean a mirror, so the main objects to use in this case are cleaning sprays. These objects will be aligned with the robot hands and they will transform their position and rotation according to the controller's movement. As mentioned, the robot prefab has all its joints available to work with, so to make this more realistic both hands have been modeled to look like it is holding that object.

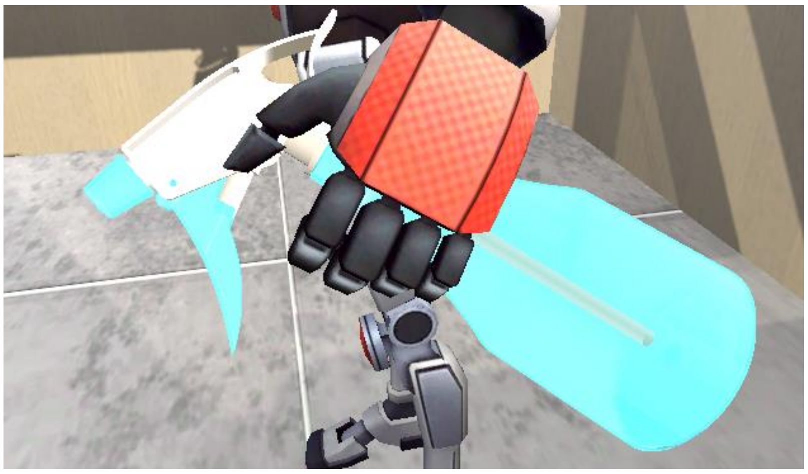


Figure 6.29. Robot Kyle with spray's hand poses.

There are two functions related with this object, both are related with the liquid that will come out from the spray. One has the aesthetic function and the other one will be used to calculate necessary values to the subsequent analysis.

Spray's Particle System

Particles system simulates a water effect with small images, in this case these particles are small 3D spheres similar as a liquid spray. There is an empty game object aligned in front of each spray which is called the "particle launcher" which will be the starting point where the particles will be emitted. To prevent the user from missing the aim during the game, these particles will come out in a straight-line direction.

Each particle system has a collider option in its inspector where it is possible to set some collision effects when that happens. That will be useful for the users to see which is the trajectory they are doing during the process, like a painting game. These particles stacked over the mirror will not disturb the game course, because they are not too big.

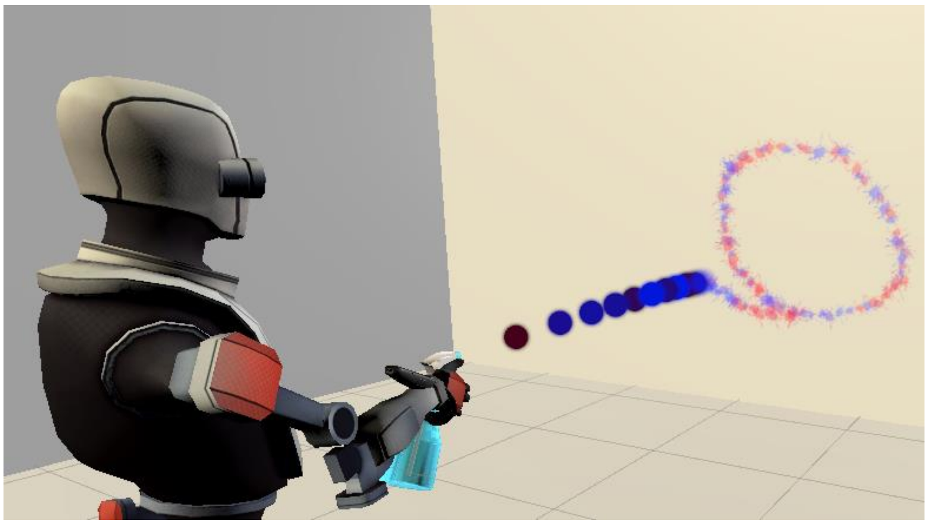


Figure 6.30. Spray's Particle System and Decal Particles

Spray's Raycast

Following the same particle system path there will be an invisible Raycast inside it. This is the second and the last Raycast in the game, however this one is not made to select UI elements. In this case, it is used to detect the colours of the pixels where the patient is pointing. This part is better explained in the game section, which introduces the image that the user must follow to succeed in the game. But, to get an idea about the spray is setup, it is important to highlight this element.

6.2.6 User Interface (UI) design

In this game, there are different kind of UI elements like buttons, text, raw images and so on. They have been customized changing their materials, textures, and text fonts to be more creative. So, this section is divided in different parts to explain each important zone of the interface.

All those parts are interactable for the user. Depending on the situation it will be with the pointer or with the spray particle system. All the UI are taken from the Unity default options, there is no new UI assets incorporated.

Immersed menu

A different menu concept has been created in this game, compared with the horse game. In this case, there is a menu full immersed in the game (i.e. there is not another scene in the game with a previous menu). That idea had been taken from a VR games company in USA called “Stress Level Zero” based in “Hover Junkers” game. Just starting the game, the users are inside the VR world adapting the game according to their needs. That was so inspiring and that is why this was the choice for this menu game, shown in figure 6.32. Obviously, the game menu is not as spectacular as the Hover Junker is, shown in figure 6.31, but the main idea has been applied.

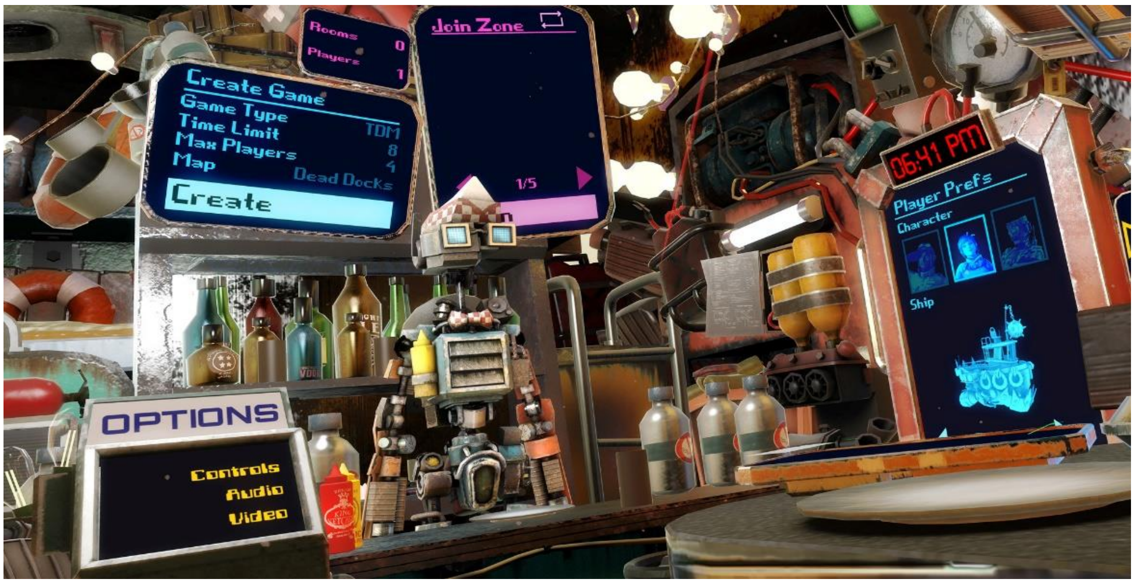
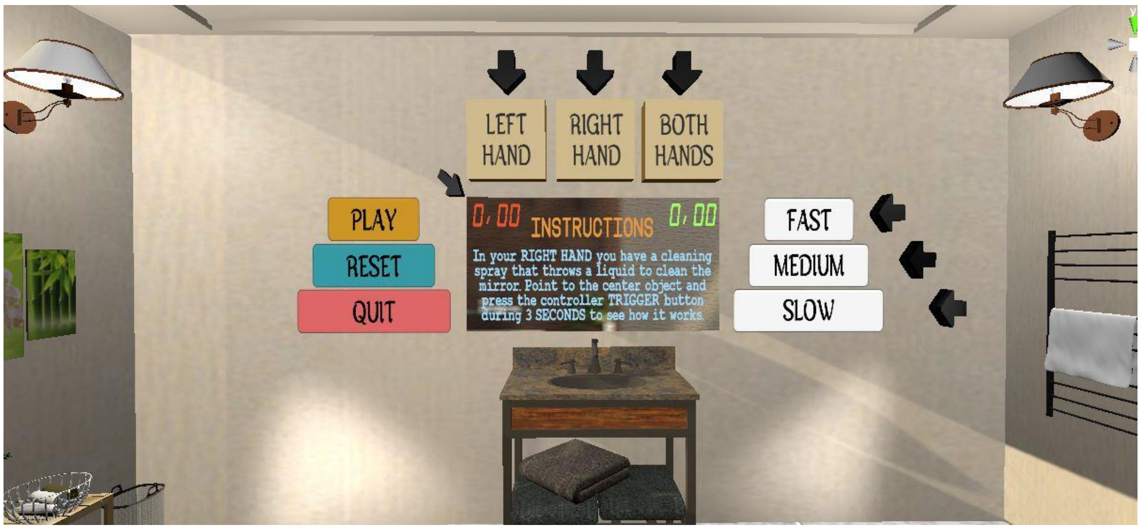


Figure 6.31. Example of Hover Junker’s Menu.
[https://store.steampowered.com/]



In your RIGHT HAND you have a cleaning spray that throws a liquid to clean the mirror. Point to the center object and press the controller TRIGGER button during 3 SECONDS to see how it works.

Figure 6.32. Bathroom Game Menu and the dirty mirror.

It can be observed, in both images the UI elements are distributed all over the scene. The patients will be able to interact with them with the selector pointer, mentioned previously. In Hover Junker's game, when the users press the "Create" button, the scene is changed to the game screen and then they start playing. In our game, instead of changing the scene, the menu just disappears, and the users stay in the same room, in this case the bathroom. This is especially important to not have so many elements during the course of the game and, in this way, the user can get focused in the main goal.

It can be observed, there are different zones in our menu compound by "conventional buttons (Play, Reset and Quit) and the game options buttons like which hand the patient would like to use (Right, Left or Both hands) or the gameplay speed (Fast, Medium or Slow). There is a sequence before starting the game to select all the options indicated by the "Instructions" text and by the arrow indicators. During the process, both components will be updated to help the user follow the entire game-setup sequence correctly until the last step.

Instructions and arrow indicators

These two components will be in charge of indicating at all times which are the points that are important to observe in each step. "Instructions" component is a text UI that changes its text attribute depending on the point where the patient is. It is set in a place where it is easy to read and with a big font size.

Arrow indicators have the mission to not lose the path to followed to reach the last point. So, during the selecting steps arrows will be appearing and disappearing to make it easy and get the attention from the patients. Those arrows are animated with a mechanical animator and changes their position to make them more visible. Their colour is black to get a good contrast with the background.

Hand Selector

When the game is opened, it appears the main character and the environment just with the dirty mirror and an animated button mentioning, “Clean It”. After pressing it, the first option will appear to select with its respective arrow indicators and the instructions text updated, explaining what they mean. As you have seen in figure 6.32, this section is in the highest part of the wall, above the mirror. These options will determine which hand will be selected for the patient to perform the game.
There are three options available “Right Hand”, “Left Hand” and “Both Hands”, it will depend on the patient. The chosen hand will be available to move with the controller while the other one will not.

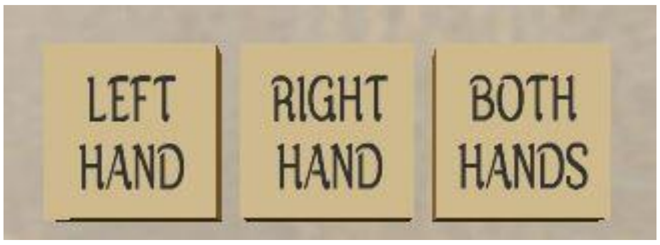


Figure 6.33. Bathroom Game Menu: Hand selector.

Pressing the desired button, it will change its background colour to show that it is clicked. There is also the possibility to swap to another option with no need to reset the game. After pressing the first “Hand Selector” button, the new arrow indicators will appear, and the “Instructions” text will explain how the next step will be carried out.

Level Selector

This is the next step after selecting the hand which will determine the game difficulty. This section is placed in the right side of the mirror composed by three buttons (figure 6.34.). These UI have the same transitions (normal, hover and pressed positions) as the previous buttons.

There are three possible options as well that will affect the time of the game execution that the patients will have to finish it. It means that there is a timer doing a countdown when the game starts and that is the limit time to success in the game. Therefore, the difficulty buttons, based on time selecting, will be “Fast”, “Medium” or “Slow” mode.

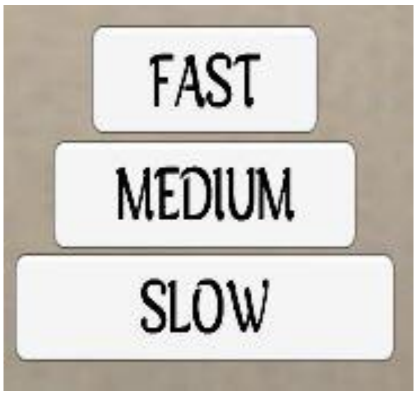


Figure 6.34. Bathroom Game Menu: Level selector.

Selecting one option, the timer arrow indicator will appear showing how the time is changing. If the patients want to select another mode, the timer will update immediately its value. Doing this, they will be able to check which are the values for each option to make the decision easier.

Timers

There are two timers in this game placed in the upper corners of the mirror. One timer has been mentioned in the previous section, which limits the cleaning process, however, the game will not stop if the patients do not reach that. So, that is why there is another timer, doing a chronometer function, to know how long they have taken to carry out the cleaning process. That allows the therapists to check the time of the execution even if there is a failed attempt.

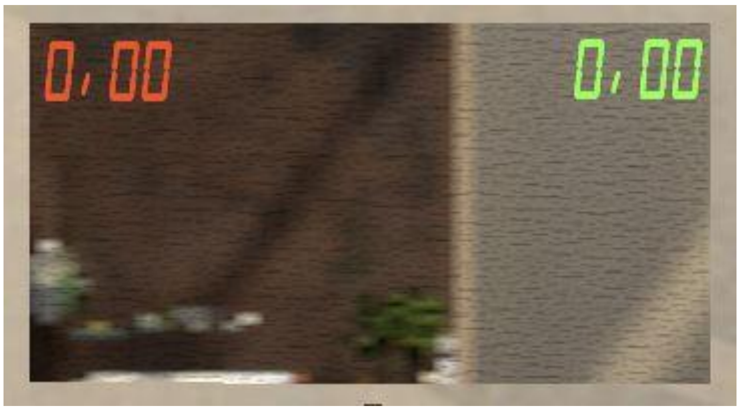


Figure 6.35. Bathroom Game Menu: Timer and chronometer.

Both timers will start at the same time. Actually, they are UI texts changing their text strings in real time. This is possible working in float variables in the script and using the Time class method "DeltaTime", and then change to string each float value. You can check those scripts in the timer's section in Annex D.

Play, Reset and Quit

Usually the game menus have these three buttons in their interfaces. So, this would not be an exception. They are placed in the left side of the mirror (figure 6.32). First of all, the "Play" button has a lot of functions when it has been clicked:

- Make the pointer disappear.
- Enable the cleaning spray particle system.
- Update instructions to explain how the spray works.
- Start a tutorial aiming an object in the middle of mirror with the spray.
- Make all the menu elements disappear to make the game less distractive.

All these methods are in the "PlayButtonFunctions" in Annex D.

Secondly, "Reset" button has just one function which is reset all the game options, previously selected, to create a new game configuration when the game has finished. Finally, "Quit" button is programmed to return to the main platform where all the other project games are located. This option is available when the game has been started and when it finishes.

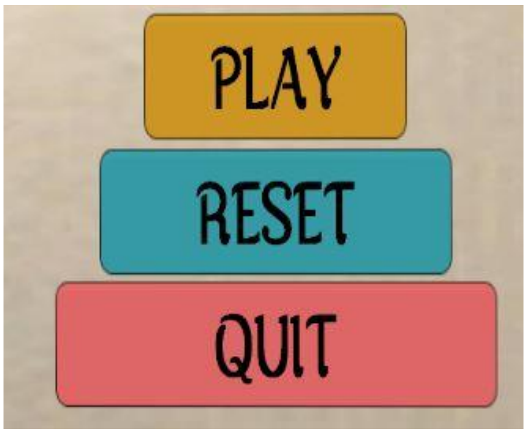
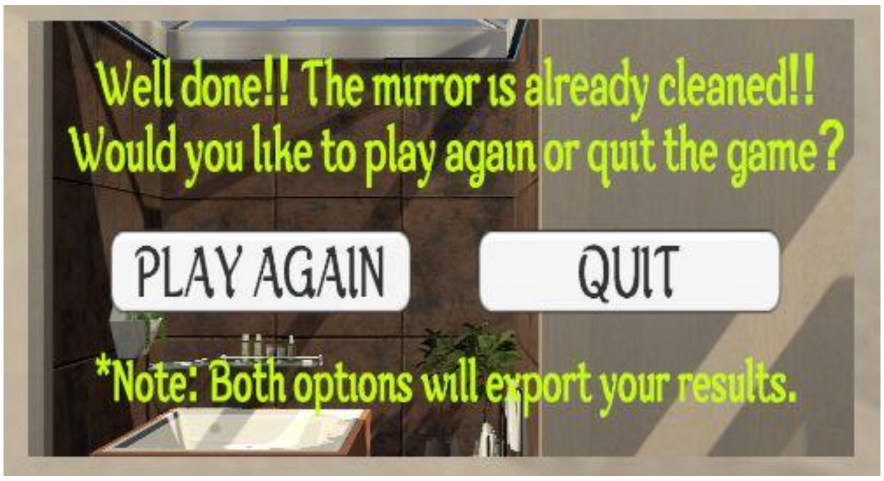


Figure 6.36. Bathroom Game Menu: Play, Reset and Quit Buttons.

Final Scene

As its name says, this is final step done with UI elements that appears in the middle of the mirror when this is already cleaned (figure 6.37.). It is a composition of two buttons and a text. The text mentions the game has been completed and the buttons ask if the patients want to play again or quit the game. Both options will export the data recollected during the game in a .csv file.

When the "Play again" button is pressed, the game starts from the tutorial where all the options are selected, because it is easier to make more rounds with no need to select the options again. And the "Quit" button has the same goal as the other "Quit" button seen in the previous section, return to the main platform.


Well done!! The mirror is already cleaned!!
Would you like to play again or quit the game?

*Note: Both options will export your results.

Figure 6.37. Bathroom Game Menu: Final scene.

6.2.7 Game

This part is dedicated to explain what happens when the patients press "Play". This game needs a little bit of skill from the patients because it is like a shooter mode aiming certain points with the spray to clean the mirror. So, to make it easier for them there are some game objects during the process helping to take the right way and play the game without losing orientation.

This game is thought to be played with one hand or two hands, so the appearance of the interactable game objects will be different for each mode.

Tutorial object

This is the first object appearing in the mirror centre to interact with. Its mission is to help patients to point the mirror with the spray particle system at the beginning of the game. That will be helpful when the real game starts because this condition the player to practice the main game action and in this way the game does not have such a frantic beginning.

Once the spray particles collide with the tutorial object, it is necessary to hold three seconds in that position, that will secure patients controllers are already good positioned. Depending on the game mode, it will appear one or two tutorial objects (figure 6.38. and figure 6.39.). To see, the count of the three seconds, two timers will be near the tutorial objects. Those timers use the same Time class as the ones seen on the menu.

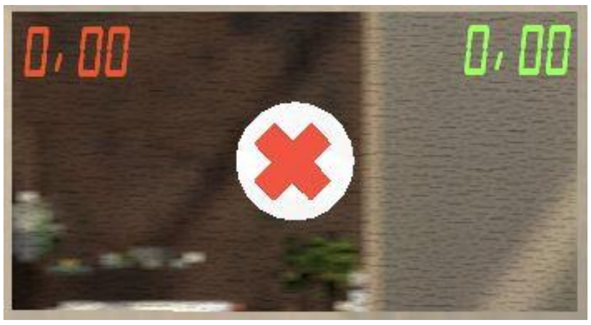


Figure 6.38. Bathroom Game: Tutorial object one hand mode.

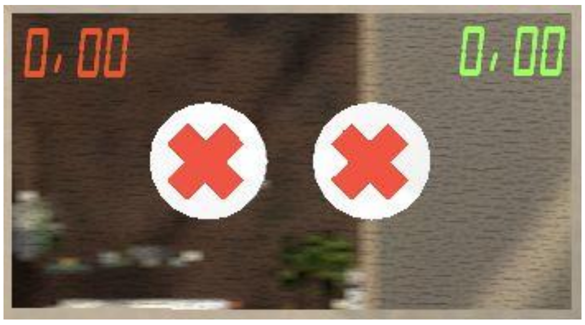


Figure 6.39. Bathroom Game: Tutorial object two hands mode.

Circle shape and Checkpoints

After finishing the tutorial, the mirror will change, and it will appear the real game objective, which is to follow a circular trace passing through some checkpoints with the particle system. When both things appear, there is a checkpoint animation showing the direction of the path that they should follow with the spray. Creating a light sequence like if it was a strip of led.

To mark the start position, it pops up an arrow indicator and a "START" text. That checkpoint has these restrictions if it this does not collide with the particle system first:

- Timers will not start working.
- Spray Raycast will not be available to recollect accuracy data.
- The "End" checkpoint and the rest of them cannot be collided.
- The animation will not stop running.

So, patients will have enough time to start playing and recreate the shape shown over the mirror.

These checkpoints are interactable. When spray's particle system collides with them, they will change their material (from orange to a green material) proving they were touched. There is no need to touch all the checkpoints to end the game, because the "End" collider will be interactable and indicated, like the "Start" checkpoint, when the time starts counting.

The goal of this game is to perform a circular move with the arms, especially a shoulder rotation. So, this could be a little bit difficult, at the beginning, to reach all the checkpoints or finish the trajectory in time. Even though the first results are bad, there is the possibility to check if there is an improvement thanks to the .csv file exported at the end of the rounds. That data export is explained in a later section, there is no need to go in to detail.

As mentioned, depending on the mode chosen, the circular shape and the checkpoints will be changed, respectively. All the modes have a common movement which is regarding the shoulder and the wrist. This will be a rotary movement of the joints, as shown in figure 6.19 and 6.20. These are the three possible game:

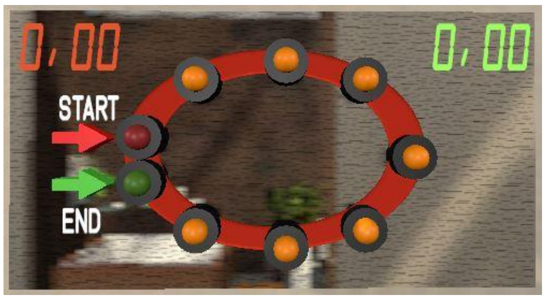


Figure 6.40. Bathroom Game: Checkpoints for the right-hand mode.

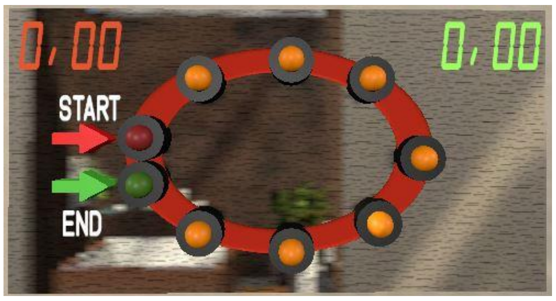


Figure 6.41. Bathroom Game: Checkpoints for the left-hand mode.

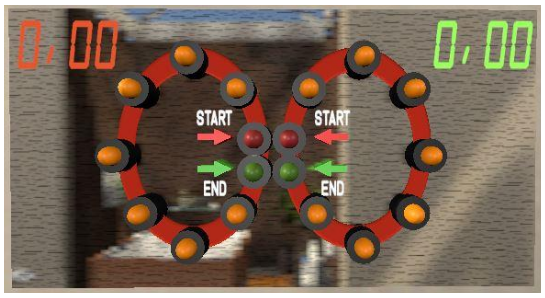


Figure 6.42. Bathroom Game: Checkpoints for both-hands mode.

When the patients finish the game touching the "End" checkpoint, the game will create a clean animation referencing that the mirror is already cleaned. After that, the final scene and will appear asking to the restart the game or quit it. The pointer is enabled again to be able to select one of both options.

Stop button

Finally, this is the last interactive element of the game. Its mission is about to reset the game when the game is already in the play mode. For example, if the user wants to play again with other options but the pointer is already disabled and there is no way to reset that this button will be particularly useful. That will avoid quitting the game, because this button interacts like the checkpoints, with the particle collision.

This object uses the same method (reload the scene) as the "Reset" button, however this one is activated with the particle system and the other with pointer, for different situations.

As you can see in figure 6.43, there is a text over it, like an instruction text, to indicate how is this button used for.

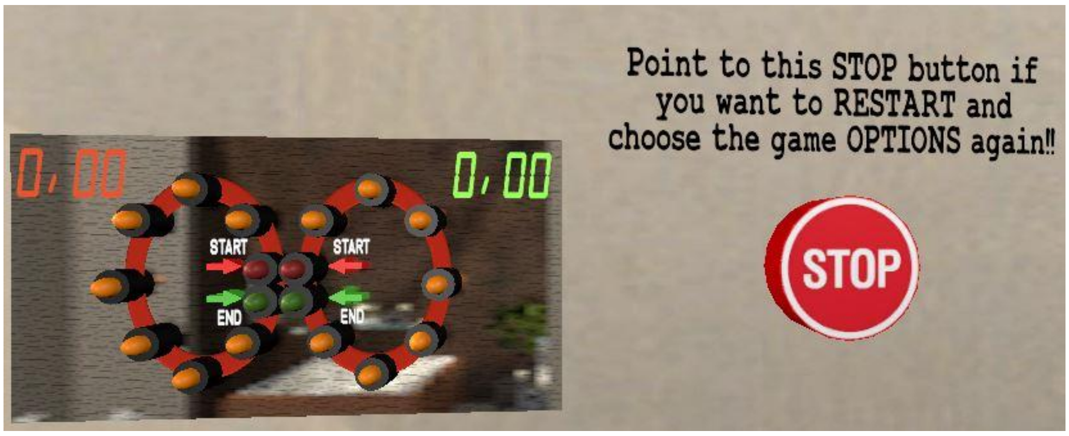

Point to this STOP button if you want to RESTART and choose the game OPTIONS again!!
0,00
0,00
START
START
END
END
STOP

Figure 6.43. Bathroom Game: Stop Button.

6.2.8 Calculate accuracy

Before exporting all the data to a .csv file, it is necessary to calculate how many times the patients have been accurate following the circle shape with the particle systems. First of all, the idea was to create a 3D plane in front of the mirror with the same size and attach a shader on it with no background texture. There is an option for shader components which is the "Unlit texture transparent shader".

What it does is to convert the no background texture to a transparent texture. So, the final result is the circular shape (the coloured part) only shown over the mirror

This technique is used to verify with the spray's Raycast if the pixel has an alpha value greater or equal than zero. So, the calculation process is described by these steps:

- The Raycast is getting pixels all the time and there are two counters, one for the transparent pixels and the other one for the coloured pixels.
- Both counters increase in the same way, with the "DeltaTime ()" method.
- This Raycast script is updating at every frame so, at the end it will get a lot of pixels.
- Collision between "Start" checkpoint and the particle system, the Raycast is enabled and it starts counting the pixels in colour.

When the game is finished ("End" collision), there are two values that will help to calculate the accuracy. To do that, it just has to calculate the coloured pixels percentage with respect to the total pixels recollected.

$$totalPixels = colouredPixels + transparentPixels;$$

(Eq. 6.1)

$$colouredPixels = \frac{colouredPixels}{totalPixels}\ x\ 100\ (\%);$$

(Eq. 6.2)

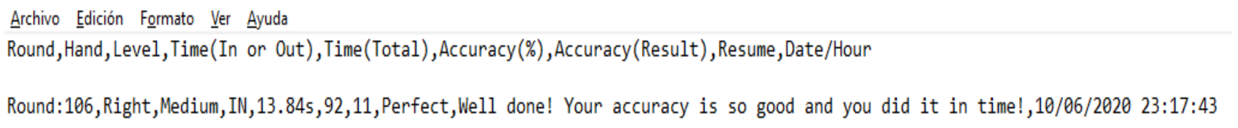
Besides calculating these percentages, the checkpoints that patients could obtain have also been calculated. These two objectives will help to rate the final result of their attempts.

6.2.9 Data exported

To have a good traceability of the patient's progress, there is a script that exports all the data. And this data extracted from this game will be the following:

- **Round:** it calculates the last file row to determine which round the patient is in.
- **Hand selected:** the hand which the patients will do the movement.
- **Level selected:** the level which limits the time to perform the circle shape.
- **Time (In or Out):** this variable returns an "In" string value if the patient can reach the "End" point within the stipulated time, if not it will return an "Out".
- **Total time:** even if the patients don't finish the game in time the chronometer will still counting at least the total time.
- **Accuracy (%):** as mentioned this is the percentage of coloured pixels touched.
- **Accuracy result:** there are three conditions that rate the accuracy depending three percentage ranges. If they get less than 50%, 80%, 100% and an accurate 100% their final result will be "Bad", "Good", "Perfect" and "Excellent" respectively.
- **Summary:** depending on the time, checkpoints, and accuracy, it will be a final summary valuing the relationship between all these values. For example, if the patient get all the checkpoints and has a good accuracy, but he/she did it out of time, the summary will be a comment like "Well done, the patient get good results, however the realization time could be improved".
- **Date/Hour:** there is a method in Time class which gives the date and the hour at this moment. This is good to see the improvement during all the days.

All the data are exported in the same way as the "Clean the Horse" game, explained in the section 6.3. This will be the .csv file appearance after each attempt:



```
Round,Hand,Level,Time(In or Out),Time(Total),Accuracy(%),Accuracy(Result),Resume,Date/Hour

Round:106,Right,Medium,IN,13.84s,92,11,Perfect,Well done! Your accuracy is so good and you did it in time!,10/06/2020 23:17:43
```

Figure 6.43. CSV file exported from the "Cleaning Mirror" game.

6.3 Exporting data

In this section, two scripts responsible of exporting the data collected during the game process are explained. Two scripts were created: “CSVManager” and “MyTools”. Both documents are located at Annex C.

In this project, there is information which requires to be saved in every single attempt. So, “CSVManager” script has two previous steps to follow before and after save all the data. The script sequence is shown in figure 6.44.

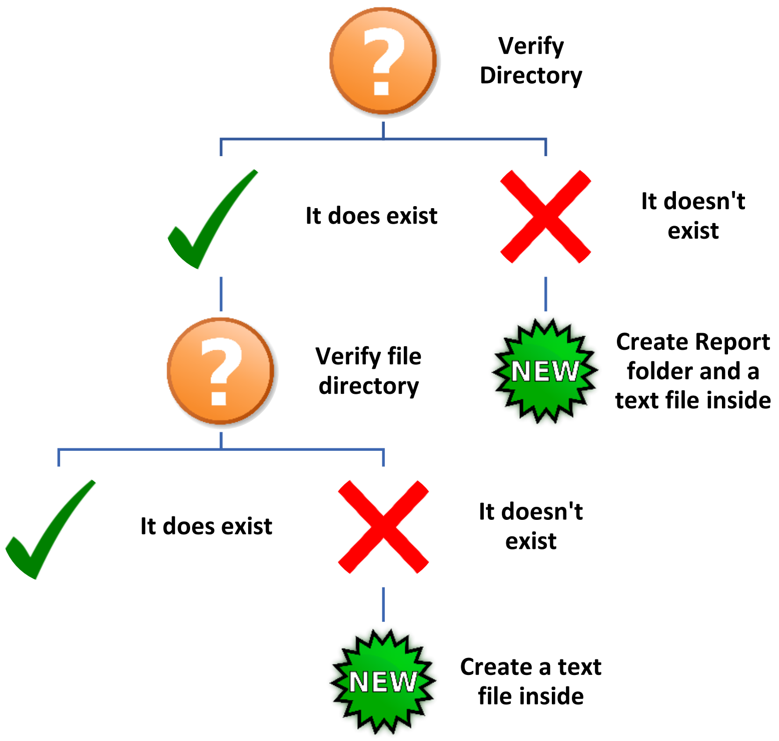


Figure 6.44. Export data: CSVManager script process.

After that, there is a method which appends all the data to the .csv file. It separates the variables with commas, so the document can be opened with the Microsoft Excel program and each variable will be displayed in a different column.

Now, it is necessary to have an input to call this method, so the class “MyTool” has been created to do that. All the game scripts are sent to this script and this will append all data to a .csv file. Each game has two different actions to call that method:

- **Horse cleaning game:** pressing the space bar at the end of the game.
- **Mirror cleaning game**: selecting with the pointer the final scene buttons.

This method (*AppendToReport*) can be found in Annex C.

7. TESTS AND ANALYSES

To evaluate the effectiveness of the games and possible improvements, a survey was conducted on different volunteers after testing these applications developed in this project. These volunteers were the relatives of the project authors because they were the most accessible people during the confinement period. All the volunteers are healthy, so these results are not related with the possible patient’s experiences.
The questions of the survey and the corresponding answers can be seen in Annex A.

Those volunteers who test the “Horse Cleaning” game have given their opinions answering the survey questions. It can be noticed that young people are more eager to try rehabilitation therapy through VR than older people.

On the other hand, the volunteers who try to play “Cleaning Mirror” game were incredibly surprised about the possibilities to take part in those kind of therapies in a virtual world. They were amazed about the visual graphics and how the character perform the same movements that they were doing.

Moreover, some of the volunteers that have undergone any rehabilitation therapy think that they were bored or neither bored nor entertained. No one has answered that they were entertained.

After taking both tests, the results of all volunteers were good because they do not have any pathology, and all of them could finish the exercises in the time proposed. Furthermore, the volunteers have valued the experience with good punctuation, and they agree that VR can improve experiences in rehabilitation sessions.

At the end of the test process, these volunteers gave their opinions about games design. They agreed that the instructions of the games were clear, and the game environments were relaxing. Except for one person who mentioned that the noise of the horse was a little bit annoying because she does not like animals too much.

Then, the volunteers also commented possible improvements for the games. In the “Clean the Horse” case, one volunteer mentioned that the arm of the player could be changed for a more realistic one. Another one commented that the camera view was not comfortable since it was moving when following the same mouse trajectory.

Last but not least, another volunteer mentioned that the mouse was moving very slowly. The sensitivity of the mouse in this game is slow because if we increase its speed the patient may find it harder to follow the trajectory and it can make them a little bit dizzy (specially for stroke subjects, the target group for this game).

In the "Clean Mirror" game, the volunteers liked the menu interactions and how they look with a robot appearance inside the game. Approximately, they had the same opinions about these kind of therapy VR applications. They mentioned that this is so innovative, motivational, and basically a different way that attracts more the patients to continue rehabilitations. About their experiences, they feel comfortable inside the VR world and enjoyed the game.

After evaluating all their valuations, here there are all the data recollected, graphically, from the volunteers of each game.

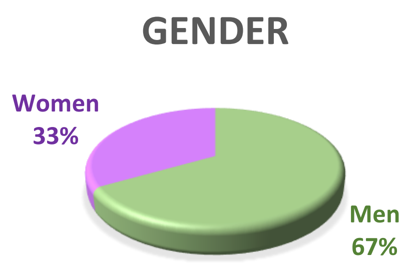


Figure 7.1. "Clean the Horse" game: Volunteer gender graph.

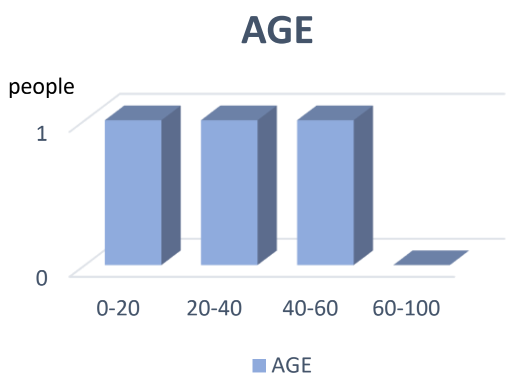


Figure 7.2. "Clean the Horse" game: Volunteer age graph.

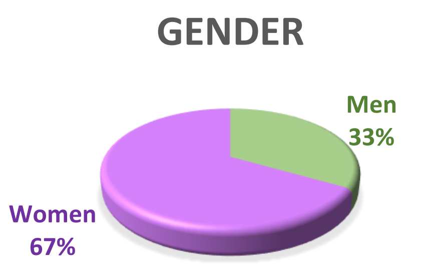


Figure 7.3. "Clean the Mirror" game: Volunteer gender graph.

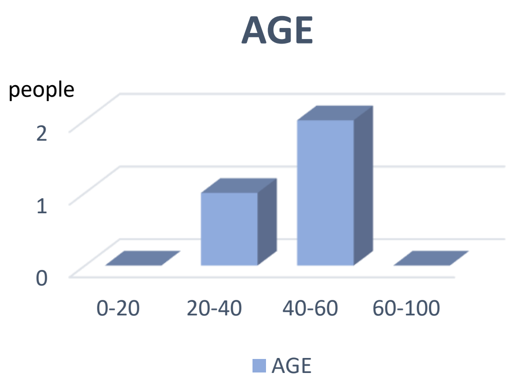


Figure 7.4. "Clean the Mirror" game: Volunteer age graph.

		QUESTIONS BEFORE TESTING THE GAMES						
		Gender	**Age**	**Question 1**	**Question 1.1**	**Question 1.2**	**Question 1.3**	**Question 2**
Clean the Horse Game	**Volunteer 1**	Male	18	Yes	Neither boring or entretaining	As soon as he asked	Yes	Yes. Because I find interesting that new technologies have influence in the health sector.
	Volunteer 2	Female	52	Yes	Boring	As soon as she asked	Yes	Yes. It doesn't catch my eye, but if it has to go well then yes.
	Volunteer 3	Male	21	No	-	-	-	Yes. Because it can be more useful, didactic and entertaining.
Clean the Mirror Game	**Volunteer 1**	Female	56	Yes	Boring	She had to wait a little bit due to the demand	Yes	Yes. Because it is an enjoyable way to face these situations.
	Volunteer 2	Female	26	No	-	-	-	Yes. All the new therapeutic methods related on new technologies are more atractrive and motivational to the patients.
	Volunteer 3	Male	57	Yes	Neither boring or entretaining	As soon as he asked	Yes	Yes. Because sometimes people don't feel confortable doing some conventional exercices and VR apps could give more freedom to improve the mobility.

Table 7.5. Questions before taking the test.

		QUESTIONS AFTER TESTING THE GAMES						
		Question 1	**Question 2**	**Question 3**	**Question 4**	**Question 5**	**Question 6**
Clean the Horse Game	**Volunteer 1**	Easy	4	Yes, because the patient can work harder and at the same time be entertained.	Yes, it's relaxing.	Yes, the instruction are well indicated in the menu scene.	The graphics: The gray arm does not look like reality.
	Volunteer 2	Normal difficulty	5	Yes, but depending on the type of rehabilitation. If it's only a massage I don't think so.	It's relaxing, but in my case, the noise of the horse annoyed me a little bit because I don't like animals too much.	Yes, the instruction are well indicated in the menu scene.	It is not comfortable that the camera is moving when the mouse is moved.
	Volunteer 3	Difficult	4	Yes, because it is more sensory; you pay more attention when you do the exercise.	Yes, it's relaxing.	Yes, the instruction are well indicated in the menu scene.	Mouse sensitivity. It was very slow.
Clean the Mirror Game	**Volunteer 1**	Normal difficulty	5	Yes, that make get more focused.	Yes, the environtment aesthetic is suitable	Yes, the instructions are well located in the middle of mirror and they are easy to read.	The spray's particles was not to much similar a real cleaning liquid.
	Volunteer 2	Easy	5	Yes, because it is funny and make time passes faster.	Yes, it's so relaxing.	Yes, the text colour contrasts perfectly with the mirror texture.	The sensibility is a little bit high.
	Volunteer 3	Normal difficulty	5	Yes, but the applications have to be well design to not cause dizziness.	Yes, there's no objections about the environment.	Yes, pleasant text font and good size.	The last mirror cleaned effect could have a sound too.

Table 7.6. Questions after testing the games the test.

7.1 Results after tests

		RESULTS OF THE EXERCISE			
		Shape	**Difficulty**	**Accuracy (%)**	**Time (minutes)**
Clean the Horse Game	**Volunteer 1**	Triangle	Slow	74	0:59
	Volunteer 2	Square	Slow	87	1:31
	Volunteer 3	Square	Fast	92	1:29

Table 7.7. “Clean the Horse” game: Results of the exercise.

		RESULTS OF THE EXERCISE				
		Hand	**Difficulty**	**Accuracy (%)**	**Time (IN or OUT)**	**Time (minutes)**
Clean the Mirror Game	**Volunteer 1**	Both	Medium	67	IN	0:23
	Volunteer 2	Left	Fast	88	IN	0:18
	Volunteer 3	Right	Fast	71	IN	0:15

Table 7.8. “Clean the Mirror” game: Results of the exercise.

8. ENVIRONMENTAL IMPACT

In this project, the impact produced to the environment has been studied. It is proved that the environment can be slightly affected by two different ways: electromagnetically [64] and due to their chemical composition too. Since the scientific studies determine that using electronic devices is harmful, new laws have been applied to their industrialization.

In June 2011, the European Union stablished a chemical substances regulation, Directive - 2011/65/UE, for the electronic device creation called RoHS (Restriction of Hazardous Substances). There forbade the use of these six noxious materials: Lead, Mercury, Cadmium, Chromium VI, PBB and PBDE. Last two elements are used in some encapsulated plastics. Nowadays, electronic devices must have a RoHS label that complies with the law (figure 8.1).

On the other hand, there is also another current law that dictates the way to dispose electronic residues. This is the Directive - 2012/19/UE and basically promotes to not throw electronic devices to the conventional waste containers. This law promotes a recycling process due to their bad decomposition. They usually have a label symbolizing that action (figure 8.2).

Figure 8.1. RoHS Certificate label [https://www.eurocircuits.com/].

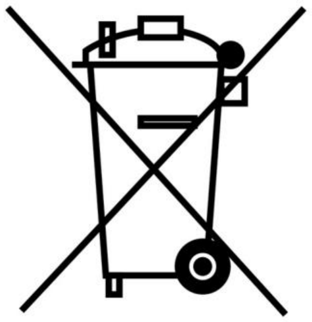


Figure 8.2. Electronic device label to avoid toxic waste [http://eco3e.eu/].

As commented, there is also one more affection by electronic devices which the generation of high-frequency radiation. This noise radiation can affect the environment and also people, causing headache, fatigue, and so on.

9. SOCIAL IMPACT

Projects that have a social impact are the ones that in one way or another are focused on improving society or they are seeked to be more sustainable for the environment. Our project has a social impact because it improves the rehabilitation process, the health status and quality of life of patients who suffered a stroke. Their quality of life is improved because the VR games developed in this project have a pleasing entertaining environment and it is possible to know the results of the therapy when the patient finishes the exercise.

On the other hand, with the conventional therapies, the patient loses motivation or interest due to the exercises become too repetitive and it can also be demotivating not seeing the progress done.

Figure 9.1. Impact of the health sector.
[https://www.openaccessgovernment.org/]

Nowadays, the Impact Economy is increasing more and more. Impact Economy is a new economic model where the main purpose of companies, investors and organizations is no longer just to maximize their economic profitability but also their social or environmental impact. Traditionally, the health innovation came only from the professionals in public or private health sector.

However, nowadays more and more entrepreneurs or even companies that are not related to this sector are innovating in this area.
This helps to connect different technologies and it also helps to have a more global vision of the problem that is trying to solve. In this way, technology not only focuses on the patient, but understands the user as something more global (families, caregivers, physiotherapists, etc.). In our case, the creation of these two games not only helps stroke patients to rehabilitate, but also helps physiotherapists by reducing their amount of work and avoiding consulting rooms saturated. And it also improves the collection of data for subsequent analysis.

Furthermore, another indicator to know that our project has social impact is that it simulates the daily living skills. Physiotherapists really like this type of applications because they simulate daily living activities. In this project, it has been developed one game that follows this concept because patients have to follow trajectories pretending, they are cleaning the mirror of the bathroom. The other game developed is about cleaning a horse using a hose. This second game can also be applicable to a real-life movement, but it is not usual to clean a horse in the everyday tasks. So, this second game is more for an entertaining purpose while the patient improves the mobility of their upper limbs after suffering from a stroke.

10. CONCLUSIONS

In this project it has been achieved the implementation of virtual reality in the rehabilitation of patients who have suffered a stroke in order to improve the motor function of their upper limbs. Two games have been created in order to fulfill this goal in collaboration with the *Associació de Diversitat Funcional d'Osona* (ADFO). Moreover, some healthy volunteers have tested these games to provide feedback of the virtual reality experience. With the data extracted from the tests carried out by these volunteers, the functionality of the two games has been evaluated and it has also been collected the possible improvements of each game that they commented in the survey. The numerical results of all the volunteers were satisfactory as expected, because they are healthy. In general, they have presented a good accuracy of the exercise and have performed it before the time runs out.

Working with an innovative technology such as virtual reality brings satisfaction of the work done. Moreover, it has allowed learning how to code in Unity3D and the C# language which will be useful in future projects or applications.

The project has improved not only the rehabilitation sessions for the patient but also for the physiotherapist. The physiotherapist will have an excel document with the results of each patient. So it is possible to assess how much progress has the patient made from one therapy to another. This will also help the collection of data for subsequent analysis.

In addition, we had some limitations during the period available to create this project. The arrival of a global pandemic due to COVID-19 has forced people to work from home. Therefore, it has been impossible to use the necessary material (the headset and the controllers) to carry out a virtual reality application in full conditions. This meant a pause in the project development but, thanks to the contribution of ADFO, the methodology of creating the games was reoriented and it was able to move forward.

Despite these drawbacks, the motivation that has been in the project has allowed to keep working on it. Most of this motivation is due to the social impact the application has because thanks to this product it is improved the quality of life of the patients, their families and caregivers, and the medical staff that surrounds them every day.

11. FUTURE LINES

In this section, there are some possible improvements for these applications for the following project students. Both games could be much better if somebody can use the SteamVR equipment and test them properly.

11.1 Horse cleaning game

- If it is desired, the game "Clean the Horse" can be integrated as a virtual reality application. For now, it is created with the idea that the patient can interact in the game through the computer mouse while sitting in front of the screen. The current goal is that the patient begins to acquire upper limb mobility while doing the exercise of dragging the mouse from side to side. However, the game could become to a virtual reality application where the goal would be that the patient performs the movements with the arm extended horizontally in front of him, following the different shapes that appear on the horse body.
- Another possible improvement would be finding an arm and a hose whose graphics are better created and more realistic that the ones that the game currently has.
- Prevent the camera from following the movements of the mouse.
- Link the data exported to the main database in MySQL.

11.2 Mirror Cleaning game

- First of all, test the game with the SteamVR setup and fix all the possible issues if necessary.
- Create different shapes to trail with the particle system, to make other kind of exercises.
- Make the particle system more realistic.
- Link the data exported to the main database in MySQL.

12. ECONOMIC ANALYSIS

In this section there is the economic analysis which includes the budget split and grouped into the following chapters: hardware costs, software costs, staff costs, and electricity and consumption costs.

12.1 Hardware costs

Calculating the linear depreciation following the equation 12.1.

$$Depreciation\ (€/months) = \frac{initial\ value\ (€) - \ residual\ value\ (€)}{product\ lifetime\ (months)}$$

(Eq. 12.1)

Where:

- **Depreciation:** periodic decrease in the value of a material or immaterial good.
- **Initial value:** the acquisition cost.
- **Residual value:** estimated value of the device when its useful life has already ended.
- **Product lifetime:** the time interval from when a product is sold to when it is discarded.

In this way, the monetary value that the device in question loses every month is known.

Then, it has been calculated the real cost of each product. It is also considered that the real cost is the cost of each device during the time it has been working (usage time). It is calculated following the equation 12.2.

$$Real\ cost\ (€) = initial\ value\ (€) \cdot \frac{usage\ time\ (months)}{product\ lifetime\ (months)}$$

(Eq. 12.2)

The table 12.1 shows the real cost of each product. Thereby, the total hardware costs for this project are 963.98 €.

Product	Initial value (€)	Residual value (€)	Product lifetime (months)	Monthly depreciation (€)	Monthly depreciation (%)	Usage time (months)	Real cost (€)
Laboratory computer set	1500	300	48	25,00	1,67	1	31,25
Macboook pro 13'	1499	600	48	18,73	1,25	4	320,14
Computer HP AllinOne	899,99	200	48	14,58	1,62	4	246,86
Mobile phone: Xiaomi Redmi 6	159	45	36	3,17	1,99	2,5	125,53
Mobile phone: Xiaomi Redmi 8	139	40	36	2,75	1,98	2,5	126,36
Mobile phone: Samsung A3	143,26	30	36	3,15	2,20	2,5	113,84
Total hardware cost	-	-	-	-	-	-	**963,98**

Table 12.1. Hardware costs.

12.2 Software costs

The software costs are shown in the table below. In this project, there was no need to pay for any of the programs used as they are open source programs.

The cost of the software also took into account that one of the members of the project paid for the online *Master in Video Game Programming with Unity® 2020 and C#* and the two members took *Unity VR Fundamentals* free course provided by our tutor.

Product	Type of license	Price (€)
Unity 3D	Unity Personal	0
Visual Studio	Open source	0
Steam VR	Open source	0
Master in Video Game Programming with Unity® 2020 and C#	Unity Personal	12,99
Unity VR Fundamentals	Free downloaded files	0
Total software cost	-	**12,99**

Table 12.2. Software costs.

12.3 Staff costs

For calculating the staff costs, it has been considered the salary of an electronic engineer and a biomedical engineer as shown in the table 12.3. A 30% of gross salary is what is paid to National Insurance.

Staff	Hours worked	Cost per hour worked (€/hour)	National Insurance	Final cost (€)
Biomedical engineer	600	30	5400	23400
Electronic engineer	600	30	5400	23400
Total staff cost	-	-		**46800**

Table 12.3. Staff costs.

12.4 Electricity consumption costs

In order to calculate the electricity consumption costs, it was searched the power (in watts) of the electronic components used to carry out the project.

Product	**Power (W)**
Laboratory computer	240
Laboratory mouse	2
Laboratory screen	45
Macboook pro 13'	61
Computer HP AllinOne	150
Mobile phone: Xiaomi Redmi 6	10
Mobile phone: Xiaomi Redmi 8	18
Mobile phone: Samsung A3	10
Total	**536**

Table 12.4. Power (in watts) of each component used.

In 2020, the fixed price of a general consumption rate is 0.1527 €/KWh, which is why the cost of electricity consumption is calculated as follows:

$$Electricity\ consumption\ costs = price\left(\frac{€}{kWh}\right)\cdot\ nº\ working\ hours \cdot nº\ total\ kW$$

(Eq. 12.3)

$$Electricity\ consumption\ costs = 0.1527\left(\frac{€}{kWh}\right)\cdot 600h \cdot 0.536kW = 49.11€$$

(Eq. 12.4)

Therefore, the total electricity consumption cost for this project is 49.11€.

12.5 Establishment costs

The indirect cost of the establishments where the authors of this project have been working these months must also be considered. It is considered to pay 300 €/month for both homes used. Therefore, the establishment costs for four months are 2,400 €.

12.6 Expense summary

Lastly, after analyzing the 5 chapters (hardware costs, software costs, staff costs, electricity consumption costs and establishment costs), a summary table of all the costs analyzed is presented to obtain the final cost of the project (table 12.5). It has also been considered a 10% of Industrial Profit. Therefore, the final cost of the project is 45,203.47 €.

Cost type	Cost
Hardware costs	963,98
Software costs	12,99
Staff costs	46800
Electricity consumption costs	49,11
Establishment costs	2400
Industrial Profit	5022,61
Total costs	**45203,47**

Table 12.5. Final cost of the project.

REFERENCES

State of the art

[1]. *Binocular Vision and Stereopsis - Distinguished Research Professor Emeritus and Founder of the Centre for Vision Research Ian P Howard, Ian P. Howard, Brian J. Rogers - Google Libros*. (n.d.). Retrieved June 25, 2020, from https://books.google.es/books?hl=es&lr=&id=I8vqITdETe0C&oi=fnd&pg=PA1&dq=vr+stereopsis&ots=JijKV_JoNz&sig=Lc31v6vQHnjwGW-kIijs0WZwKlE.#v=onepage&q&f=false

[2]. *Communication in the Age of Virtual Reality - Google Libros*. (n.d.). Retrieved June 25, 2020, from https://books.google.es/books?hl=es&lr=&id=MzaMSbzcz6UC&oi=fnd&pg=PA3&dq=virtual+reality+morton+heilig&ots=Vrn8YUhRIQ&sig=6peHBqx4urP6jWtwnRtABCWD68c#v=onepage&q=virtual reality morton heilig&f=false

[3]. *Understanding Virtual Reality: Interface, Application, and Design - William R. Sherman, Alan B. Craig - Google Libros*. (n.d.). Retrieved June 25, 2020, from https://books.google.es/books?hl=es&lr=&id=D-OcBAAAQBAJ&oi=fnd&pg=PP1&dq=virtual+reality+Comeau+and+Bryan&ots=QR3hbbcX-N&sig=rqR60fn5rj1WhT2wNVSfsx8Lw_E#v=onepage&q=virtual reality Comeau and Bryan&f=false

[4]. Mazuryk, T., Mazuryk, T., & Gervautz, M. (n.d.). *Virtual Reality - History, Applications, Technology and Future*. Retrieved June 25, 2020, from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7849

[5]. Lee, S. H. (Mark), Sergueeva, K., Catangui, M., & Kandaurova, M. (2017). Assessing Google Cardboard virtual reality as a content delivery system in business classrooms. *Journal of Education for Business*, *92*(4), 153–160. https://doi.org/10.1080/08832323.2017.1308308

[6]. *History of VR - Timeline of Events and Tech Development*. (n.d.). Retrieved June 25, 2020, from https://virtualspeech.com/blog/history-of-vr

[7]. ¿Qué es un ictus? | Tipos de ictus | Síntomas del Ictus. (n.d.). Retrieved June 23, 2020, from https://rithmi.com/que-es-un-ictus/

[8]. Ictus-ACV sintomas y tratamiento. Clínica Universidad Navarra. (n.d.). Retrieved June 23, 2020, from https://www.cun.es/enfermedades-tratamientos/enfermedades/ictus

[9]. Ictus Cerebral: Tratamientos, síntomas, causas e información. (n.d.). Retrieved June 23, 2020, from https://cuidateplus.marca.com/enfermedades/neurologicas/ictus.html

[10]. ¿Qué es un ictus? | Enfermedades neurológicas | Infosalus. (n.d.). Retrieved June 23, 2020, from https://www.infosalus.com/enfermedades/neurologia/ictus/que-es-ictus-29.html

[11]. Código Ictus - Federación Española del Ictus. (n.d.). Retrieved June 23, 2020, from https://ictusfederacion.es/infoictus/codigo-ictus/

[12]. Què és un ictus? | PortalCLÍNIC. (n.d.). Retrieved June 23, 2020, from https://www.clinicbarcelona.org/ca/asistencia/malalties/ictus/definicio

[13]. *Superar l'ictus*. (n.d.). | Departament de Salut Generalitat de Catalunya.

[14]. ¿Cómo conseguir una rehabilitación efectiva tras un ictus? (n.d.). Retrieved June 23, 2020, from https://muysaludable.sanitas.es/salud/prevencion/conseguir-una-rehabilitacion-efectiva-tras-ictus/

[15]. Tipos de rehabilitación tras sufrir un ictus | Sanitas Mayores. (n.d.). Retrieved June 23, 2020, from https://www.sanitas.es/sanitasresidencial/residencias-mayores/preguntas-frecuentes/especialidades/rehabilitacion-para-ictus

[16]. Rehabilitación - Federación Española de Ictus. (n.d.). Retrieved June 23, 2020, from https://ictusfederacion.es/infoictus/rehabilitacion/

[17]. Viñas-Diz, S., & Sobrido-Prieto, M. (2016, May 1). Realidad virtual con fines terapéuticos en pacientes con ictus: Revisión sistemática. *Neurologia*. Spanish Society of Neurology. https://doi.org/10.1016/j.nrl.2015.06.012

[18]. Efeito de um programa de reabilitação usando realidade virtual na função do membro superior em pacientes com acidente vascular cerebral. (n.d.). Retrieved June 23, 2020, from https://repositorio.unesp.br/handle/11449/149919

[19]. Mireia, A. :, Rosell, J., Pradell, M., Tutora, G., & Navarro, E. M. (2015). *Rehabilitation of the upper limb in chronic stroke patients with Virtual Reality: A Systematic Review*.

[20]. *Autora: Carolina Colomer Font*. (n.d.). | Rehabilitación del miembro superior parético en pacientes con ictus: Eficacia del empleo de Entornos virtuales, Soportes Robóticos y Retroalimentación Visual Con Espejo.

[21]. Muñoz Boje, R., & Calvo-Muñoz, I. (2018, January 1). Effects of virtual reality therapy for the upper limb in stroke patients: a systematic review. *Rehabilitacion*. Ediciones Doyma, S.L. https://doi.org/10.1016/j.rh.2017.09.001

[22]. Montalbán, M. A., & Arrogante, O. (2020). Rehabilitation through virtual reality therapy after a stroke: A literature review. *Revista Científica de La Sociedad de Enfermería Neurológica (English Ed.)*. https://doi.org/10.1016/j.sedeng.2020.01.001

[23]. Press, E. (n.d.). Crean el "Netflix" de la rehabilitación de pacientes con ictus. Retrieved from https://www.infosalus.com/asistencia/noticia-crean-netflix-rehabilitacion-pacientes-ictus-20190916164658.html

[24]. Rehabilitació virtual per superar l'ictus | Vall d'Hebron Barcelona Hospital Campus. (n.d.). Retrieved June 23, 2020, from https://www.vallhebron.com/ca/el-campus/projectes-estrategics/rehabilitacio-virtual-superar-lictus

[25]. Lo *et al*. "Prospective clinical study of rehabilitation interventions with multisensory interactive training in patients with cerebral infarction: study protocol for a randomised controlled trial". *Trials*, vol. 18, no 173, 2017.

[26]. M. Yates, A. Kelemen and C. Sik Lanyi, "Virtual reality gaming in the rehabilitation of the upper extremities post-stroke". *Brain Injury*, vol 30, no7, pp. 855-863, 2016.

[27]. Sheehy et al. "Home-based virtual reality training after discharge from hospital-based stroke rehabilitation: a parallel randomized feasibility trial. *Trials*. Vol. 20, no 333. 2019.

Functional and non-functional requirements

[28]. Requerimientos Funcionales y No funcionales - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=tF88eNhNSb4

[29]. REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=Np_LYwWjp6E

[30]. Requerimientos Funcionales y No Funcionales, ejemplos y tips. (n.d.). Retrieved June 23, 2020, from https://medium.com/@requeridosblog/requerimientos-funcionales-y-no-funcionales-ejemplos-y-tips-aa31cb59b22a

[31]. 3 Técnicas para Identificar Requisitos Funcionales y No Funcionales - Metodología Gestión de Requerimientos. (n.d.). Retrieved June 23, 2020, from https://sites.google.com/site/metodologiareq/capitulo-ii/tecnicas-para-identificar-requisitos-funcionales-y-no-funcionales

[32]. Tipos de requisitos: Funcional vs. No Funcional | Tecnología y Synergix. (n.d.). Retrieved June 23, 2020, from https://synergix.wordpress.com/2008/07/07/requisito-funcional-y-no-funcional/

[33]. Requerimientos no funcionales: Ejemplos - La Oficina de Proyectos de Informática. (n.d.). Retrieved June 23, 2020, from http://www.pmoinformatica.com/2015/05/requerimientos-no-funcionales-ejemplos.html

[34]. *Especificación de requerimientos Especificación de requerimientos Diseño de bases de datos Diseño de bases de datos*. (n.d.).

[35]. Requerimientos Funcionales y No Funcionales – Porqueria. (n.d.). Retrieved June 23, 2020, from https://ingenieriadesoftwareutmachala.wordpress.com/2017/01/20/requerimientos-funcionales-y-no-funcionales/

Applications design

[36]. Unity - Manual: Canvas. (n.d.). Retrieved June 23, 2020, from https://docs.unity3d.com/2020.1/Documentation/Manual/UICanvas.html

[37]. Nature Starter Kit 2 | 3D Environments | Unity Asset Store. (n.d.). Retrieved June 23, 2020, from https://assetstore.unity.com/packages/3d/environments/nature-starter-kit-2-52977

[38]. Fantasy Forest Environment - Free Demo | 3D Fantasy | Unity Asset Store. (n.d.). Retrieved June 23, 2020, from https://assetstore.unity.com/packages/3d/environments/fantasy/fantasy-forest-environment-free-demo-35361

[39]. Présentation d'assets #2: Nature Starter Kit 2 [Unity3D] - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=syCdyC0_9m0

[40]. Rehabilitación de la hemiplejia tras el ictus: el miembro superior severamente afectado - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=2cG2AayaBjQ

[41]. Detecting Collisions (OnCollisionEnter) - Unity Official Tutorials - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=QRp4V1JTZnM

[42]. Tutorial - Export data from Unity to .csv (which opens with Excel) - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=JR4EMeJo55A

[43]. Controlling Particles Via Script - Introduction and Session Goals [1/11] Live 2017/2/8 - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=JRa2g3vgzBo&list=PLX2vGYjWbI0QJJfR-jSqxonYuCHrUhAvN

[44]. gameobjects don't add to count when destroyed - Unity Answers. (n.d.). Retrieved June 23, 2020, from https://answers.unity.com/questions/1578559/gameobjects-dont-add-to-count-when-destroyed.html

[45]. Unity - Scripting API: Collider.enabled. (n.d.). Retrieved June 23, 2020, from https://docs.unity3d.com/ScriptReference/Collider-enabled.html

[46]. activar o desactivar collider, Unity - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=O6x0TmSPz8g

[47]. reset destroyed collider after game restart - Unity Answers. (n.d.). Retrieved June 23, 2020, from https://answers.unity.com/questions/1522309/reset-destroyed-collider-after-game-restart.html

[48]. Unity - Scripting API: GameObject.FindGameObjectsWithTag. (n.d.). Retrieved June 23, 2020, from https://docs.unity3d.com/ScriptReference/GameObject.FindGameObjectsWithTag.html

[49]. Ep. 2 : Clickable Button to change scene - Unity 5.5 tutorial for beginner - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=7KR5IKi8m8g

[50]. Unity - Scripting API: GameObject. (n.d.). Retrieved June 23, 2020, from https://docs.unity3d.com/ScriptReference/GameObject.html

[51]. c# - Accessing an object from another scene in Unity - Game Development Stack Exchange. (n.d.). Retrieved June 23, 2020, from https://gamedev.stackexchange.com/questions/128129/accessing-an-object-from-another-scene-in-unity

[52]. String.IsNullOrEmpty(String) Método (System) | Microsoft Docs. (n.d.). Retrieved June 23, 2020, from https://docs.microsoft.com/es-es/dotnet/api/system.string.isnullorempty?view=netcore-3.1

[53]. How to create a Timer [Tutorial][C#] - Unity 3d - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=x-C95TuQtf0

[54]. How to create a linear timer for your Unity game? Simple Unity 2D tutorial. - YouTube. (n.d.). Retrieved June 23, 2020, from https://www.youtube.com/watch?v=bcvLM_riVuw

[55]. Unity - Scripting API: Application.persistentDataPath. (n.d.). Retrieved June 23, 2020, from https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html

[56]. VRidge - Play PC VR on your Cardboard. (n.d.). Retrieved June 23, 2020, from https://riftcat.com/vridge

[57]. SteamVR en Steam. (n.d.). Retrieved June 23, 2020, from https://store.steampowered.com/app/250820/SteamVR/

[58]. Phone-based VR is officially over - The Verge. (n.d.). Retrieved June 23, 2020, from https://www.theverge.com/2019/10/16/20915791/google-daydream-samsung-oculus-gear-vr-mobile-vr-platforms-dead

Economic analysis

[59]. Cómo calcular la depreciación de un ordenador | Blog de Rentoox. (n.d.). Retrieved June 23, 2020, from https://www.rentoox.com/blog/como-calcular-depreciacion-ordenador/

[60].HP Pavilion All-in-One - 24-xa0035ns Especificaciones del producto | Soporte al cliente de HP®. (n.d.). Retrieved June 23, 2020, from https://support.hp.com/es-es/document/c06473541

[61]. Los nuevos MacBook Pro de 13 pulgadas no tienen carga rápida. (n.d.). Retrieved June 23, 2020, from https://lamanzanamordida.net/noticias/one-more-thing/macbook-pro-13-2020-no-tienen-carga-rapida/

[62]. Redmi 8 丨 Xiaomi España 丨 Mi.com/es - Xiaomi España. (n.d.). Retrieved June 23, 2020, from https://www.mi.com/es/redmi-8/specs/

[63]. Precio del kwh en España: Comparativa de Endesa, Iberdrola y otras. (n.d.). Retrieved June 23, 2020, from https://tarifaluzhora.es/info/precio-kwh

Environment Impact

[64]. Redl, R. (2001). Electromagnetic Environmental Impact of Power Electronics Equipment. *Proceedings of the IEEE*, *89*(6), 926–938. https://doi.org/10.1109/5.931490

IMAGE REFERENCES

Figure 1.1: frontiers-virtual-reality-medical-application-e1571208350285.jpg (1000×562). (n.d.). Retrieved June 23, 2020, from https://frontiersinblog.files.wordpress.com/2019/10/frontiers-virtual-reality-medical-application-e1571208350285.jpg

Figure 3.1: *Charles Wheatstone | artefactual*. (n.d.). Retrieved June 25, 2020, from https://artefactual.co.uk/tag/charles-wheatstone/

Figure 3.2: *Sensorama virtual reality system | Download Scientific Diagram*. (n.d.). Retrieved June 25, 2020, from https://www.researchgate.net/figure/Sensorama-virtual-reality-system_fig1_220474975

Figure 3.3: *Espada de Damocles | IDIS*. (n.d.). Retrieved June 25, 2020, from https://proyectoidis.org/espada-de-damocles/

Figure 3.4: *Réalité Augmentée timeline | Timetoast timelines*. (n.d.). Retrieved June 25, 2020, from https://www.timetoast.com/timelines/realite-augmentee

Figure 3.5: *NASA's virtual reality: The most advanced way to train astronauts*. (n.d.). Retrieved June 25, 2020, from https://www.pulseheadlines.com/nasas-virtual-reality-the-most-advanced-way-to-train-astronauts/66357/

Figure 3.6: *Valve añade compatibilidad en Linux con Steam VR | Geektopia*. (n.d.). Retrieved June 25, 2020, from https://www.geektopia.es/es/technology/2017/02/22/noticias/valve-anade-compatibilidad-en-linux-con-steam-vr.html

Figure 3.7: *Virtual reality, the future of sport: NBA live streams as the NFL and Premier League play catch up - Page 3 of 5 - Computer Business Review*. (n.d.). Retrieved June 25, 2020, from https://www.cbronline.com/internet-of-things/virtual-reality-future-sport-nba-live-streams-nfl-premier-league-play-catch/3/

Figure 3.8: *VR , AR, & MR in Education - Awecademy - Medium*. (n.d.). Retrieved June 25, 2020, from https://medium.com/awecademy/vr-ar-mr-in-education-e789bfe5b869

Figure 3.9: 8206407922907b50c887bc634c618fc5a0268c3d.jpeg (1024×512). (n.d.). Retrieved June 23, 2020, from https://www.clinicbarcelona.org/uploads/media/default/0001/08/8206407922907b50c887bc634c618fc5a0268c3d.jpeg

Figure 3.10: 0e170aeef9897c5136b748c79403cadeec68a589.jpeg (480×328). (n.d.). Retrieved June 23, 2020, from https://www.clinicbarcelona.org/uploads/media/default/0001/09/0e170aeef9897c5136b748c79403cadeec68a589.jpeg

Figure 3.11: UPMCEast_CTscan.jpg (3456×2304). (n.d.). Retrieved June 23, 2020, from https://upload.wikimedia.org/wikipedia/commons/2/27/UPMCEast_CTscan.jpg

Figure 3.12: NEW_MRI_Machine_2018.jpg (1680×1200). (n.d.). Retrieved June 23, 2020, from https://www.sah.org.au/assets/images/Radiology/NEW_MRI_Machine_2018.jpg

Figures 3.13: *Autora: Carolina Colomer Font*. (n.d.). | Rehabilitación del miembro superior parético en pacientes con ictus: Eficacia del empleo de Entornos virtuales, Soportes Robóticos y Retroalimentación Visual Con Espejo.

Figures 3.14: Press, E. (n.d.). Crean el "Netflix" de la rehabilitación de pacientes con ictus. Retrieved from https://www.infosalus.com/asistencia/noticia-crean-netflix-rehabilitacion-pacientes-ictus-20190916164658.html

Figures 3.15: rgssystem_552.jpg (552×310). (n.d.). Retrieved June 23, 2020, from https://www.vallhebron.com/sites/default/files/styles/fitxa/public/header/rgssystem_552.jpg?itok=o1SwW8_l

Figure 5.2: *Google Cardboard? Google Cardboard VR Headset - WhatVR*. (n.d.). Retrieved June 25, 2020, from https://www.whatvr.co/guides/google-cardboard/

Figure 5.3: *How to play any Game with VRidge – RiftCat - Help Center*. (n.d.). Retrieved June 25, 2020, from https://support.riftcat.com/hc/en-us/articles/360010299479-How-to-play-any-Game-with-VRidge

Figure 5.4: *New Riftcat VRidge 2.3 features: play Steam VR games using your phone as a controller! - The Ghost Howls*. (n.d.). Retrieved June 25, 2020, from https://skarredghost.com/2019/02/15/riftcat-vridge-standalone-phone-controller-steam-vr/

Figure 5.5: *Can't have trackers and controllers ON at the same time. : Vive*. (n.d.). Retrieved June 25, 2020, from https://www.reddit.com/r/Vive/comments/8fz6rn/cant_have_trackers_and_controllers_on_at_the_same/

Figure 5.6: *I don't have the steamvr ready window when I launch steamvr, does anyone know what I can do to fix this? : oculus*. (n.d.). Retrieved June 25, 2020, from https://www.reddit.com/r/oculus/comments/9vywm7/i_dont_have_the_steamvr_ready_window_when_i/

Figure 5.7: *Easily add Interactions to your VR Game with VRTK - VR Tracker*. (n.d.). Retrieved June 25, 2020, from https://vrtracker.xyz/vrtk-vr-tracker-unity/

Figure 6.19: *Circumduction | exsci*. (n.d.). Retrieved June 25, 2020, from https://naveedmazumder.wixsite.com/exsci/blank-c1fy9

Figure 6.20: *Fitness Fix: Help for Weak Wrists - Experience Life*. (n.d.). Retrieved June 25, 2020, from https://experiencelife.com/article/fitness-fix-help-for-weak-wrists/

Figure 6.31: ss_5249920ef928bdcf40453afc66aa0fb021f9c5a0.600x338.jpg (600×337). (n.d.). Retrieved June 23, 2020, from https://steamcdn-a.akamaihd.net/steam/apps/380220/ss_5249920ef928bdcf40453afc66aa0fb021f9c5a0.600x338.jpg?t=1568752998

Figure 8.1: *WEEE - WEEE Directive - Eco-3e*. (n.d.). Retrieved June 25, 2020, from http://eco3e.eu/regulations/weee-regulation/

Figure 8.2: *RoHS and Lead-free Compliance - Eurocircuits Eurocircuits*. (n.d.). Retrieved June 25, 2020, from https://www.eurocircuits.com/blog/rohs-and-lead-free-compliance/

Figure 9.1: dreamstime_s_109105905.jpg (800×533). (n.d.). Retrieved June 23, 2020, from https://www.openaccessgovernment.org/wp-content/uploads/2019/02/dreamstime_s_109105905.jpg

ANNEX A: Volunteer survey template

Volunteer identification: ____________

QUESTIONS BEFORE TESTING

- Gender:
- Age:
- Have you ever undergone any rehabilitation therapy? Yes ☐ No ☐
    - If so, how boring do you think rehabilitation therapies are?

Very boring	**Boring**	**Neither boring nor entertaining**	**Entretaining**	**Very entretaining**

    - If so, did they give you time for the rehabilitation therapy as soon as you asked for it, or did you have to wait a few days to start because there were saturation of people in consulting rooms?
    - If so, was demotivating for you not seeing the real progress done from one therapy to another? Yes ☐ No ☐
- Would you agree to do rehabilitation therapy through VR? Yes ☐ No ☐
    - Why?

QUESTIONS AFTER TESTING

- Was it difficult for you to do the exercise?

Very difficult	**Difficult**	**Normal difficulty**	**Easy**	**Very easy**

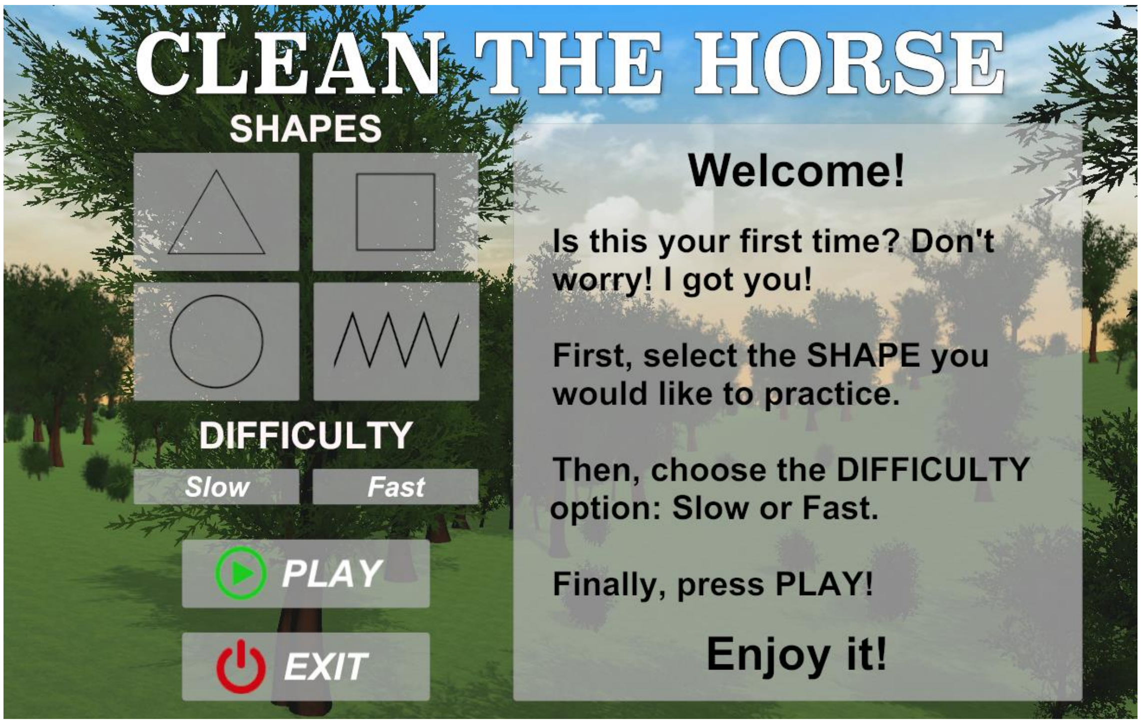
- What score (from 1 to 5) would you give to this experience?
- Do you think the application of VR will improve experiences in rehabilitation sessions?
    - Why?
- Do you think that the game environment is relaxing or there is something that disturbs your calmness while doing the exercise?

- Are the game instructions clear? ☐ Yes ☐ No
    - Why?
- What would you improve from this game?
- Other comments / observations

RESULTS OF THE EXERCISE:

ANNEX B: Instructions for starting the game "Clean the horse"

Once the application is opened, where all the games are placed, run the game called *Clean the horse*. Then, the scene that represents the menu of the game opens.



Welcome!

Is this your first time? Don't worry! I got you!

First, select the SHAPE you would like to practice.

Then, choose the DIFFICULTY option: Slow or Fast.

Finally, press PLAY!

Enjoy it!

Figure B.1 *Scene that represents the menu of the game "Clean the horse".*

As you can see in the instruction panel of the scene, first it has to be selected the shape that the patient would like to practice. After selecting the shape, it has to be selected the difficulty which is based on the time available to do the exercise (slow or fast).

Once the shape and difficulty have been selected, the PLAY button will be enabled. If the PLAY button is pressed, the Game Scene will open. When the Game Scene opens, the patient has to move the character closer to the body of the horse, where the shape selected is placed, by using the arrows of the computer keyboard. When the patients are in front of the trajectory to follow, they have to press the mouse button in order to be able to throw water with the hose. The patient has to begin to trace the trajectory through where the number 1 is placed. At the moment the patient touches it, the game starts, the chronometer begins to count and the countdown begins. Then, the patient has to follow the path following the ascending order of the numbers or following the arrows.

Otherwise, it can be pressed the EXIT button in the Menu Scene, and it will go out of the *Clean the horse* game and go to the main menu where all the games are placed.

Moreover, it is necessary to keep in mind that a mouse must be connected to the computer to carry out the exercises. Lastly, the computer must have the Microsoft Excel program to be able to extract the data correctly. The game will extract the data automatically when the patient returns to the main menu after completing the exercise.

ANNEX C: Horse Cleaning game C# Scripts

C.1 Menu script

```
public class Menu : MonoBehaviour
{

    public static string Shape;
    public static string Difficulty;
    public GameObject play;



    public void ChangeScene(string scenename)
    {
        SceneManager.LoadSceneAsync(scenename);
        Cursor.visible = true;
        Time.timeScale = 1;
    }


    private void Start()
    {
        play.GetComponent<Button>().interactable = false;
        Cursor.visible = true;
    }


    public void Triangle()
    {
        Shape = "triangle";
    }

    public void Square()
    {
        Shape = "square";
    }

    public void Circle()
    {
        Shape = "circle";
    }

    public void Zigzag()
    {
        Shape = "zigzag";
    }

    public void Slow()
    {
        Difficulty = "slow";
        play.GetComponent<Button>().interactable = true;
    }

    public void Fast()
    {
        Difficulty = "fast";
        play.GetComponent<Button>().interactable = true;
    }
}
```

C.2 Game script

```
public class GameScript : MonoBehaviour
{
    private GameObject[] trajectory; //to activate the shape
    private GameObject[] objecte; //to activate colliders
    private GameObject[] other; //to activate MiddleCollider and EndCollider
    public static float maxTime;
    public static string LevelResult;

    // Start is called before the first frame update
    void Start()
    {
        Cursor.visible = false;
        LevelResult = "Level not finished";

        if (Menu.Shape == "triangle" && Menu.Difficulty == "slow")
        {
            trajectory = GameObject.FindGameObjectsWithTag("ShapeSquare");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeCircle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeZigzag");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderSquare");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderCircle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderZigzag");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderSquare");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderCircle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderZigzag");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            maxTime = 120f;
        }

        if (Menu.Shape == "triangle" && Menu.Difficulty == "fast")
        {
            trajectory = GameObject.FindGameObjectsWithTag("ShapeSquare");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeCircle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeZigzag");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderSquare");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderCircle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderZigzag");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderSquare");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderCircle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderZigzag");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            maxTime = 90f;
        }

        if (Menu.Shape == "square" && Menu.Difficulty == "slow")
        {
            trajectory = GameObject.FindGameObjectsWithTag("ShapeTriangle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeCircle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeZigzag");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderTriangle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderCircle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderZigzag");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderTriangle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderCircle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderZigzag");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            maxTime = 120f;
        }

        if (Menu.Shape == "square" && Menu.Difficulty == "fast")
        {
            trajectory = GameObject.FindGameObjectsWithTag("ShapeTriangle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeCircle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeZigzag");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderTriangle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderCircle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderZigzag");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderTriangle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderCircle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderZigzag");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            maxTime = 90f;
        }

        if (Menu.Shape == "circle" && Menu.Difficulty == "slow")
        {
            trajectory = GameObject.FindGameObjectsWithTag("ShapeTriangle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeSquare");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeZigzag");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderTriangle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderSquare");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderZigzag");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderTriangle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderSquare");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderZigzag");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            maxTime = 120f;
        }
        if (Menu.Shape == "circle" && Menu.Difficulty == "fast")
        {
            trajectory = GameObject.FindGameObjectsWithTag("ShapeTriangle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeSquare");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeZigzag");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderTriangle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderSquare");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderZigzag");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderTriangle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderSquare");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderZigzag");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            maxTime = 90f;
        }

        if (Menu.Shape == "zigzag" && Menu.Difficulty == "slow")
        {
            trajectory = GameObject.FindGameObjectsWithTag("ShapeTriangle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeSquare");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeCircle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderTriangle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderSquare");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderCircle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderTriangle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderSquare");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderCircle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            maxTime = 180f;
        }

        if (Menu.Shape == "zigzag" && Menu.Difficulty == "fast")
        {
            trajectory = GameObject.FindGameObjectsWithTag("ShapeTriangle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeSquare");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            trajectory = GameObject.FindGameObjectsWithTag("ShapeCircle");
            foreach (GameObject element in trajectory)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderTriangle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderSquare");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            objecte = GameObject.FindGameObjectsWithTag("ColliderCircle");
            foreach (GameObject element in objecte)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderTriangle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderSquare");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            other = GameObject.FindGameObjectsWithTag("OtherColliderCircle");
            foreach (GameObject element in other)
            {
                element.SetActive(false);
            }

            maxTime = 120f;
        }
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            SceneManager.LoadScene("Menu");
        }
    }
}
```

Timer script

```
public class Timer : MonoBehaviour
{

    public Text timerText;  //text in the chronometer for the numbers
    public GameObject backgroundText;    //text in the chronometer before
                                         //starting the time
    private float startTime;
    private float stopTime;
    private float t;
    private float minutesInt;
    private float secondsInt;
    public static bool initiateCheck;    //check if time is running or not
    public Text timeText;   //shows the time that the exercise has been done at
                            //the end of the exercisez
    public static string timeString;     //data to export at the end of the
                                         //game (in string format)


    void Start()
    {
        initiateCheck = false;
        timeText.gameObject.SetActive(false);
        timerText.gameObject.SetActive(false);
        backgroundText.SetActive(true);
    }


    void Update()
    {
        if (initiateCheck == true)
        {
            t = Time.time - startTime;
            minutesInt = ((int)t / 60);
            secondsInt = (t % 60);

            string minutes = minutesInt.ToString();
            string seconds = secondsInt.ToString("f0");

            timerText.text = minutes + ":" + seconds;
            timeText.text = timerText.text;
            timeString = timerText.text.ToString();
        }

        if (initiateCheck == false)
        {
            stopTime = t;
        }
    }



    public void Finnish()
    {
        timeText.gameObject.SetActive(true);
        initiateCheck = false;
    }


    private void OnParticleCollision(GameObject col)
    {
        if (col.gameObject.name == "InitiateCollider")
        {
            backgroundText.SetActive(false);
            timerText.gameObject.SetActive(true);
            initiateCheck = true;
            col.SetActive(false);
            startTime = Time.time;
        }
    }
}
```

TimeBar script

```
public class TimeBar : MonoBehaviour
{
    Image timerBar;
    float timeLeft;
    public GameObject IncompleteLevel;
    public GameObject TopPanel;
    public GameObject Chronometer;


    void Start()
    {
        IncompleteLevel.SetActive(false);
        timerBar = GetComponent<Image> ();
        timeLeft = GameScript.maxTime;
        TopPanel.SetActive(true);
        Chronometer.SetActive(true);
    }

    void Update()
    {
        if (Timer.initiateCheck == true)
        {
            if (timeLeft > 0)
            {
                timeLeft -= Time.deltaTime;
                timerBar.fillAmount = timeLeft / GameScript.maxTime;
            }
            else
            {
                TopPanel.SetActive(false);
                Chronometer.SetActive(false);
                IncompleteLevel.SetActive(true);
                Cursor.visible = true;
                Time.timeScale = 0;
                GameScript.LevelResult = "Incomplete level";
            }
        }
    }
}
```

Colliders script

```
public class Colliders : MonoBehaviour
{
    public int numCol;    //number of colliders in the selected shape
    public int count;   //number of destroyed colliders
    public int count1;  //number of destroyed colliders in repetition 1
    public int count2;  //number of destroyed colliders in repetition 2
    public int count3;  //number of destroyed colliders in repetition 3
    public int result;  //average of destroyed colliders (%)
    public Text resultText;  //shows the final % of colliders destroyed
    public static string resultString;  //data to be exported in string format
    public int repetition;
    public static string finalRepetition;//data to be exported in string format
    private GameObject MiddleCollider;
    private GameObject EndCollider;
    private GameObject[] colliders;
    public GameObject CompleteLevel;
    public GameObject chronometer;
    public GameObject TopPanel;
    public GameObject TimeBar;
    public GameObject repetition1;
    public GameObject repetition2;
    public GameObject repetition3;


    void Start()
    {
        count = 0;
        count1 = 0;
        count2 = 0;
        count3 = 0;
        result = 0;
        repetition = 0;
        CompleteLevel.SetActive(false);
        repetition1.SetActive(false);
        repetition2.SetActive(false);
        repetition3.SetActive(false);
        finalRepetition = "0";
        resultString = "0";

        if (Menu.Shape == "triangle")
        {
            numCol = 31;
            colliders = GameObject.FindGameObjectsWithTag("ColliderTriangle");
            MiddleCollider = GameObject.Find("MiddleColliderTriangle");
            EndCollider = GameObject.Find("EndColliderTriangle");
            EndCollider.SetActive(false);
        }

        if (Menu.Shape == "square")
        {
            numCol = 39;
            colliders = GameObject.FindGameObjectsWithTag("ColliderSquare");
            MiddleCollider = GameObject.Find("MiddleColliderSquare");
            EndCollider = GameObject.Find("EndColliderSquare");
            EndCollider.SetActive(false);
        }

        if (Menu.Shape == "circle")
        {
            numCol = 34;
            colliders = GameObject.FindGameObjectsWithTag("ColliderCircle");
            MiddleCollider = GameObject.Find("MiddleColliderCircle");
            EndCollider = GameObject.Find("EndColliderCircle");
            EndCollider.SetActive(false);
        }
```

```
        if (Menu.Shape == "zigzag")
        {
            numCol = 66;
            colliders = GameObject.FindGameObjectsWithTag("ColliderZigzag");
            MiddleCollider = GameObject.Find("MiddleColliderZigzag");
            EndCollider = GameObject.Find("EndColliderZigzag");
            EndCollider.SetActive(false);
        }
    }


    private void OnParticleCollision(GameObject col)
    {
        if (col.gameObject.name == "Collider")
        {
            count += 1;
            col.SetActive(false);
        }

        if (col == MiddleCollider)
        {
            EndCollider.SetActive(true);
        }

        if (col == EndCollider)
        {
            repetition += 1;
            if (repetition == 1)
            {
                finalRepetition = "1";
                count1 = count;
                count = 0;
                result = count1 * 100 / numCol;
                resultString = result.ToString();
                repetition1.SetActive(true);
            }
            if (repetition == 2)
            {
                finalRepetition = "2";
                count2 = count;
                count = 0;
                result = ((count1 + count2) / 2) * 100 / numCol;
                resultString = result.ToString();
                repetition1.SetActive(false);
                repetition2.SetActive(true);
            }
            if (repetition == 3)
            {
                finalRepetition = "3";
                count3 = count;
                count = 0;
                repetition2.SetActive(false);
                repetition3.SetActive(true);
                result = ((count1 + count2 + count3) / 3) * 100 / numCol;
                resultText.text = result.ToString();
                resultString = result.ToString();
                GameObject.Find("Particle
Launcher").GetComponent<Timer>().Finnish();
                TopPanel.SetActive(false);
                CompleteLevel.SetActive(true);
                chronometer.SetActive(false);
                TimeBar.SetActive(false);
                Cursor.visible = true;
                GameScript.LevelResult = "Complete level";
            }

            EndCollider.SetActive(false);

            foreach (GameObject colli in colliders)
            {
                colli.SetActive(true);
            }
        }
    }
}
```

C.3 Character scripts

This script was downloaded with the pack of assets in the folder *AssetParticleScripting*.

```
namespace UnityStandardAssets.Characters.FirstPerson
{
    [RequireComponent(typeof (CharacterController))]
    [RequireComponent(typeof (AudioSource))]
    public class FirstPersonController : MonoBehaviour
    {
        [SerializeField] private bool m_IsWalking;
        [SerializeField] private float m_WalkSpeed = 5;
        [SerializeField] private float m_RunSpeed = 10;
        [SerializeField] [Range(0f, 1f)] private float m_RunstepLenghten = 0.7f;
        [SerializeField] private float m_JumpSpeed = 10;
        [SerializeField] private float m_StickToGroundForce = 10;
        [SerializeField] private float m_GravityMultiplier = 2;
        [SerializeField] private MouseLook m_MouseLook;
        [SerializeField] private bool m_UseFovKick;
        [SerializeField] private FOVKick m_FovKick = new FOVKick();
        [SerializeField] private bool m_UseHeadBob = false;
        [SerializeField] private CurveControlledBob m_HeadBob = new
CurveControlledBob();
        [SerializeField] private LerpControlledBob m_JumpBob = new
LerpControlledBob();
        [SerializeField] private float m_StepInterval = 5;
        [SerializeField] private AudioClip[] m_FootstepSounds;    // an array of
footstep sounds that will be randomly selected from.
        [SerializeField] private AudioClip m_JumpSound;           // the sound
played when character leaves the ground.
        [SerializeField] private AudioClip m_LandSound;           // the sound
played when character touches back on ground.

        private Camera m_Camera;
        private bool m_Jump;
        private float m_YRotation;
        private Vector2 m_Input;
        private Vector3 m_MoveDir = Vector3.zero;
        private CharacterController m_CharacterController;
        private CollisionFlags m_CollisionFlags;
        private bool m_PreviouslyGrounded;
        private Vector3 m_OriginalCameraPosition;
        private float m_StepCycle;
        private float m_NextStep;
        private bool m_Jumping;
        private AudioSource m_AudioSource;

        // Use this for initialization
        private void Start()
        {
            m_CharacterController = GetComponent<CharacterController>();
            m_Camera = Camera.main;
            m_OriginalCameraPosition = m_Camera.transform.localPosition;
            m_FovKick.Setup(m_Camera);
            m_HeadBob.Setup(m_Camera, m_StepInterval);
            m_StepCycle = 0f;
            m_NextStep = m_StepCycle/2f;
            m_Jumping = false;
            m_AudioSource = GetComponent<AudioSource>();
                    m_MouseLook.Init(transform , m_Camera.transform);
        }

        // Update is called once per frame
        private void Update()
        {
            RotateView();
            // the jump state needs to read here to make sure it is not missed
            if (!m_Jump)
            {
                m_Jump = CrossPlatformInputManager.GetButtonDown("Jump");
            }

            if (!m_PreviouslyGrounded && m_CharacterController.isGrounded)
            {
                StartCoroutine(m_JumpBob.DoBobCycle());
                PlayLandingSound();
                m_MoveDir.y = 0f;
                m_Jumping = false;
            }
            if (!m_CharacterController.isGrounded && !m_Jumping &&
m_PreviouslyGrounded)
            {
                m_MoveDir.y = 0f;
            }

            m_PreviouslyGrounded = m_CharacterController.isGrounded;
        }

        private void PlayLandingSound()
        {
            m_AudioSource.clip = m_LandSound;
            m_AudioSource.Play();
            m_NextStep = m_StepCycle + .5f;
        }

        private void FixedUpdate()
        {
            float speed;
            GetInput(out speed);
            // always move along the camera forward as it is the direction that
it being aimed at
            Vector3 desiredMove = transform.forward*m_Input.y +
transform.right*m_Input.x;

            // get a normal for the surface that is being touched to move along
it
```

```
            RaycastHit hitInfo;
            Physics.SphereCast(transform.position, m_CharacterController.radius,
Vector3.down, out hitInfo,
                               m_CharacterController.height/2f, ~0,
QueryTriggerInteraction.Ignore);
            desiredMove = Vector3.ProjectOnPlane(desiredMove,
hitInfo.normal).normalized;

            m_MoveDir.x = desiredMove.x*speed;
            m_MoveDir.z = desiredMove.z*speed;

            if (m_CharacterController.isGrounded)
            {
                m_MoveDir.y = -m_StickToGroundForce;

                if (m_Jump)
                {
                    m_MoveDir.y = m_JumpSpeed;
                    PlayJumpSound();
                    m_Jump = false;
                    m_Jumping = true;
                }
            }
            else
            {
                m_MoveDir +=
Physics.gravity*m_GravityMultiplier*Time.fixedDeltaTime;
            }
            m_CollisionFlags =
m_CharacterController.Move(m_MoveDir*Time.fixedDeltaTime);

            ProgressStepCycle(speed);
            UpdateCameraPosition(speed);

            m_MouseLook.UpdateCursorLock();
        }

        private void PlayJumpSound()
        {
            m_AudioSource.clip = m_JumpSound;
            m_AudioSource.Play();
        }

        private void ProgressStepCycle(float speed)
        {
            if (m_CharacterController.velocity.sqrMagnitude > 0 && (m_Input.x !=
0 || m_Input.y != 0))
            {
                m_StepCycle += (m_CharacterController.velocity.magnitude +
(speed*(m_IsWalking ? 1f : m_RunstepLenghten)))*
                             Time.fixedDeltaTime;
            }

            if (!(m_StepCycle > m_NextStep))
            {
                return;
            }

            m_NextStep = m_StepCycle + m_StepInterval;

            PlayFootStepAudio();
        }
        private void PlayFootStepAudio()
        {
            if (!m_CharacterController.isGrounded)
            {
                return;
            }
            // pick & play a random footstep sound from the array,
            // excluding sound at index 0
            int n = Random.Range(1, m_FootstepSounds.Length);
            m_AudioSource.clip = m_FootstepSounds[n];
            m_AudioSource.PlayOneShot(m_AudioSource.clip);
            // move picked sound to index 0 so it's not picked next time
            m_FootstepSounds[n] = m_FootstepSounds[0];
            m_FootstepSounds[0] = m_AudioSource.clip;
        }

        private void UpdateCameraPosition(float speed)
        {
            Vector3 newCameraPosition;
            if (!m_UseHeadBob)
            {
                return;
            }
            if (m_CharacterController.velocity.magnitude > 0 &&
m_CharacterController.isGrounded)
            {
                m_Camera.transform.localPosition =
                    m_HeadBob.DoHeadBob(m_CharacterController.velocity.magnitude
+
                                      (speed*(m_IsWalking ? 1f :
m_RunstepLenghten)));
                newCameraPosition = m_Camera.transform.localPosition;
                newCameraPosition.y = m_Camera.transform.localPosition.y -
m_JumpBob.Offset();
            }
            else
            {
                newCameraPosition = m_Camera.transform.localPosition;
                newCameraPosition.y = m_OriginalCameraPosition.y -
m_JumpBob.Offset();
            }
            m_Camera.transform.localPosition = newCameraPosition;
        }

        private void GetInput(out float speed)
        {
            // Read input
            float horizontal = CrossPlatformInputManager.GetAxis("Horizontal");
            float vertical = CrossPlatformInputManager.GetAxis("Vertical");

            bool waswalking = m_IsWalking;

#if !MOBILE_INPUT
            // On standalone builds, walk/run speed is modified by a key press.
            // keep track of whether or not the character is walking or running
            m_IsWalking = !Input.GetKey(KeyCode.LeftShift);
#endif
            // set the desired speed to be walking or running
            speed = m_IsWalking ? m_WalkSpeed : m_RunSpeed;
            m_Input = new Vector2(horizontal, vertical);
            // normalize input if it exceeds 1 in combined length:
            if (m_Input.sqrMagnitude > 1)
            {
                m_Input.Normalize();
            }

            // handle speed change to give an fov kick
            // only if the player is going to a run, is running and the fovkick
is to be used
            if (m_IsWalking != waswalking && m_UseFovKick &&
m_CharacterController.velocity.sqrMagnitude > 0)
            {
                StopAllCoroutines();
                StartCoroutine(!m_IsWalking ? m_FovKick.FOVKickUp() :
m_FovKick.FOVKickDown());
            }
        }

        private void RotateView()
        {
            m_MouseLook.LookRotation (transform, m_Camera.transform);
        }


        private void OnControllerColliderHit(ControllerColliderHit hit)
        {
            Rigidbody body = hit.collider.attachedRigidbody;
            //dont move the rigidbody if the character is on top of it
            if (m_CollisionFlags == CollisionFlags.Below)
            {
                return;
            }

            if (body == null || body.isKinematic)
            {
                return;
            }
            body.AddForceAtPosition(m_CharacterController.velocity*0.1f,
hit.point, ForceMode.Impulse);
        }
    }
}
```

ParticleLauncher script

```
public class ParticleLauncher : MonoBehaviour
{

    public ParticleSystem particleLauncher; //reference to the particle system
    public ParticleSystem splatterParticles;
    public Gradient particleColorGradient;
    public ParticleDecalPool splatDecalPool;

    List<ParticleCollisionEvent> collisionEvents; //store the results from
OnParticleCollision


    void Start()
    {
        collisionEvents = new List<ParticleCollisionEvent>();
    }

    private void OnParticleCollision(GameObject other) //emit a spray of
particles at the location that we shot
    {
        ParticlePhysicsExtensions.GetCollisionEvents(particleLauncher, other,
collisionEvents);

        for (int i = 0; i < collisionEvents.Count; i++)
        {
            splatDecalPool.ParticleHit(collisionEvents[i],
particleColorGradient);
            EmitAtLocation(collisionEvents[i]);
        }
    }

    private void EmitAtLocation(ParticleCollisionEvent particleCollisionEvent)
    {
        splatterParticles.transform.position =
particleCollisionEvent.intersection;
        splatterParticles.transform.rotation =
Quaternion.LookRotation(particleCollisionEvent.normal);
        ParticleSystem.MainModule psMain = splatterParticles.main; //reference to
the Main Module. Below we define the values.          l
        psMain.startColor = particleColorGradient.Evaluate(Random.Range(0f, 1f));
//selects a random color
        splatterParticles.Emit(1);
    }

    void Update()
    {
        if (Input.GetButton ("Fire1")) //when the player presses the left ouse
button
        {
            ParticleSystem.MainModule psMain = particleLauncher.main; //reference
to the Main Module. Below we define the values.
            psMain.startColor = particleColorGradient.Evaluate(Random.Range(0f,
1f)); //selects a random color
            particleLauncher.Emit(1); // spon a certain number of particles when
the function is called. Emit 1 particle in every frame. l
        }
    }
}
```

SplatOnCollision script

```
public class SplatOnCollision : MonoBehaviour {

       public ParticleSystem particleLauncher;
       public Gradient particleColorGradient;
       public ParticleDecalPool dropletDecalPool;

       List<ParticleCollisionEvent> collisionEvents;

       void Start ()
       {
              collisionEvents = new List<ParticleCollisionEvent> ();
       }

       void OnParticleCollision(GameObject other)
       {
            int numCollisionEvents =
ParticlePhysicsExtensions.GetCollisionEvents (particleLauncher, other,
collisionEvents);

            int i = 0;
            while (i < numCollisionEvents)
            {
                  dropletDecalPool.ParticleHit(collisionEvents[i],
particleColorGradient);
                  i++;
            }
      }
}
```

ParticleDecalData script

```
public class ParticleDecalData
{
    public Vector3 position;
    public float size;
    public Vector3 rotation;
    public Color color;
}
```

ParticleDecalPool script

```
public class ParticleDecalPool : MonoBehaviour
{
    public int maxDecals = 100;
    public float decalSizeMin = 0.5f;
    public float decalSizeMax = 1.5f;

    private ParticleSystem decalParticleSystem;
    private int particleDecalDataIndex;
    private ParticleDecalData[] particleData;
    private ParticleSystem.Particle[] particles;

    void Start()
    {
        decalParticleSystem = GetComponent<ParticleSystem>();
        particles = new ParticleSystem.Particle[maxDecals];
        particleData = new ParticleDecalData[maxDecals];
        for (int i = 0; i < maxDecals; i++)
        {
            particleData[i] = new ParticleDecalData();
        }
    }

    public void ParticleHit(ParticleCollisionEvent particleCollisionEvent,
Gradient colorGradient)
    {
        setParticleData(particleCollisionEvent, colorGradient);
        DisplayParticles();
    }

    void setParticleData(ParticleCollisionEvent particleCollisionEvent, Gradient
colorGradient)
    {
        if (particleDecalDataIndex >= maxDecals)
        {
            particleDecalDataIndex = 0;
        }

        particleData[particleDecalDataIndex].position =
particleCollisionEvent.intersection; //record collision position, rotation, size
and colour
        Vector3 particleRotationEuler =
Quaternion.LookRotation(particleCollisionEvent.normal).eulerAngles;
        particleRotationEuler.z = Random.Range(0,360);
        particleData[particleDecalDataIndex].rotation = particleRotationEuler;
        particleData[particleDecalDataIndex].size = Random.Range(decalSizeMin,
decalSizeMax);
        particleData[particleDecalDataIndex].color =
colorGradient.Evaluate(Random.Range(0f, 1f));

        particleDecalDataIndex++;
    }

    void DisplayParticles()
    {
        for (int i = 0; i < particleData.Length; i++)
        {
            particles[i].position = particleData[i].position;
            particles[i].rotation3D = particleData[i].rotation;
            particles[i].startSize = particleData[i].size;
            particles[i].startColor = particleData[i].color;
        }
        decalParticleSystem.SetParticles(particles, particles.Length);
    }
}
```

C.4 Exporting data scripts

CSVManager script (this script is used for the extraction of data)

```
public static class CSVManager
{
    private static string reportDirectoryName = "Report";    //folder name
    private static string reportFileName = "CleanTheHorseData.csv"; //file name
    private static string reportSeparator = ",";    //colum separator
    private static string[] reportHeaders = new string[6]    //number of headers
    {
        "shape chosen","difficulty chosen","level result","repetitions","accuracy
(%)","time doing the exercise (minutes)"
    };
    private static string timeStampHeader = "time stamp";

    #region Interactions
    public static void AppendToReport(string[] strings)
    {
        VerifyDirectory();
        VerifyFile();
        using (StreamWriter sw = File.AppendText(GetFilePath()))
        {
            string finalstring = "";
            for (int i = 0; i < strings.Length; i++)
            {
                if (finalstring != "")
                {
                    finalstring += reportSeparator;
                }
                finalstring += strings[i];
            }
            finalstring += reportSeparator + GetTimeStamp();
            sw.WriteLine(finalstring);
        }
    }
    public static void CreateReport()
    {
        using (StreamWriter sw = File.CreateText(GetFilePath()))
        {
            VerifyDirectory();
            string finalstring = "";
            for (int i = 0; i < reportHeaders.Length; i++)
            {
                if (finalstring != "")
                {
                    finalstring += reportSeparator;
                }
                finalstring += reportHeaders[i];
            }
            finalstring += reportSeparator + timeStampHeader;
            sw.WriteLine(finalstring);
        }
    }
    #endregion

    #region Operations
    static void VerifyDirectory()   //verify that the information above exists
    {
        string dir = GetDirectoryPath();
        if (!Directory.Exists(dir))
        {
            Directory.CreateDirectory(dir);
        }
    }
    static void VerifyFile()
    {
        string file = GetFilePath();
        if (!File.Exists(file))
        {
            CreateReport();
        }
    }
    #endregion

    #region Queries
    static string GetDirectoryPath()
    {
        return Application.persistentDataPath + "/" + reportDirectoryName;
    }
    static string GetFilePath()
    {
        return GetDirectoryPath() + "/" + reportFileName;
    }
    static string GetTimeStamp()
    {
        return System.DateTime.Now.ToString();
    }
    #endregion
}
```

MyTools script (this script is used for the extraction of data)

```
public class MyTools: MonoBehaviour
{
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            CSVManager.AppendToReport(
                new string[6]
                {

Menu.Shape,Menu.Difficulty,GameScript.LevelResult,Colliders.finalRepetition,Colli
ders.resultString,Timer.timeString
                }
                );
//            EditorApplication.Beep();
        }
    }
    //in case you want to reset the document. In our case we will not use it.
    static void DEV_ResetReport()
    {
        CSVManager.CreateReport();
//        EditorApplication.Beep();
    }
}
```

ANNEX D: Mirror Cleaning game C# Scripts

D.1 Environment script

Mirror reflection

```
[ExecuteInEditMode] // Make mirror live-update even when not in play mode
public class MirrorReflection : MonoBehaviour
{
    public bool m_DisablePixelLights = true;
    public int m_TextureSize = 256;
    public float m_ClipPlaneOffset = 0.07f;

    public LayerMask m_ReflectLayers = -1;

    private Hashtable m_ReflectionCameras = new Hashtable(); // Camera -> Camera
table

    private RenderTexture m_ReflectionTexture = null;
    private int m_OldReflectionTextureSize = 0;

    private static bool s_InsideRendering = false;

    // This is called when it's known that the object will be rendered by some
    // camera. We render reflections and do other updates here.
    // Because the script executes in edit mode, reflections for the scene view
    // camera will just work!
    public void OnWillRenderObject()
    {
        var rend = GetComponent<Renderer>();
        if (!enabled || !rend || !rend.sharedMaterial || !rend.enabled)
            return;

        Camera cam = Camera.current;
        if (!cam)
            return;

        // Safeguard from recursive reflections.
        if (s_InsideRendering)
            return;
        s_InsideRendering = true;

        Camera reflectionCamera;
        CreateMirrorObjects(cam, out reflectionCamera);

        // find out the reflection plane: position and normal in world space
        Vector3 pos = transform.position;
        Vector3 normal = transform.up;

        // Optionally disable pixel lights for reflection
        int oldPixelLightCount = QualitySettings.pixelLightCount;
        if (m_DisablePixelLights)
            QualitySettings.pixelLightCount = 0;

        UpdateCameraModes(cam, reflectionCamera);
        // Render reflection
        // Reflect camera around reflection plane
        float d = -Vector3.Dot(normal, pos) - m_ClipPlaneOffset;
        Vector4 reflectionPlane = new Vector4(normal.x, normal.y, normal.z, d);
        Matrix4x4 reflection = Matrix4x4.zero;
        CalculateReflectionMatrix(ref reflection, reflectionPlane);
        Vector3 oldpos = cam.transform.position;
        Vector3 newpos = reflection.MultiplyPoint(oldpos);
        reflectionCamera.worldToCameraMatrix = cam.worldToCameraMatrix *
reflection;

        // Setup oblique projection matrix so that near plane is our reflection
        // plane. This way we clip everything below/above it for free.
        Vector4 clipPlane = CameraSpacePlane(reflectionCamera, pos, normal,
1.0f);
        //Matrix4x4 projection = cam.projectionMatrix;
        Matrix4x4 projection = cam.CalculateObliqueMatrix(clipPlane);
        reflectionCamera.projectionMatrix = projection;

        reflectionCamera.cullingMask = ~(1 << 4) & m_ReflectLayers.value; //
never render water layer
        reflectionCamera.targetTexture = m_ReflectionTexture;
        GL.SetRevertBackfacing(true);
        reflectionCamera.transform.position = newpos;
        Vector3 euler = cam.transform.eulerAngles;
        reflectionCamera.transform.eulerAngles = new Vector3(0, euler.y,
euler.z);
        reflectionCamera.Render();
        reflectionCamera.transform.position = oldpos;
        GL.SetRevertBackfacing(false);
        Material[] materials = rend.sharedMaterials;
        foreach (Material mat in materials)
        {
            if (mat.HasProperty("_ReflectionTex"))
                mat.SetTexture("_ReflectionTex", m_ReflectionTexture);
        }

        // Restore pixel light count
        if (m_DisablePixelLights)
            QualitySettings.pixelLightCount = oldPixelLightCount;

        s_InsideRendering = false;
    }


    // Cleanup all the objects we possibly have created
    void OnDisable()
    {
        if (m_ReflectionTexture)
        {
            DestroyImmediate(m_ReflectionTexture);
            m_ReflectionTexture = null;
        }
        foreach (DictionaryEntry kvp in m_ReflectionCameras)
            DestroyImmediate(((Camera)kvp.Value).gameObject);
        m_ReflectionCameras.Clear();
    }


    private void UpdateCameraModes(Camera src, Camera dest)
    {
        if (dest == null)
            return;
        // set camera to clear the same way as current camera
        dest.clearFlags = src.clearFlags;
        dest.backgroundColor = src.backgroundColor;
        if (src.clearFlags == CameraClearFlags.Skybox)
        {
            Skybox sky = src.GetComponent(typeof(Skybox)) as Skybox;
            Skybox mysky = dest.GetComponent(typeof(Skybox)) as Skybox;
            if (!sky || !sky.material)
            {
                mysky.enabled = false;
            }
            else
            {
                mysky.enabled = true;
                mysky.material = sky.material;
            }
        }
        // update other values to match current camera.
        // even if we are supplying custom camera&projection matrices,
        // some of values are used elsewhere (e.g. skybox uses far plane)
        dest.farClipPlane = src.farClipPlane;
        dest.nearClipPlane = src.nearClipPlane;
        dest.orthographic = src.orthographic;
        dest.fieldOfView = src.fieldOfView;
        dest.aspect = src.aspect;
        dest.orthographicSize = src.orthographicSize;
    }

    // On-demand create any objects we need
    private void CreateMirrorObjects(Camera currentCamera, out Camera
reflectionCamera)
    {
        reflectionCamera = null;

        // Reflection render texture
        if (!m_ReflectionTexture || m_OldReflectionTextureSize != m_TextureSize)
        {
            if (m_ReflectionTexture)
                DestroyImmediate(m_ReflectionTexture);
            m_ReflectionTexture = new RenderTexture(m_TextureSize, m_TextureSize,
16);
            m_ReflectionTexture.name = "__MirrorReflection" + GetInstanceID();
            m_ReflectionTexture.isPowerOfTwo = true;
            m_ReflectionTexture.hideFlags = HideFlags.DontSave;
            m_OldReflectionTextureSize = m_TextureSize;
        }

        // Camera for reflection
        reflectionCamera = m_ReflectionCameras[currentCamera] as Camera;
        if (!reflectionCamera) // catch both not-in-dictionary and in-dictionary-
but-deleted-GO
        {
            GameObject go = new GameObject("Mirror Refl Camera id" +
GetInstanceID() + " for " + currentCamera.GetInstanceID(), typeof(Camera),
typeof(Skybox));
            reflectionCamera = go.GetComponent<Camera>();
            reflectionCamera.enabled = false;
            reflectionCamera.transform.position = transform.position;
            reflectionCamera.transform.rotation = transform.rotation;
            reflectionCamera.gameObject.AddComponent<FlareLayer>();
            go.hideFlags = HideFlags.HideAndDontSave;
            m_ReflectionCameras[currentCamera] = reflectionCamera;
        }
    }

    // Extended sign: returns -1, 0 or 1 based on sign of a
    private static float sgn(float a)
    {
        if (a > 0.0f) return 1.0f;
        if (a < 0.0f) return -1.0f;
        return 0.0f;
    }

    // Given position/normal of the plane, calculates plane in camera space.
    private Vector4 CameraSpacePlane(Camera cam, Vector3 pos, Vector3 normal,
float sideSign)
    {
        Vector3 offsetPos = pos + normal * m_ClipPlaneOffset;
        Matrix4x4 m = cam.worldToCameraMatrix;
        Vector3 cpos = m.MultiplyPoint(offsetPos);
        Vector3 cnormal = m.MultiplyVector(normal).normalized * sideSign;
        return new Vector4(cnormal.x, cnormal.y, cnormal.z, -Vector3.Dot(cpos,
cnormal));
    }

    // Calculates reflection matrix around the given plane
    private static void CalculateReflectionMatrix(ref Matrix4x4 reflectionMat,
Vector4 plane)
    {
        reflectionMat.m00 = (1F - 2F * plane[0] * plane[0]);
        reflectionMat.m01 = (-2F * plane[0] * plane[1]);
        reflectionMat.m02 = (-2F * plane[0] * plane[2]);
        reflectionMat.m03 = (-2F * plane[3] * plane[0]);

        reflectionMat.m10 = (-2F * plane[1] * plane[0]);
        reflectionMat.m11 = (1F - 2F * plane[1] * plane[1]);
        reflectionMat.m12 = (-2F * plane[1] * plane[2]);
        reflectionMat.m13 = (-2F * plane[3] * plane[1]);

        reflectionMat.m20 = (-2F * plane[2] * plane[0]);
        reflectionMat.m21 = (-2F * plane[2] * plane[1]);
        reflectionMat.m22 = (1F - 2F * plane[2] * plane[2]);
        reflectionMat.m23 = (-2F * plane[3] * plane[2]);

        reflectionMat.m30 = 0F;
        reflectionMat.m31 = 0F;
        reflectionMat.m32 = 0F;
        reflectionMat.m33 = 1F;
    }
}
```

D.2 Character scripts

Robot and Camera_Rig pairing

```
[System.Serializable]
public class VRMap
{
    public Transform vrTarget;
    public Transform rigTarget;
    public Vector3 trackingpositionoffset;
    public Vector3 trackingrotationoffset;

    public void Map ()
    {
        rigTarget.position = vrTarget.TransformPoint(trackingpositionoffset);
        rigTarget.rotation = vrTarget.rotation *
Quaternion.Euler(trackingrotationoffset);
    }
}
public class VRRig : MonoBehaviour
{
    public VRMap head;
    public VRMap LeftHand;
    public VRMap RightHand;
    public Transform headConstraint;
    public Vector3 headBodyOffset;
    public float turnSmothness;

    // Start is called before the first frame update
    void Start()
    {
        headBodyOffset = transform.position - headConstraint.position;
    }

    // Update is called once per frame
    void LateUpdate()
    {
        transform.position = headConstraint.position + headBodyOffset;
        transform.forward = Vector3.Lerp(transform.forward,
Vector3.ProjectOnPlane(headConstraint.up, Vector3.up).normalized, Time.deltaTime
* turnSmothness);

        head.Map();
        LeftHand.Map();
        RightHand.Map();
    }
}
```

Pointer

```
public class Pointer : MonoBehaviour
{
    public float m_DefaultLength = 5.0f;
    public GameObject m_Dot;
    public VrInputModule m_InputModule;

    private LineRenderer m_LineRenderer = null;
    private void Awake()
    {
        m_LineRenderer = GetComponent<LineRenderer>();
    }

    // Update is called once per frame
    void Update()
    {
        UpdateLine();
    }

    private void UpdateLine()
    {
        //Use default or distance
        PointerEventData data = m_InputModule.GetData();
        float targetlenght = data.pointerCurrentRaycast.distance == 0 ?
m_DefaultLength : data.pointerCurrentRaycast.distance;

        //Raycast
        RaycastHit hit = CreateRaycast(targetlenght);

        //Default
        Vector3 endPosition = transform.position + (transform.forward *
targetlenght);

        //Or based on hit
        if (hit.collider != null)
            endPosition = hit.point;

        //Set position of the dot
        m_Dot.transform.position = endPosition;

        //Set linerenderer
        m_LineRenderer.SetPosition(0, transform.position);
        m_LineRenderer.SetPosition(1, endPosition);
    }

    private RaycastHit CreateRaycast (float lenght)
    {
        RaycastHit hit;

        Ray ray = new Ray(transform.position, transform.forward);
        Physics.Raycast(ray, out hit, m_DefaultLength);

        return hit;
    }
}
```

Pointer Input module

```
public class VrInputModule : BaseInputModule
{
    public Camera m_Camera;
    public SteamVR_Input_Sources m_TargetSource;
    public SteamVR_Action_Boolean m_ClickAction;
    private GameObject m_CurrentObject = null;
    private PointerEventData m_Data = null;
    protected override void Awake()
    {
        base.Awake();
        m_Data = new PointerEventData(eventSystem);
    }

    public override void Process()
    {
        //Reset data, set camera
        m_Data.Reset();
        m_Data.position = new Vector2(m_Camera.pixelWidth / 2,
m_Camera.pixelHeight / 2);

        //RayCast
        eventSystem.RaycastAll(m_Data, m_RaycastResultCache);
        m_Data.pointerCurrentRaycast = FindFirstRaycast(m_RaycastResultCache);
        m_CurrentObject = m_Data.pointerCurrentRaycast.gameObject;

        //Clear
        m_RaycastResultCache.Clear();

        //Hover
        HandlePointerExitAndEnter(m_Data, m_CurrentObject);

        //Press
        if (m_ClickAction.GetStateDown(m_TargetSource))
           ProcessPress(m_Data);
    }

    public PointerEventData GetData()
    {
        return m_Data;
    }

    private void ProcessPress(PointerEventData data)
    {
        // Set raycast
        data.pointerPressRaycast = data.pointerCurrentRaycast;
        // Check for object hit, get the down handler, call
        GameObject newPointerPress =
ExecuteEvents.ExecuteHierarchy(m_CurrentObject, data,
ExecuteEvents.pointerClickHandler);
        // If no down handler, try and get click handler
        if (newPointerPress == null)
            newPointerPress =
ExecuteEvents.GetEventHandler<IPointerClickHandler>(m_CurrentObject);
        // Set data
        data.pressPosition = data.position;
        data.pointerPress = newPointerPress;
        data.rawPointerPress = m_CurrentObject;
    }
}
```

D.3 UI Scripts

Global Script

This script will be the main script which condition all the process from the beginning to the end.

```
public class GlobalScript : MonoBehaviour
{
    public Text Instructions;
    public GameObject[] Interactors1;
    public GameObject[] Interactors2;
    public GameObject CleanButton;
    public GameObject HandArrows;
    public GameObject LevelArrows;
    public GameObject TimerArrow;
    public GameObject OneHandTutorialButton;
    public GameObject TwoHandTutorialButton;
    public GameObject TutorialTimerOneHand;
    public GameObject TutorialTimerBothHands;
    public GameObject[] Arrows;
    public MeshRenderer ShapeMeshRenderer;
    public MeshCollider ShapeMeshCollider;
    public GameObject[] Checkpoints;
    public GameObject StopButton;
    public GameObject finalScreen;
    public GameObject StopInstruc;
    public GameObject BrightEffect;

    void Start()
    {
        Instructions.text = "Hello there!! Welcome to your bathroom, this mirror
is so dirty. " +
            "Could you please remove this dust??";
        for (int i = 0; i < Interactors1.Length; i++)
        {
            Interactors1[i].SetActive(false);
        }
        for (int i = 0; i < Interactors2.Length; i++)
        {
            Interactors2[i].SetActive(false);
        }
        for (int i = 0; i < Arrows.Length; i++)
        {
            Arrows[i].SetActive(false);
        }
        for (int i = 0; i < Checkpoints.Length; i++)
        {
            Checkpoints[i].SetActive(false);
        }
        HandArrows.SetActive(false);
        LevelArrows.SetActive(false);
        TimerArrow.SetActive(false);
        OneHandTutorialButton.SetActive(false);
        TwoHandTutorialButton.SetActive(false);
        TutorialTimerOneHand.SetActive(false);
        TutorialTimerBothHands.SetActive(false);
        ShapeMeshRenderer.enabled = false;
        ShapeMeshCollider.enabled = false;
        StopButton.SetActive(false);
        finalScreen.SetActive(false);
        StopInstruc.SetActive(false);
        BrightEffect.SetActive(false);
    }
    public void MakeButtonsAppear()
    {
        for (int i = 0; i < Interactors1.Length; i++)
        {
            Interactors1[i].SetActive(true);
        }
        Instructions.text = "Choose the hand you would like to use to clean the 
mirror!";
        CleanButton.SetActive(false);
        HandArrows.SetActive(true);

    }

    public void HandSelected()
    {
        for (int i = 0; i < Interactors2.Length; i++)
        {
            Interactors2[i].SetActive(true);
        }
        Instructions.text = "How fast can you be cleaning? Choose the 
difficulty!!";
        LevelArrows.SetActive(true);
        HandArrows.SetActive(false);
    }

    public void LevelSelected()
    {
        TimerArrow.SetActive(true);
        LevelArrows.SetActive(false);
        Instructions.text = "Depending on the difficulty you will have more or 
less time. " +
            "Press the PLAY button and good luck!";
    }
}
```

Button Transitions

These are the colour transition when the pointer interacts with UI buttons.

```
public class ButtonTransitioner : MonoBehaviour, IPointerEnterHandler, 
IPointerExitHandler, IPointerClickHandler, IPointerDownHandler, IPointerUpHandler
{
    public Color m_NormalColor;
    public Color m_HoverColor;
    public Color m_DownColor;
    public Color m_PressedColor;
    public Color m_DisabledColor;


    public bool selected = false;
    public bool interactable = true;

    public GlobalScript GlobalScript;

    private Image m_Image = null;
    private Button m_Button = null;
    private void Awake()
    {
        m_Image = GetComponent<Image>();
        m_Button = GetComponent<Button>();
        UpdateColor(m_NormalColor);
    }

    public void OnPointerEnter (PointerEventData eventData)
    {
        print("Enter");
        if (interactable == false) return;

        if (selected == false)
            UpdateColor(m_HoverColor);
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        print("Exit");
        if (interactable == false) return;
        if (selected == false)
            UpdateColor(m_NormalColor);
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        if (interactable == false) return;
        if (selected == false)
            UpdateColor(m_DownColor);
    }

    public void OnPointerClick(PointerEventData eventData)
    {
        if (interactable == false) return;
        print("Click");
        UpdateColor(m_PressedColor);
        SelectButton(true);
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        if (interactable == false) return;
        print("Up");
    }

    public void UpdateColor(Color newcolor)
    {
        m_Image.color = newcolor;
    }

    public void SelectButton(bool isSelected)
    {
        selected = isSelected;
    }

    public void SetInteractable (bool isEnabled)
    {
        if (isEnabled)
        {
            UpdateColor(m_NormalColor);
        }
        else
        {
            UpdateColor(m_DisabledColor);
        }
        interactable = isEnabled;
        m_Button.interactable = isEnabled;
    }
}

public class ButtonObjTransitioner : MonoBehaviour, IPointerEnterHandler,
IPointerExitHandler, IPointerClickHandler, IPointerDownHandler
{
    public Color m_NormalColor;
    public Color m_HoverColor;
    public Color m_DownColor;
    public bool selected = false;
    public Color m_PressedColor;

    private RawImage m_Rawimage = null;


    private void Awake()
    {
        m_Rawimage = GetComponent<RawImage>();
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        print("Enter");
        if (selected == false)
            UpdateColor(m_HoverColor);
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        print("Exit");
        if (selected == false)
            UpdateColor(m_NormalColor);
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        if (selected == false)
            UpdateColor(m_DownColor);

    }

    public void OnPointerClick(PointerEventData eventData)
    {
        print("Click");
        UpdateColor(m_PressedColor);
        SelectButton(true);
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        print("Up");
    }

    public void UpdateColor(Color newtexture)
    {
        m_Rawimage.color = newtexture;
    }

    public void SelectButton(bool isSelected)
    {
        selected = isSelected;
    }
}
```

Hand Selector

```
public class HandSelector : MonoBehaviour
{
    public ButtonObjTransitioner[] buttons;
    public PlayButtonController PlayButtonController;
    public MeshRenderer ShapeMeshRenderer;
    public TutorialButtonActions TutorialButtonActions;
    public GameObject[] CheckPointsGroup;
    public PlayButtonFunctions PlayButtonFunctions;
    public TwoBoneIKConstraint[] twoBoneIKConstraints;
    public GlobalScript GlobalScript;
    public FinalDataToExport FinalDataToExport;


    public void SetLevel(ButtonObjTransitioner selectedbutton)
    {
        for (int i = 0; i < buttons.Length; i++)
        {
            if (buttons[i] != selectedbutton)
            {
                buttons[i].SelectButton(false);
                buttons[i].UpdateColor(buttons[i].m_NormalColor);
            }

        }
        if (PlayButtonController.handSelected == false)
        {
            PlayButtonController.handSelected = true;
            PlayButtonController.UpdateButtonState();
            GlobalScript.HandSelected();
        }
    }

    public void SetShape(Material shapeTrail)
    {
        TutorialButtonActions.ShapeMeshRenderer.material = shapeTrail;
    }

    public void SetTutorialObj(GameObject TutorialObj)
    {
        PlayButtonFunctions.TutorialObj = TutorialObj;
    }

    public void TextSelector(string TextAfterPlay)
    {
        PlayButtonFunctions.TextAfterPlay = TextAfterPlay;
    }

    public void TutorialTimerSelector(GameObject Timer)
    {
        PlayButtonFunctions.TutorialTimer = Timer;
    }
    public void SetCheckpoints(GameObject Checkpoints)
    {
        for (int i = 0; i < CheckPointsGroup.Length; i++)
        {
            if (CheckPointsGroup[i] = Checkpoints)
            {
                TutorialButtonActions.checkpoints = Checkpoints;
            }
        }
    }

    public void SetArrows(GameObject Arrows)
    {
        TutorialButtonActions.StartArrows = Arrows;
    }

    public void SetConstraints(TwoBoneIKConstraint ArmConstraint)
    {
        for (int i = 0; i < twoBoneIKConstraints.Length; i++)
        {
            if (twoBoneIKConstraints[i] = ArmConstraint)
            {
                PlayButtonFunctions.HandSelected = ArmConstraint.name;
                PlayButtonFunctions.TwoBoneIKConstraint = ArmConstraint;
            }
        }
    }

    public void HandName(string Hand)
    {
        FinalDataToExport.Hand = Hand;
    }

}
```

Level Selector

```
public class LevelSelector : MonoBehaviour
{
    public ButtonTransitioner[] buttons;
    public PlayButtonController PlayButtonController;
    public Text Timer;
    public Timer TimerFloatValue;
    public FinalDataToExport FinalDataToExport;
    public GlobalScript GlobalScript;

    public void SetLevel(ButtonTransitioner selectedbutton)
    {
        // Determine if the button is pressed and change the levelselected bool
to enable play button
        for (int i = 0; i < buttons.Length; i++)
        {
            if (buttons[i] != selectedbutton)
            {
                buttons[i].SelectButton(false);
                buttons[i].UpdateColor(buttons[i].m_NormalColor);
            }

        }
        if(PlayButtonController.levelSelected == false)
        {
            PlayButtonController.levelSelected = true;
            PlayButtonController.UpdateButtonState();
            GlobalScript.LevelSelected();
        }
    }

    public void SetTime (string time)
    {
        Timer.text = time;
        TimerFloatValue.time = float.Parse(time)/100;
        FinalDataToExport.TimerInitValue = time;
    }
    public void LevelName(string Level)
    {
        FinalDataToExport.Level = Level;
    }

}
```

Play button controller

```
public class PlayButtonController : MonoBehaviour
{
    public bool handSelected;
    public bool levelSelected;
    public ButtonTransitioner ButtonTransitioner;

    private void Start()
    {
        UpdateButtonState();
    }

    public void UpdateButtonState()
    {
        if(handSelected && levelSelected)
        {
            ButtonTransitioner.SetInteractable(true);
        }
        else
        {
            ButtonTransitioner.SetInteractable(false);
        }
    }

}
```

Play button functions

```
public class PlayButtonFunctions : MonoBehaviour
{
    public string HandSelected;
    public GameObject ParticleSystemLeft;
    public GameObject DecalParticleSystemLeft;
    public GameObject ParticleSystemRight;
    public GameObject DecalParticleSystemRight;
    public GameObject Pointer;
    public GameObject VrInput;
    public GameObject ModelController;
    public GameObject TutorialObj;
    public GameObject[] DisableAll;
    public Text Instruction;
    public string TextAfterPlay;
    public GameObject TutorialTimer;
    public TwoBoneIKConstraint TwoBoneIKConstraint;
    public GameObject StopInstruc;


    public void ParticleSystemEnabled ()
    {
        Invoke("ParticleSystemEnabledAfterATime", 1.0f);
    }

    public void ParticleSystemEnabledAfterATime()
    {
        if(HandSelected == "Right Arm IK")
        {
            ParticleSystemLeft.SetActive(true);
            DecalParticleSystemLeft.SetActive(true);
        }
        if(HandSelected == "Left Arm IK")
        {
            ParticleSystemRight.SetActive(true);
            DecalParticleSystemRight.SetActive(true);
        }
        else
        {
            ParticleSystemLeft.SetActive(true);
            DecalParticleSystemLeft.SetActive(true);
            ParticleSystemRight.SetActive(true);
            DecalParticleSystemRight.SetActive(true);
        }

    }
    public void MakeAnObjectAppear(GameObject gameObject)
    {
        gameObject.SetActive(true);
    }

    public void TutorialObjAppear()
    {
        TutorialObj.SetActive(true);
    }

    public void TutorialTimerAppear()
    {
        TutorialTimer.SetActive(true);
    }

    public void MakePointerDisappear()
    {
        Pointer.SetActive(false);
        VrInput.SetActive(false);
        ModelController.SetActive(false);
    }

    public void DisableButtons()
    {
        for (int i = 0; i < DisableAll.Length; i++)
        {
            DisableAll[i].SetActive(false);
        }
    }

    public void DisableConstraints()
    {
        if(HandSelected != "Two Arm IK")
        {
            TwoBoneIKConstraint.weight = 0;
        }
    }

    public void DisableTimerArrow(GameObject Arrow)
    {
        Arrow.SetActive(false);
    }

    public void Instructions()
    {
        Instruction.text = TextAfterPlay;
        StopInstruc.SetActive(true);
    }
}
```

Reset

```
public class ResetScene : MonoBehaviour
{
    public void ResetOptions()
    {
        SceneManager.LoadScene("SampleScene");
    }
}
```

Quit

```
public class QuitApp : MonoBehaviour
{
    public void QuitTheGame()
    {
        SceneManager.LoadScene("QuitScene");
    }
}
```

Timers

```
public class Timer : MonoBehaviour {

    public Text timer;
    public Text crono;
    public float time;
    public bool timerGo = false;
    public FinalDataToExport FinalDataToExport;

    public float cronoTime = 0.00f;

    public void Update()
    {
        if (timerGo)
        {
            TimerDown();
            TimerUp();
        }
        else
        {
            StopTimer();
        }
    }
    public void TimerDown()
    {
        if(time > 0.00f)
        {
            time -= Time.deltaTime;
            timer.text = "" + time.ToString("f2");
            FinalDataToExport.TimeInorOut = "IN";
        }
        else
        {
            FinalDataToExport.TimeInorOut = "OUT";
        }
    }


    public void TimerUp()
    {
        cronoTime += Time.deltaTime;
        crono.text = "" + cronoTime.ToString("f2");
    }

    public void StopTimer()
    {
        timer.text = timer.text;
        crono.text = crono.text;
        FinalDataToExport.TimeTotal = crono.text.Replace(',' , '.') + "s";
        FinalDataToExport.Round = "Round:" + "" + GetRound();
    }

    public string GetRound()
    {
        return
(File.ReadAllLines(@"C:\Users\joser\Desktop\TFG\Laser\Assets\Report\report.csv").
Length + 1).ToString();
    }
}
```

Tutorial object actions

```
public class TutorialButtonActions : MonoBehaviour
{
    public Text TutorialTimer;
    public float time = 0.0f;
    public MeshRenderer ShapeMeshRenderer;
    public MeshCollider ShapeMeshCollider;
    public GameObject checkpoints, TutorialObj, TTimer, StartArrows;
    public Text Instructions;

    public void OnParticleCollision(GameObject other)
    {
        if(TutorialTimer.text != "3,0")
        {
            Debug.Log("HIIT");
            time += Time.deltaTime;
            TutorialTimer.text = "" + time.ToString("f1");
        }
        else
        {
            Debug.Log("3 seconds");
            MakeTheGameAppear();
        }
    }

    public void MakeTheGameAppear()
    {
        TutorialObj.SetActive(false);
        TTimer.SetActive(false);
        ShapeMeshRenderer.enabled = true;
        ShapeMeshCollider.enabled = true;
        checkpoints.SetActive(true);
        Instructions.text = "Point the START checkpoint with your spray and
FOLLOW the trajectory until " +
            "the END checkpoint to clean the mirror.";
        StartArrows.SetActive(true);
    }
}

public class TutorialButtonActionsBothHands : MonoBehaviour
{
    public Text TutorialTimer;
    public float time = 0.0f;
    public MeshRenderer ShapeMeshRenderer;
    public MeshCollider ShapeMeshCollider;
    public GameObject checkpoints, TutorialObj, TTimer, StartArrows;
    public Text Instructions;

    public void OnParticleCollision(GameObject other)
    {
        Debug.Log("HIIT");
        time += Time.deltaTime;
        TutorialTimer.text = "" + time.ToString("f1");

        if (TutorialTimer.text == "3,0")
        {
            Debug.Log("3 seconds");
            MakeTheGameAppear();
        }
    }

    public void MakeTheGameAppear()
    {
        TutorialObj.SetActive(false);
        TTimer.SetActive(false);
        ShapeMeshRenderer.enabled = true;
        ShapeMeshCollider.enabled = true;
        checkpoints.SetActive(true);
        Instructions.text = "Point the START checkpoint with your spray and
FOLLOW the trajectory until " +
            "the END checkpoint to clean the mirror.";
        StartArrows.SetActive(true);

    }
}
```

Checkpoints scripts

Here there are the Start and End scripts and the rest of the checkpoint's functions.

```
public class StartCollision : MonoBehaviour
{
    public Timer Timer;
    public GameObject EndArrow, StartArrow;
    public Animator Sequence;
    public bool StopSequence = false;

    public void OnParticleCollision(GameObject other)
    {
        Debug.Log("HIIIT");
        Timer.timerGo = true;
        StartAndEndArrows();
        StopSequence = true;
        SeqAction();
    }

    public void SeqAction()
    {
        if (StopSequence)
        {
            Sequence.enabled = false;
        }
    }

    public void StartAndEndArrows()
    {
        StartArrow.SetActive(false);
        EndArrow.SetActive(true);
    }
}

public class CheckPointsTransition : MonoBehaviour
{
    public Material normalMaterial;
    public Material touchedMaterial;
    public bool checkPointtouched = false;

    private MeshRenderer CapsuleMat;

    private void Awake()
    {
        CapsuleMat = GetComponent<MeshRenderer>();
    }

    public void Update()
    {
        if (checkPointtouched)
        {
            CapsuleMat.material = touchedMaterial;
        }
        else
        {
            CapsuleMat.material = normalMaterial;
        }
    }
}
```

```
public class TouchingCheckPoints : MonoBehaviour
{
    public CheckPointsTransition CheckPointsTransition;
    public string StartTouched = "";

    public void OnParticleCollision(GameObject other)
    {
        if(StartTouched == "Touched")
        {
            Debug.Log("HIIIT");
            CheckPointsTransition.checkPointtouched = true;
        }
    }
}

public class EndCollision : MonoBehaviour
{
    public Timer Timer;
    public GameObject[] DisappearedObjects;
    public GameObject[] AppearedObjects;
    public MeshRenderer ShapeGameMeshRender;
    public MeshCollider ShapeGameMeshCollider;
    public CheckPointsTransition[] CheckPointsTransition;
    public StartCollision StartCollision;
    public string StartTouched = "";
    public GameObject BrightEffect, FinalScreen;
    public Accuracy Accuracy;

    public void OnParticleCollision(GameObject other)
    {
        if(StartTouched == "Touched")
        {
            Timer.timerGo = false;
            for (int i = 0; i < DisappearedObjects.Length; i++)
            {
                DisappearedObjects[i].SetActive(false);
            }
            for (int i = 0; i < AppearedObjects.Length; i++)
            {
                AppearedObjects[i].SetActive(true);
            }
            ShapeGameMeshRender.enabled = false;
            ShapeGameMeshCollider.enabled = false;
            for (int i = 0; i < CheckPointsTransition.Length; i++)
            {
                CheckPointsTransition[i].checkPointtouched = false;
                CheckPointsTransition[i].Update();
            }
            StartCollision.Sequence.enabled = true;
        }
        Invoke("BrightEffectDesactive", 2.0f);
        Accuracy.CalculateAccuracy();
    }
    public void BrightEffectDesactive()
    {
        BrightEffect.SetActive(false);
        FinalScreen.SetActive(true);
    }
}

public class EndCollision2 : MonoBehaviour
{
    public EndCollision2Actions EndCollision2Actions;
    public string StartTouched = "";

    private void OnParticleCollision(GameObject other)
    {
        if (StartTouched == "Touched")
        {
            EndCollision2Actions.EndCollision2 = true;
            EndCollision2Actions.EndActions();
        }
    }
}

public class EndCollision2Actions : MonoBehaviour
{
    public bool EndCollision2 = false;
    public bool EndCollision3 = false;
    public MeshRenderer ShapeGameMeshRender;
    public MeshCollider ShapeGameMeshCollider;
    public Timer Timer;
    public GameObject[] Checkpoints, EndArrows;
    public GameObject[] DisappearedObjects;
    public GameObject[] AppearedObjects;
    public CheckPointsTransition[] CheckPointsTransition;
    public StartCollision StartCollisionL, StartCollisionR;
    public GameObject BrightEffect, FinalScreen;
    public Accuracy Accuracy;

    public void EndActions()
    {
        if(EndCollision2 && EndCollision3)
        {
            Timer.timerGo = false;
            ShapeGameMeshRender.enabled = false;
            ShapeGameMeshCollider.enabled = false;
            StartCollisionL.Sequence.enabled = true;
            StartCollisionR.Sequence.enabled = true;

            for (int i = 0; i < DisappearedObjects.Length; i++)
            {
                DisappearedObjects[i].SetActive(false);
            }
            for (int i = 0; i < AppearedObjects.Length; i++)
            {
                AppearedObjects[i].SetActive(true);
            }
            for (int i = 0; i < CheckPointsTransition.Length; i++)
            {
                CheckPointsTransition[i].checkPointtouched = false;
                CheckPointsTransition[i].Update();
            }
            Invoke("BrightEffectDesactive", 2.0f);
            Accuracy.CalculateAccuracy();
        }
        if (EndCollision2)
        {
            Checkpoints[0].SetActive(false);
            EndArrows[0].SetActive(false);
        }
        if (EndCollision3)
        {
            Checkpoints[1].SetActive(false);
            EndArrows[1].SetActive(false);
        }
    }
    public void BrightEffectDesactive()
    {
        BrightEffect.SetActive(false);
        FinalScreen.SetActive(true);
    }
}

public class EndCollision3 : MonoBehaviour
{
    public EndCollision2Actions EndCollision2Actions;
    public string StartTouched = "";

    private void OnParticleCollision(GameObject other)
    {
        if (StartTouched == "Touched")
        {
            EndCollision2Actions.EndCollision3 = true;
            EndCollision2Actions.EndActions();
        }
    }
}
```

Accuracy

```
public class Accuracy : MonoBehaviour
{
    public float range;
    public GameObject Group3Spray;
    public float HitTime = 0;
    public float TotalTime = 0;
    public FinalDataToExport FinalDataToExport;
    void Update()
    {
        bool state =
SteamVR_Actions.default_GrabPinch[SteamVR_Input_Sources.Any].state;

        if (state == true)
        {
            RaycastHit hit;
            if
(Physics.Raycast(Group3Spray.transform.position,Group3Spray.transform.forward,
out hit, range))
            {
                Texture2D texture =
hit.collider.gameObject.GetComponent<Renderer>().material.mainTexture as
Texture2D;
                if (texture != null)
                {
                    if (texture.isReadable)
                    {
                        Vector2 Coordenates = hit.textureCoord;
                        Vector2 pixelUV = new Vector2(Coordenates.x *
texture.width, Coordenates.y * texture.height);
                        Color collisionColor = texture.GetPixel((int)pixelUV.x,
(int)pixelUV.y);
                        TotalTime += Time.deltaTime;
                        if(collisionColor.a != 0)
                        {
                            HitTime += Time.deltaTime;
                        }
                    }
                }
            }
        }
    }

    public void CalculateAccuracy()
    {
        float Accuracy = (HitTime / TotalTime) * 100;
        FinalDataToExport.AccuracyPerCent = String.Format("{0:0.00}",
Accuracy).Replace(',', '.');

        if(Accuracy < 50)
        {
            FinalDataToExport.AccuracyResult = "Bad";
        }
        else if (Accuracy < 80)
        {
            FinalDataToExport.AccuracyResult = "Good";
        }
        else if (Accuracy < 100)
        {
            FinalDataToExport.AccuracyResult = "Perfect";
        }
        else
        {
            FinalDataToExport.AccuracyResult = "Excelent";
        }
    }

    private void OnDrawGizmos()
    {
        Gizmos.color = Color.red;
        Gizmos.DrawRay(Group3Spray.transform.position,
Group3Spray.transform.forward * range);
    }
}
```

Final Scene

```
public class FinalDataToExport : MonoBehaviour
{
    public string Round, Hand, Level, TimeInorOut, TimeTotal, AccuracyPerCent,
        AccuracyResult = "", Resume = "";
    public PlayButtonFunctions PlayButtonFunctions;
    public GameObject[] AppearedObjects;
    public GameObject[] DisappearedObjects;
    public string TimerInitValue;
    public Text[] TimersToReset;
    public Timer Timer;
    public TutorialButtonActions TutorialButtonActions;
    public TutorialButtonActionsBothHands[] TutorialButtonActionsBothHands;
    public EndCollision[] EndCollision;
    public EndCollision2 EndCollision2;
    public EndCollision3 EndCollision3;
    public TouchingCheckPoints[] TouchingCheckPoints;
    public EndCollision2Actions EndCollision2Actions;


    public void DEV_AppendToReport()
    {
        CSVManager.AppendToReport(
            new string[8]
            {Round, Hand, Level, TimeInorOut, TimeTotal, AccuracyPerCent, 
AccuracyResult, Resume}
            );
        EditorApplication.Beep();
    }

    public void DEV_ResetToReport()
    {
        CSVManager.CreateReport();
        EditorApplication.Beep();
    }

    public void PlayAgain()
    {
        EndCollision2Actions.EndCollision2 = false;
        EndCollision2Actions.EndCollision3 = false;
        PlayButtonFunctions.TutorialObj.SetActive(true);
        PlayButtonFunctions.TutorialTimer.SetActive(true);
        Timer.cronoTime = 0.00f;

        for (int i = 0; i < AppearedObjects.Length; i++)
        {
            AppearedObjects[i].SetActive(true);
        }
        for (int i = 0; i < DisappearedObjects.Length; i++)
        {
            DisappearedObjects[i].SetActive(false);
        }
        PlayButtonFunctions.ParticleSystemEnabled();
        for (int i = 0; i < EndCollision.Length; i++)
        {
            EndCollision[i].StartTouched = "";
        }
        EndCollision2.StartTouched = "";
        EndCollision3.StartTouched = "";
        for (int i = 0; i < TouchingCheckPoints.Length; i++)
        {
            TouchingCheckPoints[i].StartTouched = "";
        }
    }

    public void ResetTimers()
    {
        for (int i = 0; i < TimersToReset.Length; i++)
        {
            TimersToReset[0].text = "0,00";
            TimersToReset[1].text = "0";
            TimersToReset[2].text = "0";
            TimersToReset[3].text = "0";
        }
        TutorialButtonActions.time = 0.0f;
        for (int i = 0; i < TutorialButtonActionsBothHands.Length; i++)
        {
            TutorialButtonActionsBothHands[i].time = 0.0f;
        }
    }

    public void TimerValue(Text timer)
    {
        timer.text = TimerInitValue;
        Timer.time = float.Parse(TimerInitValue) / 100;
    }

    public void InstructionsAgain(Text Instruc)
    {
        Instruc.text = PlayButtonFunctions.TextAfterPlay;
    }
}
```

Stop Button

```
public class StopButtonActions : MonoBehaviour
{
    private void OnParticleCollision(GameObject other)
    {
        SceneManager.LoadScene("SampleScene");
    }
}
```

Clear Button

```
public class ClearItButton : MonoBehaviour, IPointerClickHandler
{
    public GlobalScript GlobalScript;

    public void OnPointerClick(PointerEventData eventData)
    {
        GlobalScript.MakeButtonsAppear();
    }
}
```