



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

FINAL DEGREE PROJECT

Biomedical Engineering Degree

Materials Engineering Degree

VIRTUAL REALITY APPLICATION FOR REHABILITATION

FLYING MODULE



Project Report and Annex

Author: Marta Picazo Cot ^{*1}
Aitor Rodríguez González ^{*2}
Director: Francisco Alpiste Penalba
Co-Director: Jordi Torner
Announcement: June 2020

Resum

Aquest projecte té com a objectiu desenvolupar una aplicació de Realitat Virtual funcional per a la rehabilitació i el benestar de la gent gran en residències d'avis. El que es pretén és complementar el tractament de rehabilitació dels pacients mitjançant jocs de Realitat Virtual que els permetin realitzar moviments repetitius mentre es mouen per un entorn natural immersiu.

L'empresa *Visyon360* ha proporcionat l'equipament necessari per al projecte i el Dr. Stavros, CTO de *Senopi*, ha establert els objectius a assolir.

Aquest projecte neix d'una idea sorgida d'anteriors projectes, desenvolupats per detectar malalties cognitives. En aquells projectes els pacients es trobaven immersos en un entorn virtual natural en el qual recorrien un camí guiat ja sigui volant, corrent o nedant. Aquelles experiències van tenir lloc mentre el pacient estava en un escàner de ressonància magnètica (RM), combinant les imatges per ressonància magnètica a temps real amb la Realitat Virtual. La resposta cognitiva del pacient durant el joc regulava la seva velocitat.

Així doncs, aquest projecte sorgeix dels anteriors projectes amb l'objectiu d'aprofitar el mateix entorn natural i oferir al pacient la possibilitat de moure's per ell amb llibertat. Aleshores, l'objectiu principal de l'aplicació de rehabilitació és animar a la gent gran a realitzar alguns moviments requerits relacionats amb la marxa o la natació, entre d'altres, depenent del mòdul en el que estiguin participant. Aquests moviments repetitius es capturen a través d'un sistema de Realitat Virtual per tal de provocar el moviment de l'avatar al món virtual.

Aquest projecte està centrat en els moviments de vol, ja que es tracta del mòdul de vol en primera persona que forma part del projecte de rehabilitació. Així, mitjançant aquest mòdul, es motiva als pacients a realitzar moviments repetitius de les extremitats superiors relacionats amb el moviment de vol, que és similar al dels ocells.

La següent memòria descriu les diferents eines emprades per desenvolupar l'aplicació, així com l'ús de cadascuna d'elles i la metodologia seguida.

També s'han hagut de trobar algunes alternatives a l'equipament de realitat virtual per poder continuar amb el desenvolupament del projecte a causa de la pandèmia COVID-19.

Finalment, s'han proposat algunes millores futures i s'ha conclòs que el mòdul de vol pot funcionar amb Realitat Virtual tot i no haver-lo pogut provar amb els equips de Realitat Virtual adequats.

Resumen

Este proyecto tiene como objetivo desarrollar una aplicación de Realidad Virtual funcional para la rehabilitación y el bienestar de las personas mayores en residencias de ancianos. Lo que se pretende es complementar el tratamiento de rehabilitación de los pacientes mediante juegos de Realidad Virtual que les permitan realizar movimientos repetitivos mientras se mueven por un entorno natural inmersivo.

La empresa *Visyon360* ha proporcionado el equipamiento necesario para el proyecto y el Dr. Stavros, CTO de *Senopi*, ha establecido los objetivos a alcanzar.

Este proyecto nace de una idea surgida de anteriores proyectos, desarrollados para detectar enfermedades cognitivas. En esos proyectos los pacientes se encontraban inmersos en un entorno virtual natural en el que recorrían un camino guiado ya sea volando, corriendo o nadando. Esas experiencias tuvieron lugar mientras el paciente estaba en un escáner de resonancia magnética (RM), combinando imágenes de MRI en tiempo real con Realidad Virtual. La respuesta cognitiva del paciente durante el juego regulaba su velocidad.

Así pues, este proyecto surgió de los anteriores con el objetivo de aprovechar el mismo entorno natural y ofrecer al paciente la posibilidad de moverse por él con libertad. Entonces, el objetivo principal de la aplicación de rehabilitación es animar a la gente mayor a realizar algunos movimientos requeridos relacionados con la marcha o la natación, entre otros, dependiendo del módulo en el que estén participando. Estos movimientos repetitivos se capturan a través de un sistema de Realidad Virtual para provocar el movimiento del avatar en el mundo virtual.

Este proyecto está centrado en los movimientos de vuelo, ya que se trata del módulo de vuelo en primera persona que forma parte del proyecto de rehabilitación. Así, mediante este módulo, se motiva a los pacientes a realizar movimientos repetitivos de las extremidades superiores relacionados con el movimiento de vuelo, que es similar al de los pájaros.

La siguiente memoria describe las diferentes herramientas utilizadas para desarrollar la aplicación, así como el uso de cada una de ellas y la metodología seguida.

También se han tenido que encontrar algunas alternativas al equipamiento de Realidad Virtual para poder continuar con el desarrollo del proyecto a causa de la pandemia COVID-19.

Finalmente, se han propuesto algunas mejoras futuras y se ha concluido que el módulo de vuelo puede funcionar con Realidad Virtual, aunque no se ha podido probar con los equipos de Realidad Virtual adecuados.

Abstract

This project aims to develop a functional Virtual Reality application for the rehabilitation and well-being of older adults in nursing homes. It intends to engage the patient in the rehabilitation treatment by means of Virtual Reality games. To play these games, the patient must execute repetitive movements while moving around in an immersive natural environment.

The company *Visyon360* provided the necessary equipment for the project, and *Senopi's* CTO, Dr Stavros, established the goals to achieve.

The idea of this project was born from previous projects developed to detect cognitive illnesses. In those projects, patients were immersed in a virtual natural environment and went through a guided path either running, flying or swimming. Those experiences took place while the patient was in a magnetic resonance imaging (MRI) scanner, combining real-time MRI images with Virtual Reality. The cognitive response of the patient during the game regulated his speed.

Therefore, the idea of this project emerged from these previous projects with the aim to take advantage of the same natural environment, but also offering the patient the possibility to move around freely. Thus, the main objective of the rehabilitation application is to make the elderly do some required movements related to walk or swim, among others, depending on the module in which they are participating. These repetitive movements are captured by a Virtual Reality system to cause an avatar's motion in the virtual world.

This project is focused on flying movements, as it is the first-person flight module that belongs to the rehabilitation project. Thus, by using this module, patients are encouraged to perform repetitive upper limb movements related to a flying movement similar to that of birds.

The following report describes the different tools employed to develop the application as well as how they have been used and the methodology applied.

Due to the COVID-19 pandemic, some alternatives to the Virtual Reality equipment had to be found to carry on the development of the project.

Finally, future improvements have been proposed and it has been concluded that the first-person flight module is functional with Virtual Reality, even though no tests could be made with the appropriate Virtual Reality equipment.



Acknowledgments

During the project implementation, we have received the help and collaboration of various people. In this section, we would like to thank them all for their invaluable assistance.

First of all, we would like to thank our directors, Jordi Torner and Francesc Alpiste, for providing the necessary tools and information to carry out the project. We thank them, as well as Dr Stavros, for their useful ideas on how to develop the project without the supposed technology.

We want to express our gratitude for Oriol Perez, who has helped us clarify some doubts about programming and suggested some ideas to improve the script development. He has also tested the application to look for major bugs.

Then, we would like to thank Roser Cot. She has helped us improve the understanding of the whole report, by checking grammar, punctuation and spelling.

Last but not least, we would like to thank our family and friends for all the support given throughout the project. They know better than anyone else the effort made by the developers.



Figure List

Figure 3.1. Charles Wheatstones' stereoscope on the left and Morton Heilig's Sensorama on the right (12)	7
Figure 3.2. Headsight on the left and Sutherland's Sword of Damocles on the right (12)	7
Figure 3.3. NASA's VIEW (12) on the left and Nintendo's Virtual Boy on the right (14)	8
Figure 3.4. Oculus Rift HMD (15) on the left and HTC Vive HMD on the right (16)	9
Figure 3.5. Example of Immersive VR System (17)	10
Figure 3.6. Example of Semi-Immersive VR System (18)	10
Figure 3.7. Example of Non-Immersive VR System (19)	11
Figure 3.8. Simulation of an oil and gas Industry (20)	11
Figure 3.9. Museum of Monte San Michele. "Great War Told Through VR" (21)	12
Figure 3.10. Visualisation of Scientific Data on the HTC Vive (22)	12
Figure 3.11. Virtual Reality for Seniors Experiencing Nature (3)	13
Figure 4.1. Gantt Diagram	19
Figure 5.1. Degrees of Freedom (38)	21
Figure 5.2. Lighthouses Position in the Play Area (37)	22
Figure 5.3. Two-dimensional Infrared Laser Beam (40)	22
Figure 5.4. Controller Input	24
Figure 5.5. HTC Vive Components (37)	25

Figure 5.6. HTC Vive's Components Status	26
Figure 6.1. Unity Hub	27
Figure 6.2. Unity Interface	28
Figure 6.3. Virtual Reality Supported Settings	30
Figure 6.4. SteamVR Prefabs in Hierarchy Window	31
Figure 7.1. External Script Editor Settings	32
Figure 8.1. Initial Set of Movements	34
Figure 8.2. Human Glenohumeral Joint Movements (50)	35
Figure 8.3. Human Glenohumeral Joint Movements (50)	36
Figure 8.4. Human Wrist Joint Movements (50)	36
Figure 8.5. Camera Rig from Steam VR	37
Figure 8.6. How to Import Animation Rigging	37
Figure 8.7. Neck Motion Range (51)	39
Figure 8.8. VRRig Script in Unity's Inspector	40
Figure 8.9. Each Bone Axis (Upper-side Picture). Left-hand and Right-hand Position Vectors (Bottom-side picture).	41
Figure 8.10. Right Limb Angles Calibration Equation: $y = -0,0001x^3 + 0,0064x^2 + 2,0394x - 4,533$. Left Limb Angles Calibration Equation: $y = 0,0065x^3 + 0,004x^2 + 1,1976x + 2,0219$.	42
Figure 8.11. <i>Mesh Collider</i> Interface	43
Figure 8.12. Limits Placement in Unity Interface	43

Figure 8.13. Limit's Tags _____	44
Figure 8.14. Animations Imported from "3rd Person Controller + Fly Mode" Asset in Unity _____	47
Figure 8.15. Creating an Avatar from a Character's Model _____	48
Figure 8.16. Animator Component _____	48
Figure 8.17. Unity's Animation Window _____	49
Figure 8.18. Unity's Animator Window _____	50
Figure 8.19. Motion of the Avatar in Unity _____	52
Figure 8.20. Xbox One Controller Mapping for Unity _____	55
Figure 8.21. PlayStation 4 Controller Mapping for Unity _____	55
Figure 8.22. Controller Script on Left Target _____	56
Figure 9.1. Use Case Diagram Basic Elements _____	57
Figure 9.2. Use Case Diagram _____	59
Figure 9.3. Adapted Use Case Diagram _____	60
Figure 10.1. The WEEE Symbol _____	68
Figure 11.1. Minimap of the World in Warcraft Game _____	69
Figure A1.1. Two Bone IK Constraint Interface _____	79
Figure A1.2. a) Target. b) Hint. _____	80
Figure A1.3. Multi-Parent Constraint Interface _____	81
Figure A1.4. Rig Hierarchy Scheme _____	82

Figure A1.5. Rig Layer Interface within Rig Builder _____	82
Figure A1.6. Rig Component Interface _____	82
Figure A1.7. Skeleton of the Avatar by Bone Renderer Component _____	83
Figure A1.8. Bone Renderer Component Interface _____	83
Figure A2.1. Angles Display _____	85
Figure A3.1. Unity's Input Manager _____	88
Figure A3.2. Left Joystick Vertical Axis Implementation in Unity's Input Manager _____	89
Figure A3.3. PlayStation 4 Right Joystick Horizontal Axis Implementation in Unity's Input Manager	89
Figure A3.4. Xbox LB Button Implementation in Unity's Input Manager _____	89

Table List

Table 4.1. Gantt Diagram Tasks _____	20
Table 5.1. HTC Vive Headset Specifications (33) _____	23
Table 5.2. HTC Vive Controller Specifications (33) _____	24
Table 5.3. HTC Vive Minimum System Requirements (33) _____	25
Table 6.1. Unity 2019.3 Minimum System Requirements (32) _____	29
Table 7.1. Microsoft Visual Studio Minimum System Requirements (41) _____	33
Table 8.1. Animation Rigging Scripts Components _____	38
Table 8.2. Upper Limbs Angles Calculations Results. α = Expected Angle. α' = Current Angle in Unity. _____	41
Table 8.3. Limits Requirement for Each Move _____	45
Table 8.4. Conditions for Animations _____	50
Table 8.5. Flow Diagram Basic Symbols _____	51
Table 8.6. RiftCat Requirements (44) _____	54
Table 9.1. Basic Communication Relationships in Use Case Diagrams _____	58
Table 9.2. UC1 Specification _____	61
Table 9.3. UC2 Specification _____	62
Table 9.4. UC3 Specification _____	62
Table 9.5. UC4 Specification _____	62

Table 9.6. Varied UC1 Specification _____	63
Table 9.7. Varied UC2 Specification _____	63
Table 9.8. Varied UC3 Specification _____	64
Table 9.9. Varied UC4 Specification _____	64
Table 9.10. Varied UC5 Specification _____	65
Table 9.11. Varied UC6 Specification _____	65
Table 9.12. Varied UC7 Specification _____	66
Table B.1. Salary and S.S Direct Costs _____	72
Table B.2. Costs Related to Equipment _____	72
Table B.3. Installation Costs _____	73
Table B.4. Total Costs _____	73
Table A2.1. X, Y Position Results from Angle Definition _____	85

Glossary

HMD – Head Mounted Display

LCD - Liquid-Crystal Display

3D – Three Dimensions

3DoF – Three Degrees of Freedom

6DoF – Six Degrees of Freedom

USB – Universal Serial Bus

HDMI - High Definition Multimedia Interface

IR – Infrared

SDK – Software Development Kit

CPU – Central Processing Unit

GPU – Graphics Processing Unit

SSD – Solid State Drive

VR – Virtual Reality

API - Application Programming Interface

GameObject - The principal object in Unity scenes, which can represent characters, scenery, cameras, and more. A GameObject's functionality is defined by the components attached to it.

Component - A functional part of a GameObject. GameObjects can contain various components.

Script - A piece of programming code that allows to create and modify components, *Trigger* game events, and respond to user input as defined.

Asset – An item that can be used during a project. It can be a 3D model, an audio file, an image file or any other kind of file supported by the software.

Trigger - A 3D volume, which is invisible to the player and that allows the developer to establish colliding zones. It returns a Boolean status to allow programming actions or events.

Collider - A 3D volume, which is invisible to the player and that allows the developer to create invisible walls and limit spaces, where player can only collide, but not go through.

C# - A programming language similar to Python or C++. It is specially used in game design or Windows desktop applications.

CTO - Chief Technology Officer.

Patient/ User/ Player – The person who is going to use this application in the final stage of the project.

Index

ABSTRACT	III
RESUM	I
RESUMEN	II
ACKNOWLEDGMENTS	III
FIGURE LIST	VII
TABLE LIST	XI
GLOSSARY	XIII
INDEX	XV
1. PREFACE	1
1.1. Origin of the Project	1
1.2. Motivation	1
1.3. Previous Requirements	2
2. INTRODUCTION	3
2.1. Objectives	3
2.2. Scope	4
3. STATE-OF-ART	5
3.1. Virtual Reality	5
3.1.1. How VR Works	5
3.1.2. History	6
3.2. Types of Virtual Reality	9
3.3. Virtual Reality Applications	11
3.4. Virtual Reality Applications for Rehabilitation and Well-being of Older Adults ...	13
4. DEFINITION OF THE PROJECT	17
4.1. Scale of the Project	17
4.2. Project Planification and Methodology	17
4.3. Gantt Diagram	19
5. HTC VIVE	21

5.1.	HTC Vive tracking System	22
5.2.	HTC Vive Hardware Characteristics	23
5.3.	HTC Vive System Requirements	25
5.4.	Configuration of HTC Vive with SteamVR	26
6.	UNITY	27
6.1.	Unity Interface	28
6.2.	Unity System Requirements	29
6.3.	Unity Connections to HTC Vive.....	29
7.	MICROSOFT VISUAL STUDIO	32
7.1.	Microsoft Visual Studio Link to Unity	32
7.2.	Microsoft Visual Studio Requirements.....	33
8.	APPLICATION DESIGN	34
8.1.	Set of Movements	34
8.1.1.	Movement Analysis	35
8.2.	Kinematics into Movement Tracking	36
8.2.1.	Linking VR with Avatar	38
8.3.	Movement Detection in Unity.....	40
8.3.1.	Technical Constraints	40
8.3.2.	Detection Methodology	42
8.4.	Avatar Animation.....	45
8.4.1.	Animations in Unity	45
8.4.2.	Animations in the Project.....	46
8.5.	Movement Translation in Unity	51
8.6.	Alternative Tools to Design in Virtual Reality.....	53
8.6.1.	Simulate HTC Vive Headset	53
8.6.2.	Simulate Virtual Reality with Controllers	54
9.	APPLICATION PERFORMANCE	57
9.1.	Use of Case Diagram	57
9.2.	Use of Case Specification.....	60
9.2.1.	VR First-person Flight Module.....	61
9.2.2.	Adapted VR First-person Flight Module.....	63
10.	ENVIRONMENTAL IMPACT ANALYSIS	67
11.	IMPROVEMENT PROPOSALS AND FUTURE APPLICATIONS	69

CONCLUSIONS	71
BUDGET	72
Direct Cost	72
Indirect Cost	73
Total Cost	73
BIBLIOGRAPHY	75
ANNEX	79
A1. Animation Rigging Package	79
A1.1. Two Bone IK Constraint	79
A1.2. Multi-Parent Constraint.....	80
A1.3. Rig	81
A1.4. Rig Builder	82
A1.5. Bone Renderer	83
A2. Angle Calculations	85
A3. Alternative to HTC Vive Controllers	88
A4. Immersion	90
A4.1. Character Conditions	90
A4.2. Audio Files.....	90
A5. Scripts.....	91
A5.1. Pose.....	91
A5.2. FlyMovement.....	94
A5.3. Controller	108

1. Preface

In recent years, interest in Virtual Reality (VR) has undoubtedly increased. Although it is known that VR has been used mainly for entertainment and research purposes, it has lately been implemented in many other fields such as medicine, training simulation or design, where it has contributed with essential benefits. VR provides a simple, intuitive way to interact with a virtual environment.

One of the areas of application is health, and more specifically, physical rehabilitation and mental health. VR systems were first studied as powerful tools in rehabilitation, being a motivating element for patients. Treatment and assessment with Virtual Reality have proved to be successful in fields such as motor and cognitive rehabilitation (1)(2).

1.1. Origin of the Project

This project was inspired by the work of a Swiss digital health startup called *Senopi* (3). Using Virtual Reality experiences, *Senopi* promotes physical and mental health in nursing homes. *Senopi* aims at improving seniors' quality of life by using VR headsets that provide evidence-based therapeutic activities to make them live stimulating experiences.

In this context, UPC EEBE School is developing a VR computer application in line with *Senopi*'s goal and values. Therefore, this project is intended for implementation in nursing homes to improve seniors' health. The VR application developed consists of a first-person flight module in a natural environment that seeks to activate the elderly by making them carry out repetitive upper limb movements in order to live an immersive VR flight experience.

1.2. Motivation

The motivation to carry out this project arises from the possibility of using innovative technology such as Virtual Reality to develop an application that is expected to benefit patients in nursing homes. For this reason, the motivation is based on two main points:

- **Medical motivation:**

The purpose of the application that will be developed in the course of this project is its use by seniors, as mentioned before.

In the future, this software will be used in rehabilitation activities by immersing the patient in a Virtual Reality environment to benefit from these activities. It will bring benefits such as distraction, getting out of monotony and entertainment.

So far, it has been experimentally shown that partakers in VR experiences applied to medical procedures reduced levels of pain and distress (4)(5).

In the near future, VR will have a significant impact on medical specialty areas such as pain management and rehabilitation. The main reasons are that this tool is becoming more and more affordable and that it can adapt to any type of activity. VR can be integrated to perform routine medical procedures such as physical therapy, pain rehabilitation or the treatment of a variety of psychiatric illnesses, such as anxiety (6).

The ability to move patients to a different entertaining reality in order to perform healthcare activities makes VR a powerful tool with high potential.

- **Personal motivation:**

From a personal point of view, VR is a technology that the developers of this project have not used before and for which they have received no training. However, it is a revolutionary tool in full development that will be crucial in the near future. That is the reason why they are motivated to study VR technology more closely - to take part in one of the many benefits it brings to society.

It is quite a challenge for the developers to start from scratch in this area, but they believe that the goal of the project is essential, both for the nursing homes that will implement this technology and for their professional development.

1.3. Previous Requirements

This project involves the creation of a Virtual Reality application. Here below, there are quoted the previous knowledge and tools necessary for its development:

- Prior experience: basic knowledge in C# programming language and in the use of the Unity3D platform for the creation of video games. It is also necessary to take into account human biomechanics in 2D and 3D for the proper development of a first-person application. Last but not least, it is important to be up to date on previous projects for proper development.
- Tools:
 - HTC Vive hardware with HMD and two controllers
 - Unity engine
 - Visual Studio software
 - SteamVR software

2. Introduction

The following end-of-degree project focusses on seniors' health. It creates an immersive Virtual Reality experience to entertain seniors while forcing them to undertake a certain degree of mobility within their capacities.

Rehabilitation through Virtual Reality tools is an emerging technology that helps promote physical activity. Recently there is a belief that this technology can to increase physical activity among the elderly.

These tools allow combining the emotional force of a natural (virtual) outdoor environment and physical activity indoors. By creating an immersive presence in a virtual environment, the user can perform rehabilitation exercises while having a positive distraction or entertainment. This new possibility may imply a change of mindset for seniors. It may change the negative way many of them see physical activity, either because it involves difficulty or because it is annoying.

The virtual environment (nature and effect of sounds) plays an important role and should create comfort in the patient. Studies show that these tools can calm and distract patients, relieving the feeling of pain or stress. Researchers believe it may even help fight depression or high blood pressure (7).

While some aspects of VR benefits for older adults still have to be explored, there are studies (8)(9) that show great potential in combining game-like exercises with the multimodal sensorial stimulation that Virtual Reality entails.

2.1. Objectives

The main goal of the project is to create an immersive Virtual Reality application in an attractive environmental framework. The users of the app -seniors living in nursing homes- should be able to perform physical activity, according to their physical capabilities, while using this distracting and motivating tool.

Since the application is addressed to the elderly, it is crucial to be especially careful when designing the app as far as movements are concerned. It must be functional.

Another point is to implement the HTC Vive Virtual Reality system (HMD and controllers) in order to control the movement of an avatar in a Virtual Reality environment and make it appear as natural as

possible on the screen. This movement to be detected is that of the user who is running the first-person flight module; so that the user sees his movement in VR.

Therefore, at the beginning of the project, the main objectives are defined as follows:

- Take advantage of the natural environment and sounds in the application to create an immersive and satisfying experience
- Define simple movements that force the user to move the upper limbs to be able to move around in the environment
- Play the user's movements in the avatar in real time to control the movement
- Offer the user a top view of the environment
- Start the flight with ten per cent of the maximum speed and at a certain height
- Initiate the application with the avatar at ground height
- Establish acceleration and deceleration mechanisms to create a sense of speed

2.2. Scope

In recent years, several students have been working on VR experiences, such as VR applications for the rehabilitation of upper and lower limbs or a VR application for the prevention of Alzheimer's disease. For the development of these previous applications, a natural island environment was created. The same environment is used for this project.

This project is the flight module of a future VR application that will include different modules, such as walking, dancing or swimming, in order to activate different parts of the body and enhance physical activity more enjoyably, allowing patients to feel themselves in a natural environment rather than the usual clinical environment.

Later on, the application will surely allow caregivers at nursing homes to keep a record of patients' exercise.

3. State-of-Art

3.1. Virtual Reality

Virtual Reality is an interactive and immersive experience in a three-dimensional computer-simulated environment that gives the user the illusion of participation in a synthetic environment, instead of the one they are actually in.

Nowadays, most video games have developed the technology to place the user in an interactive world: in a first-person shooter or the driver's seat of a car. However, the user is a simple spectator overseeing the events that are happening in that world and his perception of reality is not altered.

It is necessary to create an immersive experience for our brain to perceive a virtual environment. While there are different display methods, one of the most popular ways to experience VR is through a headset. Headset devices use a stereoscopic display to make what users see three-dimensional and to give depth to the image they are looking at. However, a stereoscopic display does not make an immersive experience. It is the ability to track the user's motion, particularly their head and eye movements, that allows the image displayed in the headset to change user's perspective (10).

Besides vision, certain VR experiences will also include other sensory feedback to stimulate the user; like for instance sound feedback or even tactile feedback for touch. Finally, in order to truly alter the perception of our reality there has to be a certain level of virtual interactivity to allow a certain degree of user-controlled navigation (walk or turn around in space within the environment). When the user is able to freely move within that environment and even interact with things in it, the brain can truly perceive that VR world as real, and thus Virtual Reality.

Summarizing, a VR system must include the following basic elements: a virtual world, immersion, sensory stimulation and interactivity.

3.1.1. How VR Works

The purpose of Virtual Reality technologies is to convince the user's brain to accept a virtual environment as reality. VR plays with our senses through simulate settings and environments realistically enough to transport us to any world that we can imagine.

Our human brains interpret what we see and develop a mental picture. We need to know that human's perception of reality is based on patterns developed from our own experience. These patterns increase the brain's efficiency. For example, we know where light comes from due to shades.

VR developers use those patterns to create virtual environments that provide the same type of information to the brain. Relevant mental expectations such as the laws of physics, textures, etc play an essential role in building up reality. If these expectations are not achieved, the user may experience disorientation or motion sickness.

When developers create an effective virtual environment, the result is an experience that we interpret as real (11).

3.1.2. History

The origin of Virtual Reality goes back to 1838, when Charles Wheatstone investigated how the human brain processes images (11). He discovered that viewing two side-by-side stereoscopic images through a stereoscope gave the user a sense of depth and immersion. Wheatstone's device used a pair of mirrors at 45-degree angles to the user's eyes, each reflecting a picture located off to the side. About ten years later, in 1839, David Brewster further refined this design and helped develop a more portable option dubbed the Lenticular Stereoscope.

In 1929 Edward Link created the "Link Trainer", a flight simulator which could be considered the very first form of electronic VR device.

Ten years later, in 1939 the View-master was patented. It was a simple device that designed from the innovations of the 1800's stereoscope. The View-master used thin cardboard discs containing seven stereoscopic 3D pairs of small colour photographs on film. The user could then flip through the images bringing him to a new destination each time. It was often used for virtual tourism but was then more commonly adopted as a toy. These devices set up the path for innovation and experimentation that has led to today's technology.

In 1960 appeared the first head mounted display or HMD. It was developed and patented by cinematographer Morton Heilig. The Telesphere Mask provided users with a stereoscopic 3d experience with wide vision and stereo sound, albeit for non-interactive film.

Heilig also developed The Sensorama, a multi-sensory simulator using scent producers, stereo speakers, oscillating fans, a rotating chair and a stereoscopic 3D screen (12).

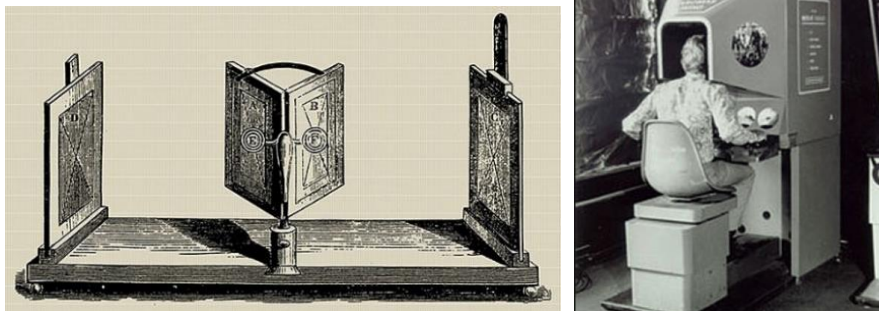


Figure 3.1. Charles Wheatstones' stereoscope on the left and Morton Heilig's Sensorama on the right (12)

In 1961 engineers at Philco Corporation introduced the Headsight, which is often considered the first true precursor to the HMD that we see today. It incorporated a video screen for each eye and a magnetic motion tracking system which was linked to a closed-circuit camera. It was a close approach to a Virtual Reality system, but it was not interactive.

In 1968 Ivan Sutherland and Bob Sprout created the first VR head mounted display that was connected to a computer and not a camera. They called it Sword of Damocles, and it was bulky and heavy to simile. This massive device hung from the lab ceiling and allowed users to explore rudimentary wireframe environments and correctly reflected the user's tracked viewing position (13).

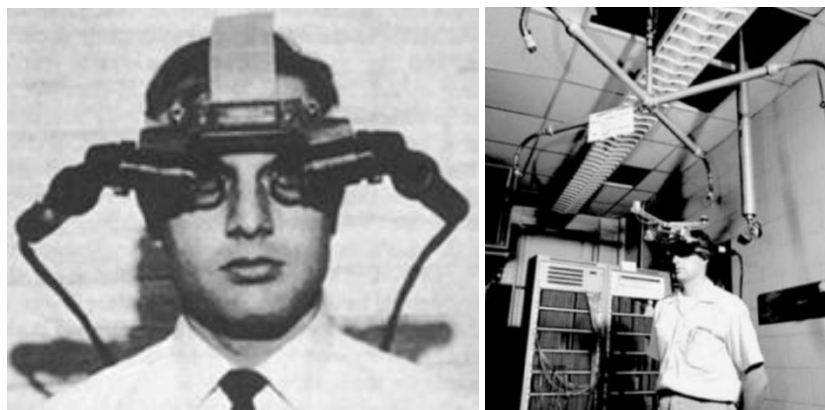


Figure 3.2. Headsight on the left and Sutherland's Sword of Damocles on the right (12)

Virtual Reality took a backseat for the next 20 years as personal computers were the upcoming stars of the technology world, but the first considered VR video game was released in 1980, using wireframe graphics and first-person control of a tank in a 3D world by looking to the arcade game.

In the mid-1980 the NASA Ames Research Center created a system that combined HMD with gloves for a haptic interaction. But it was in 1987 when Jaron Lanier and his company VPL coined that term that we use today: Virtual Reality. They began researching and developing an array of Virtual Reality

gear. Most notably they introduced the world to the data glove, a precursor to the infamous Nintendo Power glove. These devices were the first commercially available VR devices and cost nine thousand dollars each in 1989.

In 1991 virtual arcade machines were causing sensation on shopping malls, but the idea of having one in someone's home was far away. However big companies were taking notice of this VR craze and they wanted their piece of the pie.

In 1993 Sega announced the Sega VR headset. This feature came equipped with head tracking, stereo sound and LCD screen. Sega had every intention of releasing the device even developing forward full games for it, but unfortunately due to development difficulties, Sega device never saw the light.

Then Nintendo had something in the works and in 1995 they brought the Virtual Boy to market with a lack of software supporting it, nausea-inducing black and red graphics and a scoliosis producing position to use. The Nintendo Virtual Boy was a commercial failure. The world and its technology were just not ready for Virtual Reality.



Figure 3.3. NASA's VIEW (12) on the left and Nintendo's Virtual Boy on the right (14)

But now, jumped ahead some 20 years, the advancement in technology has made computers more powerful, hardware smaller and displays more vivid and immersive. The advent of such technology has made VR in our homes a reality and there's no looking back.

In 2010 Google introduced a stereoscopic 3D mode for Street View and by 2014 companies such as Sony or Samsung announced they were working on VR projects and exploring VR possibilities.

The Oculus Rift is a Virtual Reality headset developed and manufactured by Oculus VR, released on March 28, 2016. The Oculus Rift has advanced the VR experience by including superior graphics, improved latency and wider range of motion.

HTC Vive is a Virtual Reality headset developed by HTC and Valve Corporation, released on 5 April 2016. This headset is designed to use "room scale" technology to turn a room into a 3D space via sensors,

with the virtual world allowing the user to navigate naturally, with the ability to walk around and use motion-tracked handheld controllers to vividly manipulate objects, interact with precision, communicate and experience immersive environments.



Figure 3.4. Oculus Rift HMD (15) on the left and HTC Vive HMD on the right (16)

PlayStation VR, known by the codename Project Morpheus during development, is a Virtual Reality gaming head-mounted display developed by Sony Interactive Entertainment and manufactured by Sony, it is designed to be fully functional with the PlayStation 4 home video game console.

Virtual Reality has significantly progressed and is now being used in a variety of ways, from providing immersive gaming experiences to teaching new skills. VR has many applications and with the rise in smartphone technology VR will be even more accessible.

With large numbers of companies competing, novel controllers being explored and lots of uses for VR, this field can only improve.

3.2. Types of Virtual Reality

Virtual Reality systems use various technological devices, and they are classified according to the user's level of immersion these devices can provide (10):

- **Immersion System:** this one is the ultimate version of VR systems. Immersion systems require an HMD supporting a stereoscopic view of the Scene, tracking the user's head orientation and position. They may involve some haptic and sensory interfaces, such as data gloves for user's interaction with the virtual world. These systems encase the visual and auditive perception of the user to make the experience fully immersive in the computer-generated world, without conception of what's happening outside.



Figure 3.5. Example of Immersive VR System (17)

- **Semi-Immersive System:** also known as hybrid systems. These systems are an improvement of non-immersive systems because they have a higher level of immersion. They give the user the perception of being in a different reality.

This type of VR is mainly used for training purposes and it consists of VR and the real world, using large projector systems, graphical computing and user interfaces. Physical environments are usually created to improve the experience and let the user interact with the real world.



Figure 3.6. Example of Semi-Immersive VR System (18)

- **Non-Immersive System:** also known as Desktop VR, it is the simplest type of Virtual Reality. Non-Immersive systems are the least sophisticated as they provide a lower level of presence. However, they are often used in education to enhance observation and understanding of the information. These systems are based on a conventional monitor displaying the image of the virtual world with no need for sensory outputs. The user and the virtual world interact by means of avatars; the user can control them with standard tools, such as a keyboard or a mouse.



Figure 3.7. Example of Non-Immersive VR System (19)

3.3. Virtual Reality Applications

In recent years, Virtual Reality has enhanced the development of many fields, from entertaining to medical appliances, because it provides a full range of benefits, with even more to provide in the coming years.

- **Business:** The use of Virtual Reality in business has become more and more extended due to the substantial cost savings, both in time and materials, it implies. The use made by companies includes training new employees without involving them in real work, and also to evaluate the impact of dangerous or harmful products in a virtual environment avoiding risks.



Figure 3.8. Simulation of an oil and gas Industry (20)

- **Leisure and Entertainment:** In society, Virtual Reality outstands mainly in the field of leisure. The videogame industry provides most Virtual Reality updates, by aiming for more noticeable player's immersion and experience. However, it is not only used for videogames but also to enhance people to visit locations without being there physically, to experience historical heritage, engaging the user into a more active interaction rather than passively.



Figure 3.9. Museum of Monte San Michele. "Great War Told Through VR" (21)

- **Education:** Virtual Reality benefits in education are countless. From elementary school by showing the Solar System, to university with medicine degrees simulating surgery. In fact, Virtual Reality offers motivation by allowing students to process information dynamically and interactively.
- **Scientific Visualisation:** Expressing complex ideas in three-dimensional space may ease understanding as well as enabling scientists of different disciplines to collaborate. For instance, viewing a molecular structure at different angles or by taking a look at the human body in an immersive and interactive environment (Virtual Reality society).

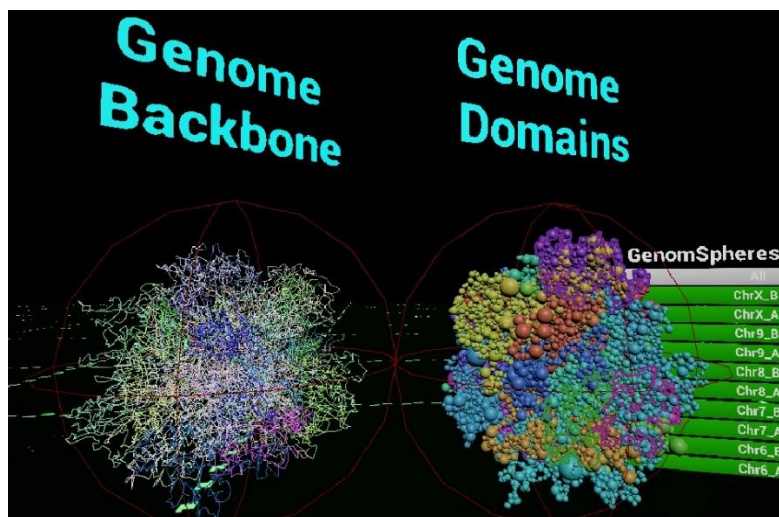


Figure 3.10. Visualisation of Scientific Data on the HTC Vive (22)

- **Medicine:** Virtual Reality has brought several improvements in the healthcare field, becoming a common practice in different areas of medicine like psychology, neuroscience, physical and mental therapy, clinical training or virtual robotic surgery, (Virtual Reality society).



Figure 3.11. Virtual Reality for Seniors Experiencing Nature (3)

- **Military:** The military sector gets benefits from Virtual Reality by training professionals without expending resources or involving them into dangerous situations. Instead, VR simulates these resources and situations to develop their skills to get better professionals. Among the uses provided are teaching soldiers to use military vehicles such as tanks or aircraft, medical training in battlefield or battlefield simulation to train reaction.
- **Engineering and architecture:** Design and manufacture in both engineering and architecture has been revolutionized with Virtual Reality technology. Virtual Reality brings the ability to observe some changes in a project by simulating it in an immersive environment, or it is used as an innovation tool to realize the dimensions of any project, among others.

3.4. Virtual Reality Applications for Rehabilitation and Well-being of Older Adults

As explained in previous sections of this project, Virtual Reality has many practical purposes outside the gaming industry. As of today, it has been used in many fields such as medical practice, education, architecture and other areas. Nowadays, the interaction of Virtual Reality with the brain has already led to applications that have proved more efficiency in health and medicine than in conventional

methods, including the treatment of post-traumatic stress disorder (23)(24), physical therapy, post stroke rehabilitation (25)(26)(27) and even pain distraction (4)(6). Even though VR applications may not be as impactful for the elderly's well-being than for younger generations in terms of emotions (28), there are still several benefits that scientists are exploring in order to change social attitudes by helping people see the world from another point of view.

However, this project focusses on applying this tool for rehabilitation purposes. More specifically, it focusses on the rehabilitation of the upper extremities of older adults living in nursing homes.

Several studies before have researched on the potential of Virtual Reality for sensorimotor rehabilitation after suffering a stroke, by engaging and motivating the patient to do a repetitive task on with a passive task, rather than an active one thus enhancing motor relearning (29)(30). Evidence-based controlled trials show that limb function rehabilitation with Virtual Reality therapy is not better than conventional therapy, but VR produces more significant improvements when used in addition to it (9). However, a meta-analysis by the *American Society of Neurorehabilitation* states that VR systems performed specifically for neurorehabilitation - task-specific practice, explicit feedback, increasing difficulty, implicit feedback, or variable practice - are valid and effective. Therefore, future researchers should find out what is the most effective technology, including the already proven VR (26).

Several controlled trials focussing on the effect of upper limb post-stroke rehabilitation show better results in the restoration of motor disorders when using VR technology than with conventional interventions. The results are even more significant by combining VR with conventional therapies (31)(32).

The effectiveness of using Virtual Reality training to improve gait and balance in post-stroke patients has also been proved. As compared to conventional therapies, the added value of Virtual Reality is believed to be the fact that it can produce much more repetitive and varied motor training (8)(27).

The effectiveness of post-stroke rehabilitation with VR tools not only has been studied, but there is also a wide range of studies that show improvements in therapies for Parkinson disease (33), multiple sclerosis (34) and postural instability and repeated falls in the elderly (1)(35).

There is also a study on the use of VR as a tool to identify patients with a higher risk of developing Alzheimer or dementia. This study, for example, uses navigation tasks done by the patients as a measure of entorhinal-based navigation and correlates the performances with MRI measures of entorhinal cortex volume. These comparisons can identify patients with mild cognitive impairment, at risk of developing dementia, and may aid early diagnosis of Alzheimer's disease (36).

In line with *Senopi's* values and objectives, this project also gives importance to the well-being of older people who reside in nursing homes. VR tools have been shown to improve people's ability to handle

emotionally rich life situations. These tools foster mastery of the environment and a sense of personal growth and autonomy, encouraging and motivating patients to engage in physical activity (28)(37).

4. Definition of the Project

4.1. Scale of the Project

This project aims to take part in the improvement of an already existing project developed for cognitive training. The root or mother project consists of a Virtual Reality application designed for the rehabilitation training of older people, in which they can visit a natural environment by executing movements like running, flying and swimming. Thus, the main task has been to develop the flying module of this project.

4.2. Project Planification and Methodology

In order to get the project done in time and to accomplish the objectives, the project was divided into several simple tasks. Once these parts are put together, they become a meaningful whole.

The project was divided as follows:

- Learn C# and Unity language
- Definition of user's movements and interactions
- Information research about biomechanics
- Understand the Virtual Reality tracking system
- Develop a movement detection method in Unity
- Create a "flying experience" within the environment
 - Flap
 - Glide
- Alternatives to develop without HTC Vive Equipment
 - Controllers
 - Headset
- Write the project report
- Test the application
- Meetings
- Fix errors and clean the code

During the project, a few meetings were held. A first meeting was organised with Dr Stavros and other students, to explain both the current status of the project and the way each group was going to take part in it. After this presentation, another meeting was organised to clarify all the ideas. The third meeting with Dr Stavros was never held due to the Covid-19 pandemic. However, later on, two more

meetings took place, being those, an additional meeting with coordinators to help determine how to proceed, provided the lack of resources - HTC Vive equipment and the LAM's computers -, and a final meeting with Dr Stavros both to report the project's progress without having these resources and to show him the alternatives used. This last meeting with Dr Stavros also contributed to establishing a methodology that will set the path for future students, who will go on with the project.

In the beginning, learning C# and Unity language was undertaken without knowing the project's exact details. Later on, as more and more details were disclosed, new steps were gradually taken. It was then made clear that some more C# and Unity learning was needed to add or remove pieces of work, with the aim to improve the whole project.

Consequently, after knowing the details of the project, it was necessary to search for information both on how HTC Vive tracks its controllers and how upper body biomechanics work to execute the desired movements. Now, the following step consisted of merging all the knowledge achieved in order to create a Unity system for the detection of the user's upper limb movements.

Come to this point, all the data collected were translated into movement, creating the flying experience, which is split into flap and glide.

Last but not least, the app was tested, and the code cleaned to get maximum optimization.

4.3. Gantt Diagram

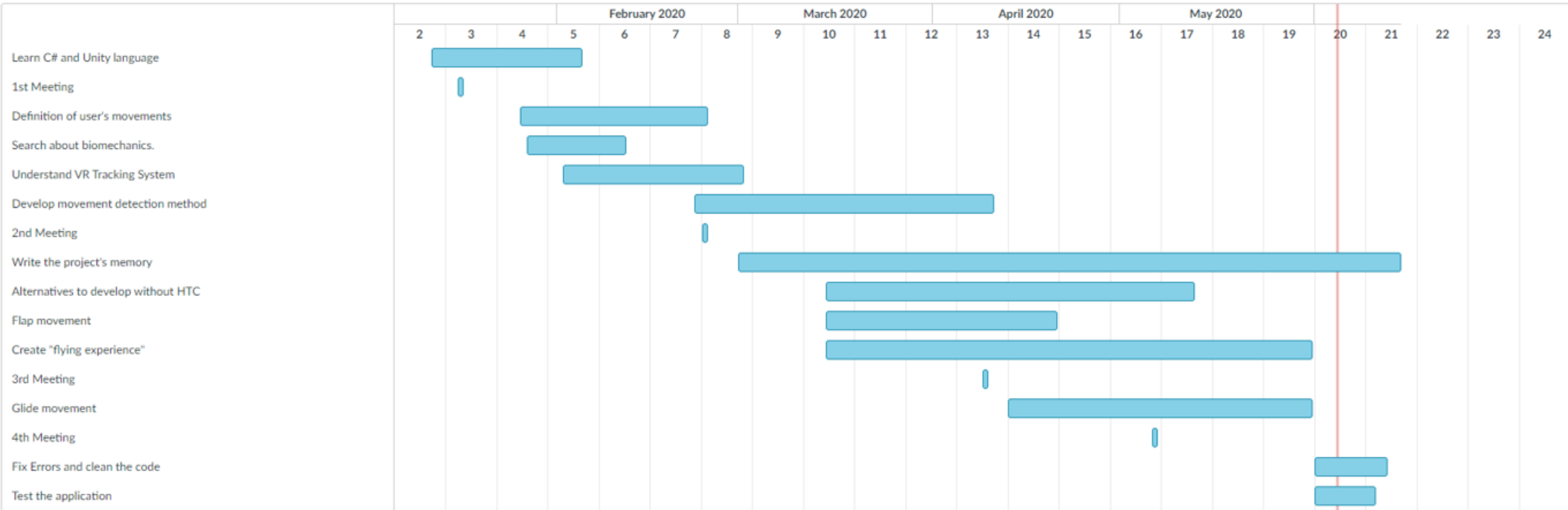


Figure 4.1. Gantt Diagram

TASK NAME	START DATE	FINAL DATE
1st Meeting	1/16	1/16
Learn C# and Unity Language	1/12	2/4
Information Research about Biomechanics	1/27	2/11
2nd Meeting	2/24	2/24
Definition of User's Movements and Interactions	1/26	2/24
Understand the Virtual Reality Tracking System	2/2	3/1
3rd Meeting	4/9	4/9
Develop a Movement Detection Method in Unity	2/23	4/10
Flap Movement	3/15	4/20
4th Meeting	5/6	5/6
Alternatives to Develop without HTC Vive Equipment	3/15	5/12
Create a "Flying Experience" within the Environment	3/15	5/31
Glide Movement	4/13	5/31
Test the Application	6/1	6/10
Fix Errors and Clean the Code	6/1	6/12
Write the Project Report	3/1	6/14

Table 4.1. Gantt Diagram Tasks

5. HTC Vive

HTC Vive is a Virtual Reality headset developed by HTC and Valve released in 2016. The system's headset uses room-scale tracking technology; it is a Virtual Reality device that allows some degree of physical scrolling and real hand tracking. HTC Vive is a system of peripherals that connect to the computer via USB and HDMI or Display port (38).

On devices like the Oculus Go or Google Daydream an immobile virtual experience as an outside observer is achieved, like a film viewer. In contrast, the HTC Vive device enhances the virtual experience including physical movement and hand tracking, thus enhancing the feeling of immersion.

The difference between one virtual experience and the other is made by how accurately each VR platform can track the user's movements. DoF, or degrees of freedom, measure this difference. A VR system's DoF describes the level of rotational movement the system can track (39).

On the one hand, the first devices mentioned (Oculus Go and Google Daydream) only use rotational tracking with three degrees of freedom (3DoF). This fact implies that they can only track head rotations. The user can look up and down or sideways or tilt his head side-to-side, and the virtual world will move with him but not let him move around the virtual world physically. What is more, direct interaction with the virtual environment is not allowed since controllers are also rotation-only, acting only as pointers.

On the other hand, devices like HTC Vive have positional tracking with six degrees of freedom (6DoF) in order to create an experience as realistic as possible. Positional tracking allows the user to be tracked as he moves around the room, causing movement throughout the virtual environment. If controllers that also have 6DoF are used, then the user can also interact directly with virtual world objects by moving their hands in the real world.

To recap, 3-DOF rotational tracking devices can track the direction that the VR headset is pointing towards to know where the user is looking, whereas 6-DOF positional tracking devices can track the direction and position within space to know where he is looking and located in the room.

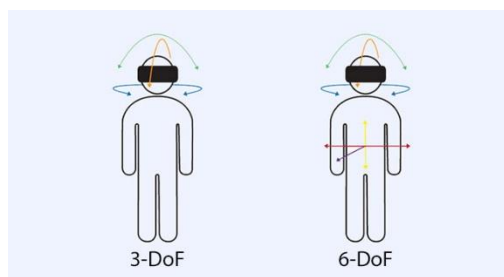


Figure 5.1. Degrees of Freedom (39)

5.1. HTC Vive tracking System

Rotational tracking (3DoF) is achieved with the use of microscopic electromechanical gyroscopes. Combining these gyroscopes with a variety of technologies, positional tracking (6DoF) is achieved.

HTC Vive uses a SteamVR technology called Lighthouse (40), a unique system designed to allow room-scale positional tracking without having to wire sensors back to the user's PC. By using the Lighthouse technology, there is no need for the computer to process any data, as no camera readings or complex vision algorithms on the computer are required.

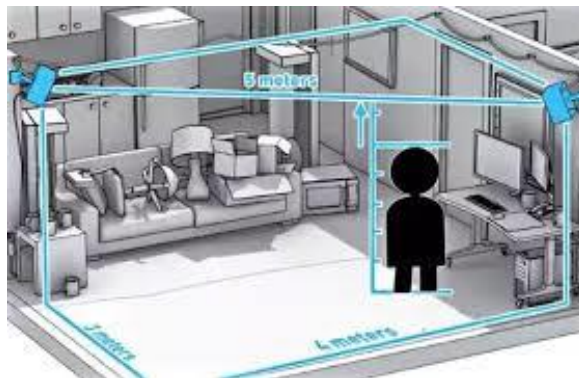


Figure 5.2. Lighthouses Position in the Play Area (38)

Base stations, also known as Lighthouses, are placed at high positions on opposite corners of the room (Figure 5.2). The Lighthouses bathe the entire room with two-dimensional infrared (IR) laser beams, left-to-right and then top-to-bottom, emitting an IR flash before each sweep. Then the sensor covered headset and controllers detect these laser messages. This pattern allows any receiving sensor (IR photodiodes connected to a chip) to understand where a laser beam originates and the time lapse between the IR flash and being hit by the laser.



Figure 5.3. Two-dimensional Infrared Laser Beam (41)

This information enables the system to identify the position of the sensor. Each laser message received on a headset and controllers allows the system to pinpoint the devices in the room with sub-millimetre accuracy.

5.2. HTC Vive Hardware Characteristics

HTC Vive box includes the following main peripherals:

- Headset: HMD with 36 infrared sensors around it to track and record its position. Headset characteristics are specified in Table 5.1.

HEADSET SPECIFICATIONS

SCREEN	Dual AMOLED 3.6'' diagonal
RESOLUTION	1080 x 1200 pixels per eye
REFRESH RATE	90 Hz
FIELD OF VIEW	110 degrees
SENSORS	SteamVR Tracking, G-sensor, gyroscope, proximity
CONNECTIONS	HDMI, USB 2.0, stereo 3.5 mm headphone jack, Power, Bluetooth
INPUT	Integrated microphone
EYE RELIEF	Interpupillary distance and lens distance adjustment

Table 5.1. HTC Vive Headset Specifications (42)

- Controllers: two wireless controls, each one is consisting of a set of 24 infrared sensors and different input methods. Controller characteristics are specified in Table 5.2.

CONTROLLER SPECIFICATIONS

SENSORS	SteamVR Tracking
INPUT	Multifunction trackpad, Grip buttons, dual-stage <i>Trigger</i> , System button, Menu button
USE PER CHARGE	6 hours approximately
CONNECTIONS	Micro-USB charging port

Table 5.2. HTC Vive Controller Specifications (42)

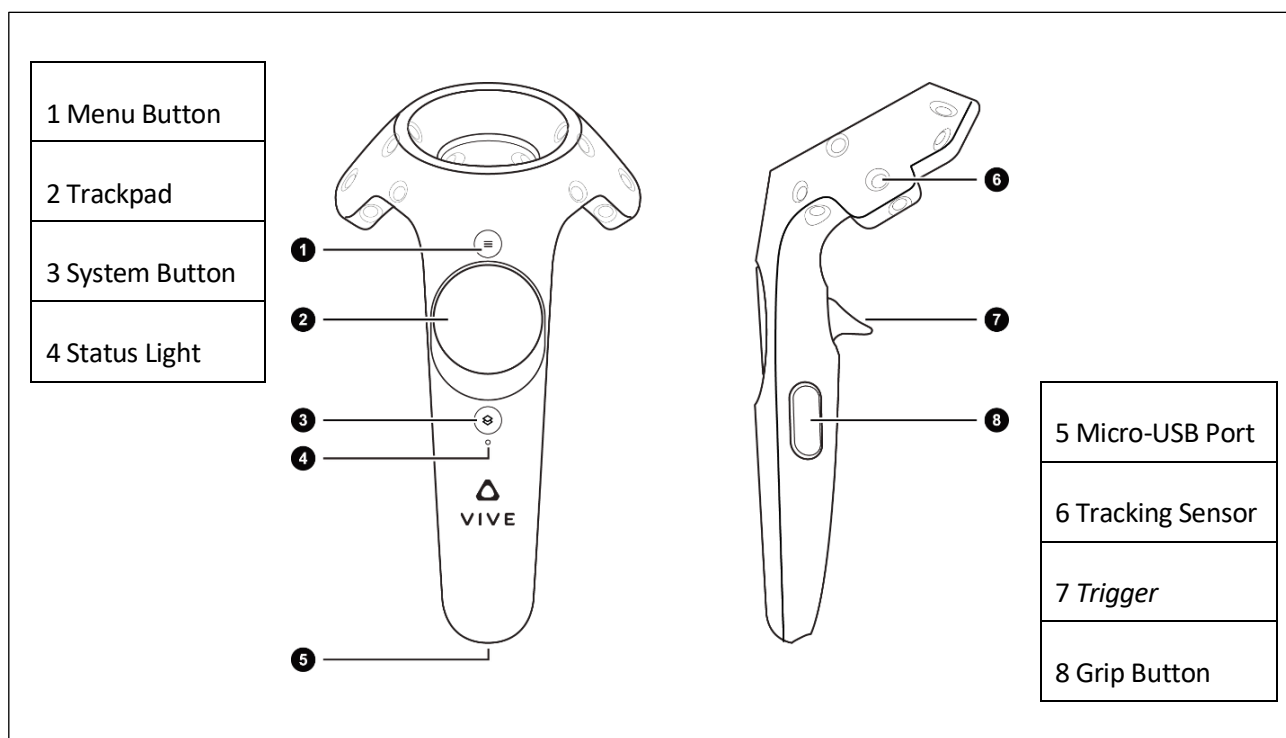


Figure 5.4. Controller Input

- Base stations: two position sensors with a 120-degree visual field that require connection to a plug and direct vision between them. It is recommended to place them more than 2 meters above head height, angle down 30 to 45 degrees and at a maximum distance of 5 meters between them.
- Link box: junction accessory that connects the headset to the computer. It has the following elements: 3-in-1 cable, power port, USB port, Mini Display Port (used if the computer does not support HDMI) and HDMI port.



Figure 5.5. HTC Vive Components (38)

5.3. HTC Vive System Requirements

For proper operation, HTC Vive demands a powerful computer to meet the requirements of Virtual Reality. For this reason, before use, it should be verified that the computer meets the minimum requirements shown here below in Table 5.3.

COMPONENT	MINIMUM SYSTEM REQUIREMENTS
GPU	NVIDIA GeForce GTX 970, AMD Radeon R9 290 equivalent or better
PROCESSOR	Intel Core i5-4590/AMD FX 8350 equivalent or better
MEMORY	4 GB RAM or more
VIDEO OUTPUT	HDMI 1.4, DisplayPort 1.2 or newer
USB PORTS	1x USB 2.0 or newer
OPERATING SYSTEM	Windows 7 SP1, Windows 8.1 or later, Windows 10

Table 5.3. HTC Vive Minimum System Requirements (42)

5.4. Configuration of HTC Vive with SteamVR

Any Virtual Reality hardware requires software that enables its functionality. Thus, HTC Vive requires using Valve's Steam platform. This platform includes a section for Virtual Reality called SteamVR, which is an ultimate tool where VR experiences are integrated. It is necessary to install Steam and SteamVR SDK (43).

SteamVR shows the status of the hardware being used, in this case, HTC Vive, indicating whether all components are correctly connected (headset, controllers and base stations).

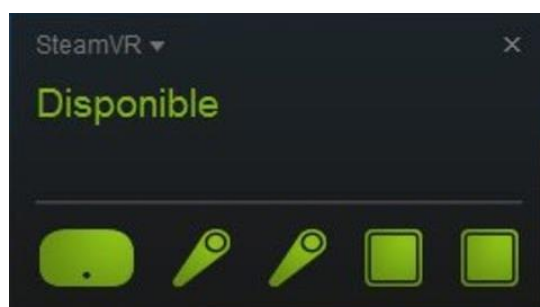


Figure 5.6. HTC Vive's Components Status

An essential requirement to take into account when setting up Steam VR is the size of the play area. It is necessary to configure the play area before starting using HTC Vive. If the goal is to experience VR while seated or standing, there is no minimum space required. However, if there is a need to have space to move, the room scale must be set with a minimum area of 2 x 1.5 meters.

Another benefit of SteamVR is the utility design called Chaperone system, which assures that users will not collide with any physical barriers during the virtual experience. The Chaperone system shows the boundaries in the play area. These limits are set when starting Steam VR with the controllers. Once set up, Chaperone keeps track of where a user is in relation to the physical walls and objects around them, to warn them when they approach a physical barrier. If necessary, Chaperone displays a blue grid pattern within the user's virtual space to let them know they are close to a physical element, as they are blind because of the headset they are wearing. Thus, it is solved the problem of real space navigation in Virtual Reality.

6. Unity

Unity is a game development engine. In other words, it is a software with programming routines that allow the creation and operation of a virtual environment. Using this engine, there is the possibility to create 2D or 3D games, Virtual Reality and augmented reality games, and even simulation experiences (44).

Unity is one of the most used engines when it comes to learning how to develop games thanks to the size of its community. It is a big community that allows access to all kinds of documentation and forums to resolve doubts and learn development methods.

This software has a visual editor and programming via scripts to get excellent results. Besides, Unity allows creating video games not only for computers but also for other platforms such as videogame consoles or mobile phones.

An application named Unity Hub must be downloaded to start using Unity (45). This application works as a desktop where installed versions and developed projects can be managed. It is recommended to install the latest version of Unity, as all Unity projects can be opened using newer versions but not using older ones that the one used for development.

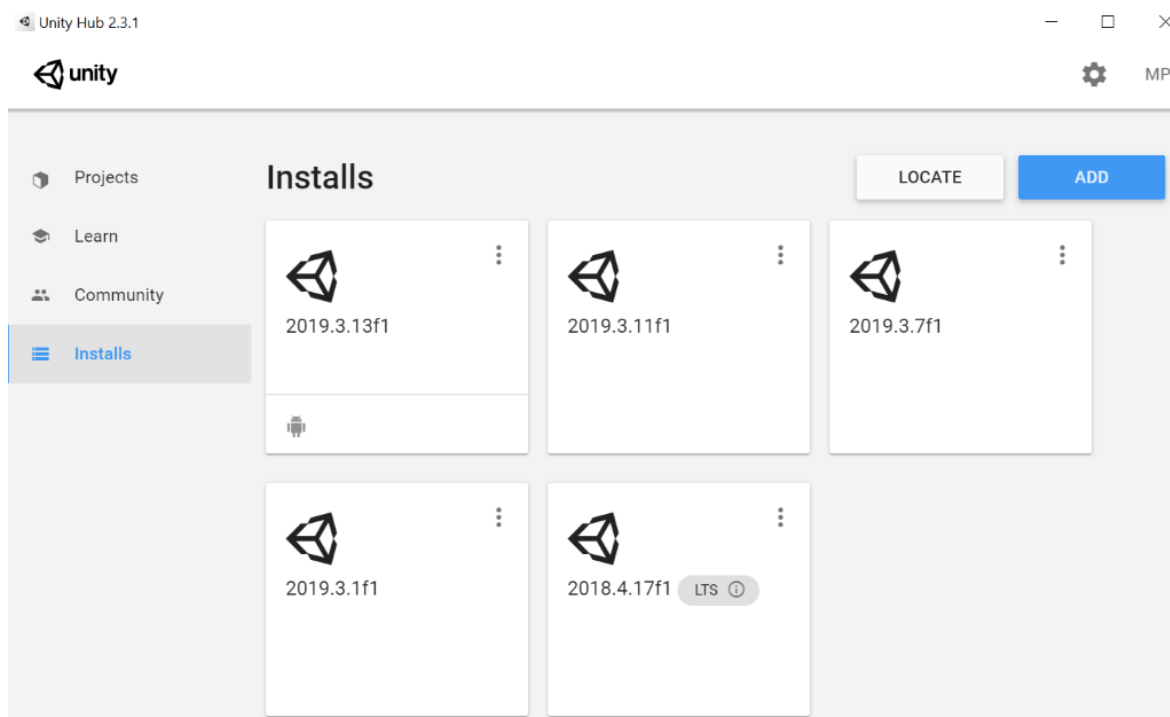


Figure 6.1. Unity Hub

6.1. Unity Interface

Once Unity is installed, and a new project is opened, the visual editor is the centre of the game's content creation.

This section takes a look at the Unity interface. Notice that the editor windows can be rearranged according to the developer's preferences. Default positions are shown below:

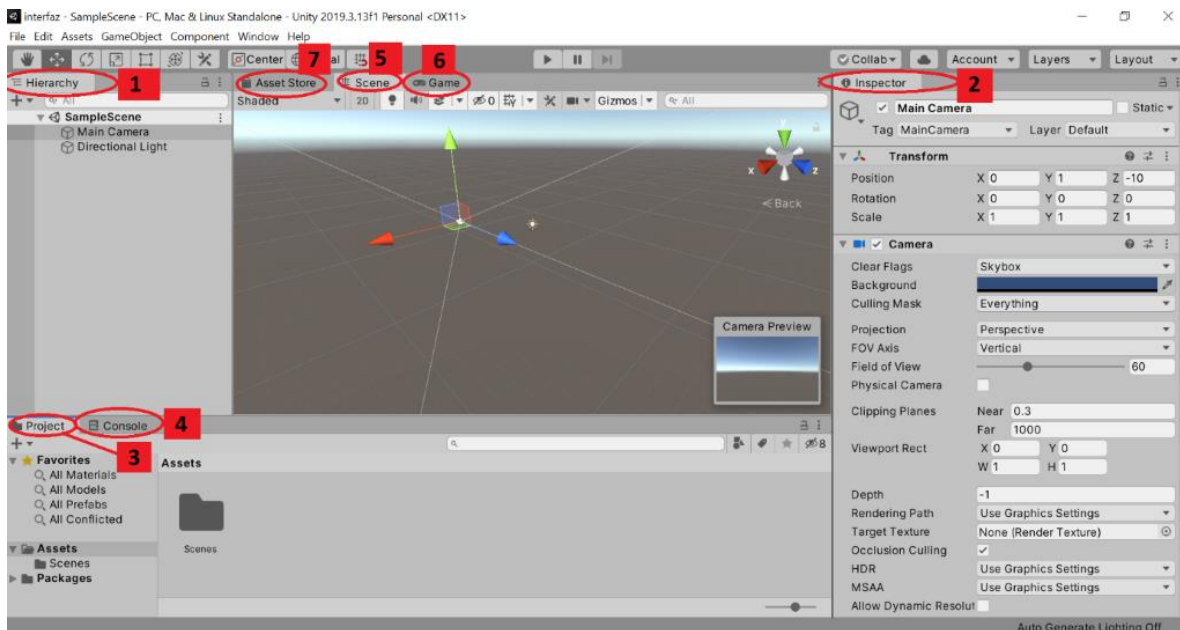


Figure 6.2. Unity Interface

1. **The Hierarchy Window** shows all the elements in the Scene - from lights to terrain, as well as player and control elements. It allows the developer to create new objects and attach one object to another. It is from this window that the developer establishes and controls the hierarchical structure that rules the objects.
2. **The Inspector Window** displays information about the selected object, such as its position in the Scene, components, or associated scripts. This window allows for editing the immediate properties of the objects. If it is wanted to see the inspector of a single object without changing it when selecting another object, the window can be locked with the padlock in the upper right corner.
3. **The Project Window** is the folder that contains all the assets, such as scripts, animations or models that can be used in the project. If the developer drags an object from Project to Scene, he introduces that object into the Scene. Otherwise, if he drags an object from Hierarchy to Project, you create a prefabricated object.
4. **The Console Window** is another tab. It shows the errors occurred when compiling the project. It also works as a display for script prints or debugs.

5. **Scene View** is where the first changes are seen when developing a game. This section allows you to move the camera and the elements of the game, editing the game and reviewing the scenario when creating the virtual world.
6. **Game View** is where the final version of the game being developed is shown, so if the project has interactive elements, this window allows you to test it. When the PLAY button on the toolbar is pressed, Unity will warn you if any errors occur while running the project and show how it works.
7. **Asset Store** is the Unity resource store, where developers add resource packages (materials, animations and scripts among others) that can be incorporated into the project. These packages can be free of charge or payable.

6.2. Unity System Requirements

Table 6.1 below shows the minimum computer requirements necessary for the Unity engine to develop a project, according to the operating system:

COMPONENT	MINIMUM SYSTEM REQUIREMENTS
OPERATING SYSTEM VERSION	Windows: Windows 7 (SP1+) and Windows 10, only 64-bit versions macOS: Sierra 10.12.6+ Linux: Ubuntu 16.04, Ubuntu 18.04, and CentOS 7
CPU	SSE2 instruction set support
GPU	Windows: DX10, DX11, and DX12 macOS: Metal-capable Intel and AMD Linux: OpenGL 3.2+ or Vulkan-capable, Nvidia and AMD

Table 6.1. Unity 2019.3 Minimum System Requirements (32)

6.3. Unity Connections to HTC Vive

If you want to use Virtual Reality, you must enable the *Virtual Reality Supported* settings on your Unity interface. To do this, first you open the *Project Settings Window* from the Edit tab. Then, you go to the XR Settings option in Player Settings, where you check the *Virtual Reality Supported* option (46).

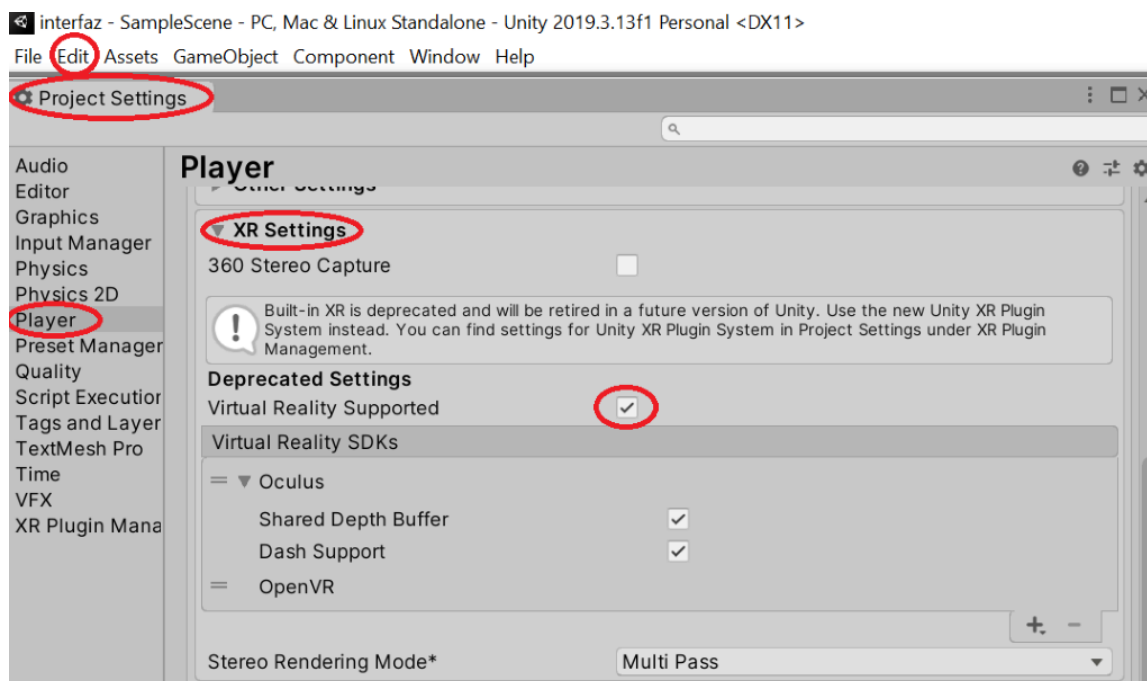


Figure 6.3. Virtual Reality Supported Settings

Once HTC Vive is set up with SteamVR (see [Section 5.4](#)), the next step is to open a Unity project and download the SteamVR Plugin through the Asset Store. The package must be imported before it can be used from Unity.

Once imported, a folder called SteamVR appears in Unity's Project window. This folder contains all the tools needed for development. Inside this folder, there is the Prefabs folder, which contains essential VR GameObjects such as the Camera Rig (element that manages headset and controllers) or a GameObject called SteamVR (prefab which has a component attached that handles the rendering of VR cameras).

When SteamVR detects the controllers, their virtual versions are created as children of the CameraRig. At this point, the camera is associated with the headset and the virtual controllers to the real ones, accurately tracking the real movement (47).

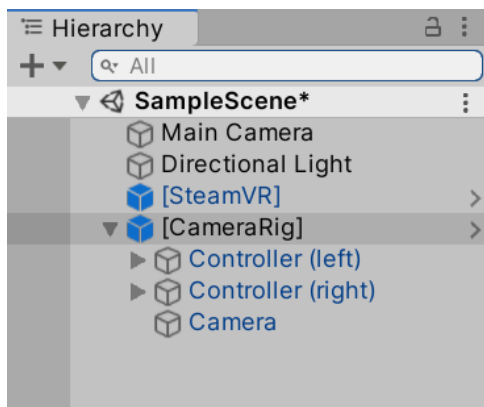


Figure 6.4. SteamVR Prefabs in Hierarchy Window

7. Microsoft Visual Studio

Visual Studio is a Microsoft integrated development environment (IDE) for Windows, Linux, and macOS. This environment allows developers to create websites, web apps, and mobile applications, among others (48). Visual Studio is a Microsoft integrated development environment (IDE) for Windows, Linux, and macOS. This environment allows developers to create websites, web apps, and mobile apps, among others (48).

Microsoft Visual Studio supports 36 different programming languages, which can be edited and debugged from the Visual Studio code editor. Supported programming languages include C, C ++, C #, JavaScript, or Visual Basic .NET. Other languages such as Python or M are available through plugins.

7.1. Microsoft Visual Studio Link to Unity

As mentioned in [Section 6](#), in addition to the visual editor, the Unity engine offers a primary scripting API in C#.

In Unity, components attached to GameObjects control their behaviour. These components can be created or modified via scripts. Scripts are created directly in Unity, in the previously selected folder in the Project Window. These scripts open by default in the Visual Studio editor, as Visual Studio integration to Unity, allowing to create and maintain project files automatically.

To ensure that Visual Studio is installed correctly, the Preferences Window must be opened from the Edit tab. Then, in External Tools, Visual Studio should be the External Script Editor (49).

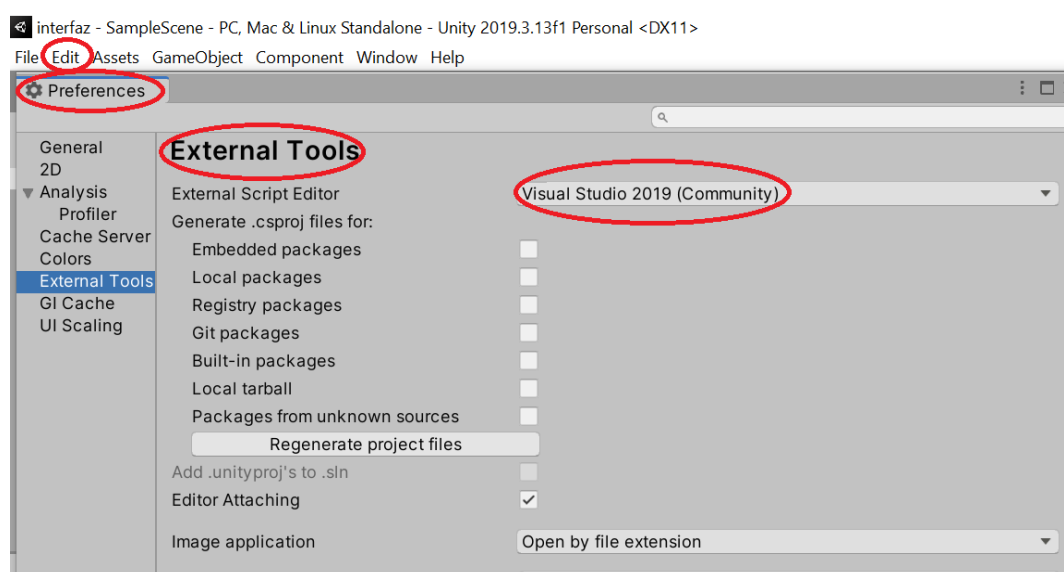


Figure 7.1. External Script Editor Settings

7.2. Microsoft Visual Studio Requirements

The minimum system requirements to use Microsoft Visual Studio 2019 are listed in Table 7.1.

COMPONENT	MINIMUM SYSTEM REQUIREMENTS
PROCESSOR	1.8 GHz, Quad-core or better
MEMORY	2 GB RAM or more
HARD DISK SPACE	800 MB
HARD DISK SPEED	SSD
DISPLAY RESOLUTION	720p or higher
OPERATION SYSTEM	Windows Server 2012, 2016, 2019; Windows 7 SP1, Windows 8.1, Windows 10

Table 7.1. Microsoft Visual Studio Minimum System Requirements (50)

8. Application Design

In this section, the development of the VR application is analysed step by step from beginning to end. Essential points, such as controller tracking and movement within Unity, are described.

During the project, some alternative ways to develop the application are considered, which are also depicted.

8.1. Set of Movements

Initially, Dr Stavros Skouras, Senopi's CTO, proposed an application which involved the player in flying in a natural environment. As shown in Figure 8.1, the first approach used simple flying movements.

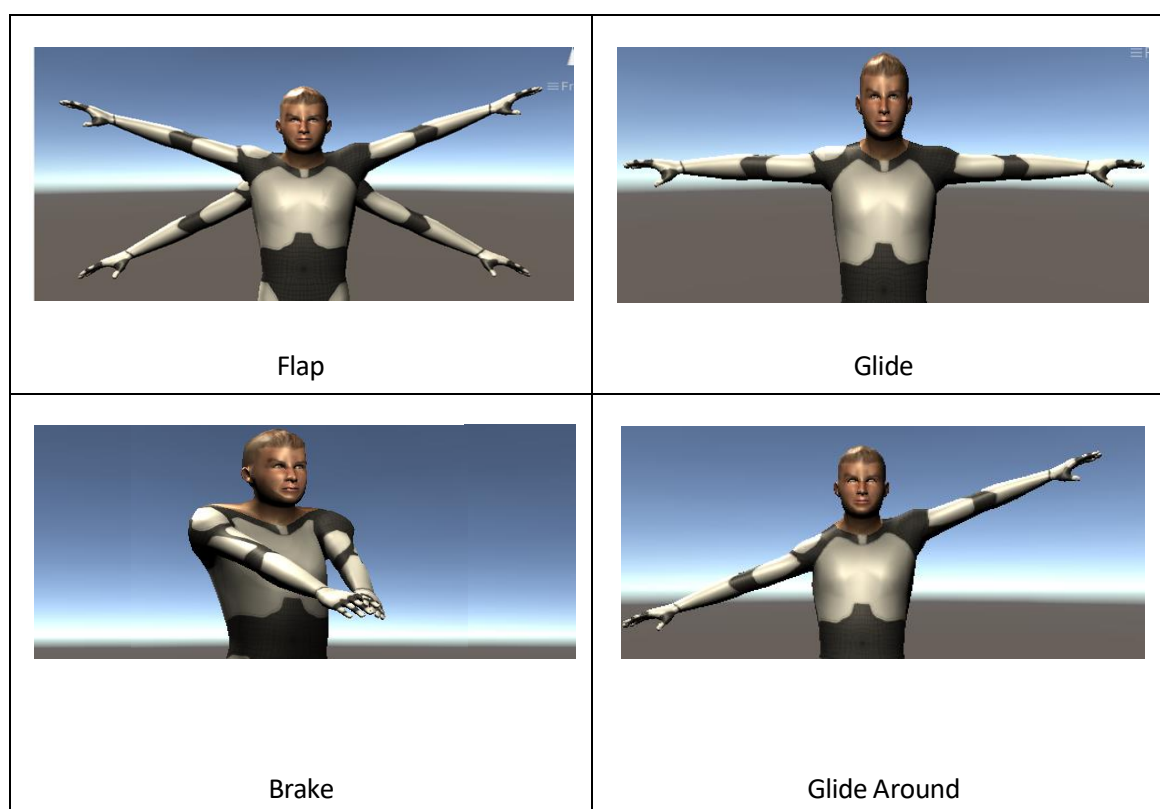


Figure 8.1. Initial Set of Movements

After the second meeting with Dr Stavros, the braking movement was removed, being then Flap, Gide and Glide Around the set of movements to consider.

8.1.1. Movement Analysis

Once the set of movements was clear, it was time to analyse these movements according to humans' motion range and its limitations.

The analysis started with flapping and gliding, which involve both the abduction and adduction of the arm. In this case, the arm motion ranges from 0° to 180° in abduction, and from 0° to 75° in adduction, as shown in Figure 8.2.

As the aim was to relate this motion range to flap and glide actions, patients should focus on abduction movements at least between 45° and 135°, marked in orange in Figure 8.2.

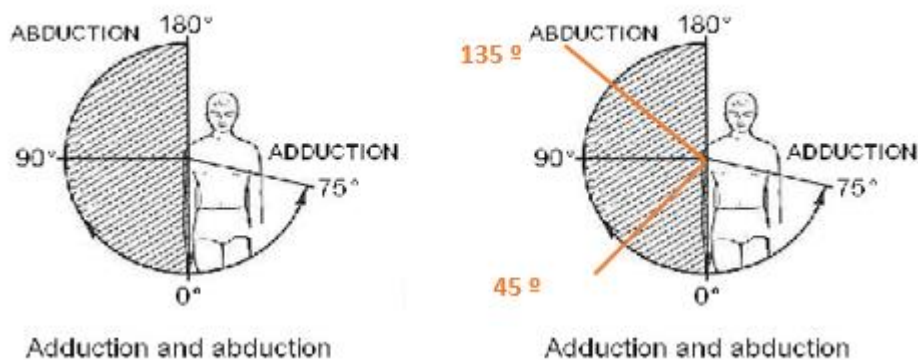


Figure 8.2. Human Glenohumeral Joint Movements (51)

In case any patient suffers from an illness that prevents he or she from doing the natural abduction movement (two dimensional), a small flexion and extension range has been considered for Unity to detect motion even if it is not perfect. This short motion range was in both extension and flexion motions (Figure 8.3).

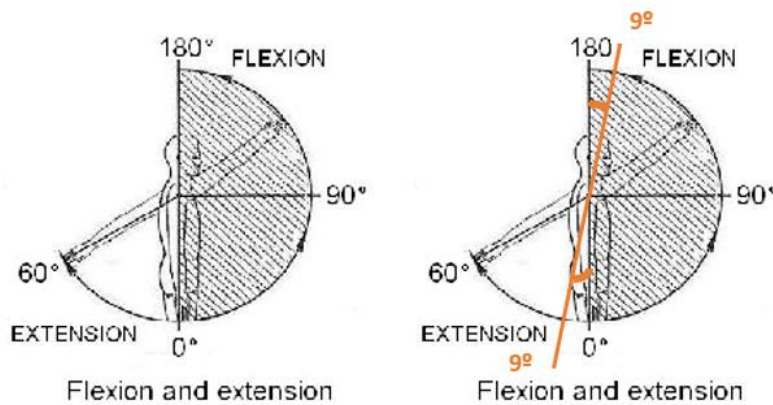


Figure 8.3. Human Glenohumeral Joint Movements (51)

Lastly, taking into account that the flying action has its origin in the hand (Figure 8.4), the wrist movement helps understand how the rest of the arm is going to behave. Mainly, it involves knowing the direction in which the flap or glide movement is going to take. For instance, if the wrist is turned 20° , it will change the 45° abduction angle by adding another point of direction into the Z-Axis.

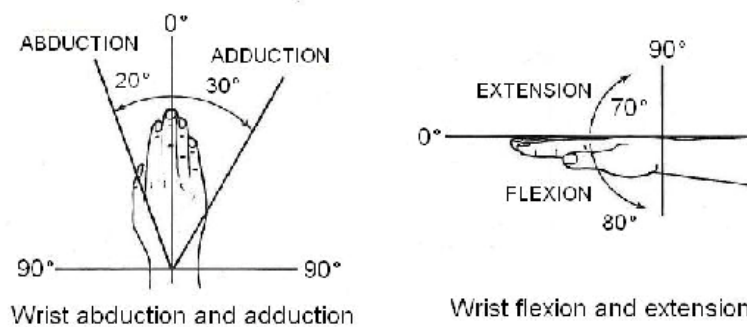


Figure 8.4. Human Wrist Joint Movements (51)

8.2. Kinematics into Movement Tracking

The sensors built and placed in the room where VR is going to be used are responsible for movement tracking. To translate all this data into Unity, Camera Rig from SteamVR needs to be placed in the Scene (Figure 8.5), making it match the avatar's place that is being used.

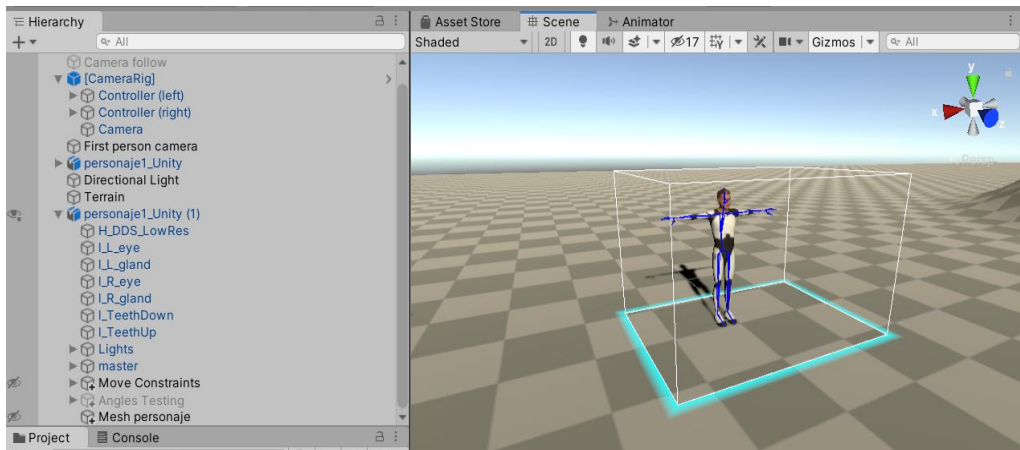


Figure 8.5. Camera Rig from Steam VR

Once everything is duly positioned in the Scene, it is time to transfer the user’s movement to the avatar. A Package named *Animation Rigging* is used to ease this movement transfer. *Animation Rigging* can be downloaded by accessing *Unity Packages*. Secondly, then selecting *Advanced* and choosing *Preview Packages*, a list of packages will show up on the left, as shown in Figure 8.6, *Animation Rigging* has to be imported into the project.

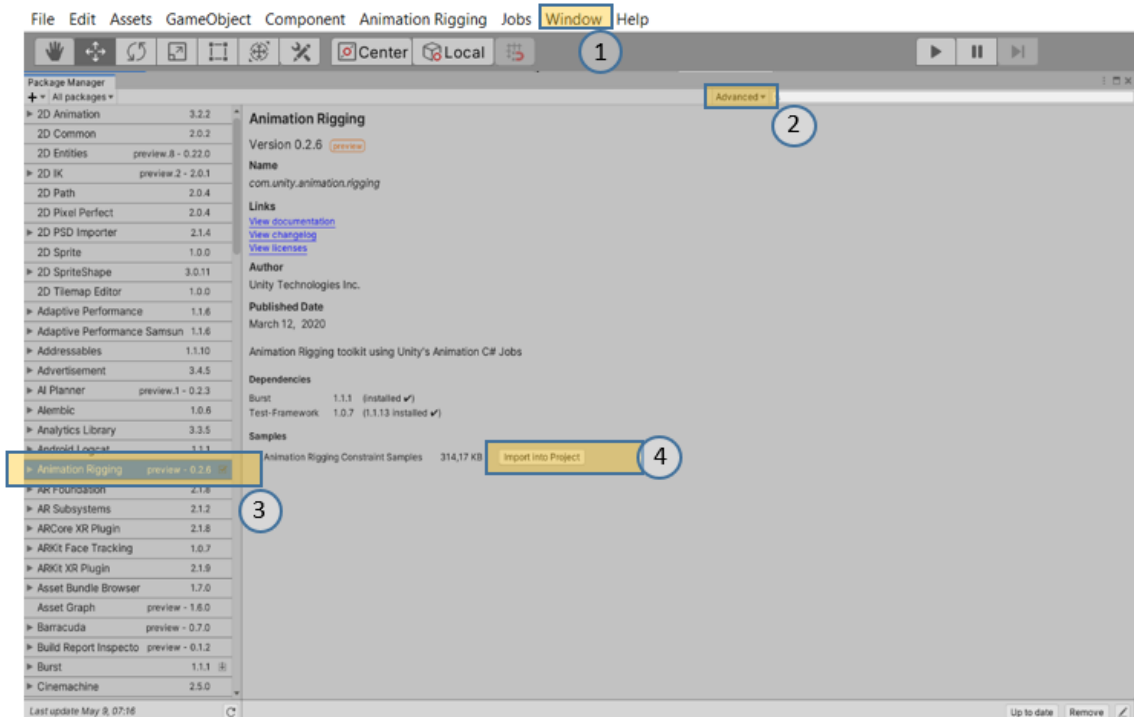


Figure 8.6. How to Import Animation Rigging

This Package contains different scripts that make the movement of the avatar even more accurate and realistic, by reproducing the movement of all the upper body, transforming forward kinematics into inverse kinematics. For instance, if a hand is moved towards the chest, the arm is going to be moved accordingly to the rest of the joints, instead of acting like a rigid object.

The main components for this work are listed in Table 8.1:

SCRIPT	DESCRIPTION
RIG BUILDER	It allows alongside the animator component to create the playable move by attaching different Rigs to it.
RIG	It is the component that stores all the rig constraints within its hierarchy following an evaluation order similar to the own object hierarchy.
BONE RENDERER	Allows the developer to define a hierarchy of bones for visualization and definition of the character skeleton for rigging purposes.
MULTI-PARENT CONSTRAINTS	It moves and rotates a GameObject as if it is the child of another GameObject in the Hierarchy window. It offers the benefit of choosing which of 3DoF can influence as a parent and also not getting affected by scale.
TWO BONE IK	This component controls and manipulates two forward kinematic bones into an inverse kinematic expression by using a Target and a Hint, which work as a hand and elbow like respectively.

Table 8.1. Animation Rigging Scripts Components

Consequently, Two Bone IK component has been used to match the controller's movement with the arm movement, and therefore, Multi-Parent constraint has been used to track the headset rotation. To see in more detail how they work, go to [Annex A1](#).

8.2.1. Linking VR with Avatar

Once Kinematics has been applied to the patient's avatar, controllers and the headset have an inner rotation and position since the game starts, so it may rotate the head and the hands of the avatar in unrealistic grades or positions, which are not natural or even possible. Therefore, an adjustment needs to be done once the game starts. To do so, each hand and the head constantly get repositioned and rotated again to match the Virtual Reality equipment. For instance, head rotation has its inner

limitations as shown in Figure 8.7, and to be able to rotate fully, our body rotates in order to match the desired end position, another example would be if the patient wants to rotate his head in order to see what is 90° on the left side, the body must rotate some certain degree to be able to match the biomechanics requirements of the neck rotation. Therefore, an adjustment needs to be done once the game starts. To do so each hand and the head constantly gets repositioned again to match the Virtual Reality equipment.

Take head rotation, for instance. As shown in Figure 8.7, our head has inherent limitations that prevent its full rotation. Therefore, our body must also rotate to match the desired end position. Another example is that of a patient that rotates his head to see what is 90° on the left side. In this case, his body must also slightly rotate to match the biomechanics requirements of neck rotation.

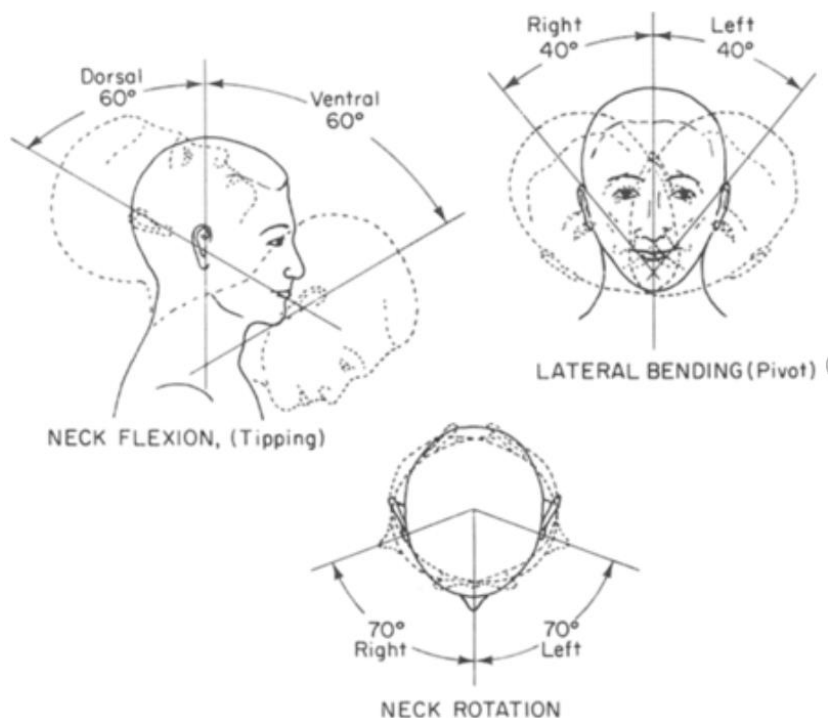


Figure 8.7. Neck Motion Range (52)

A script called *VRRig* has been developed to correlate head movement with the body. *VRRig* interpolates the user's desired direction with the actual body direction, thus getting the axis of the body move towards the head's Z-axis.

The *VRRig* script also links the VR equipment to the Targets and head constraints used to perform inverse kinematics (Figure 8.8).

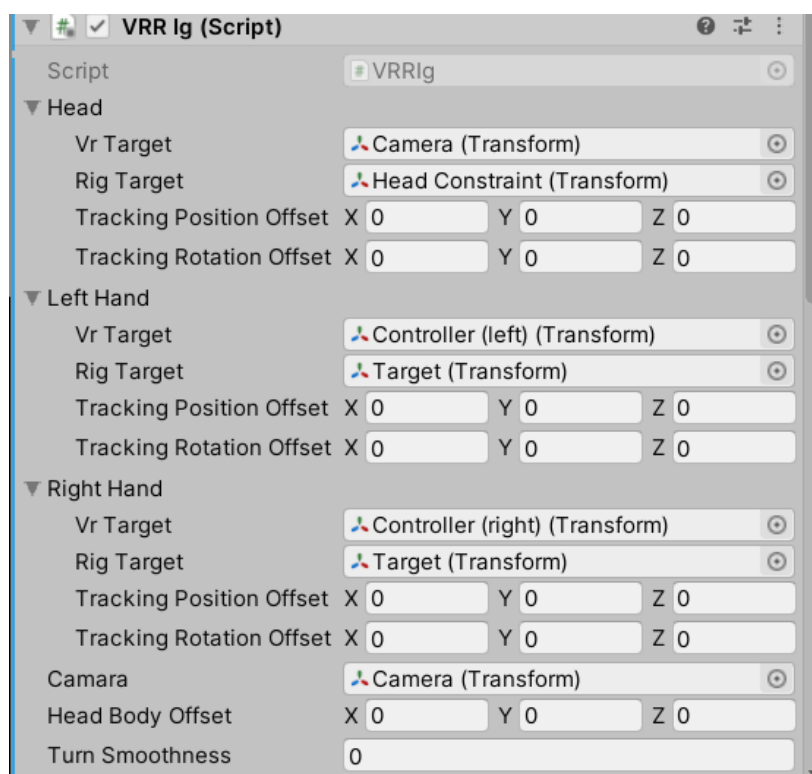


Figure 8.8. VRRig Script in Unity's Inspector

8.3. Movement Detection in Unity

Once the issue of movement tracking is solved, the set of movements previously considered needs to be detected within the game in order to create the flying experience.

The first approach to detect movement was by detecting the angle of each hand with its corresponding shoulder in order to execute the flying experience. However, some technical constraints have to be considered beforehand.

8.3.1. Technical Constraints

Both the software and design of the avatar itself reveal certain constraints that prevent full detection.

As far as design is concerned, the avatar's body skeleton shows that right and left arm vectors are entirely different, not only their X, Y, Z directions, but also their position with respect to the local reference. This constraint makes it difficult to standardize how angles work (Figure 8.9).

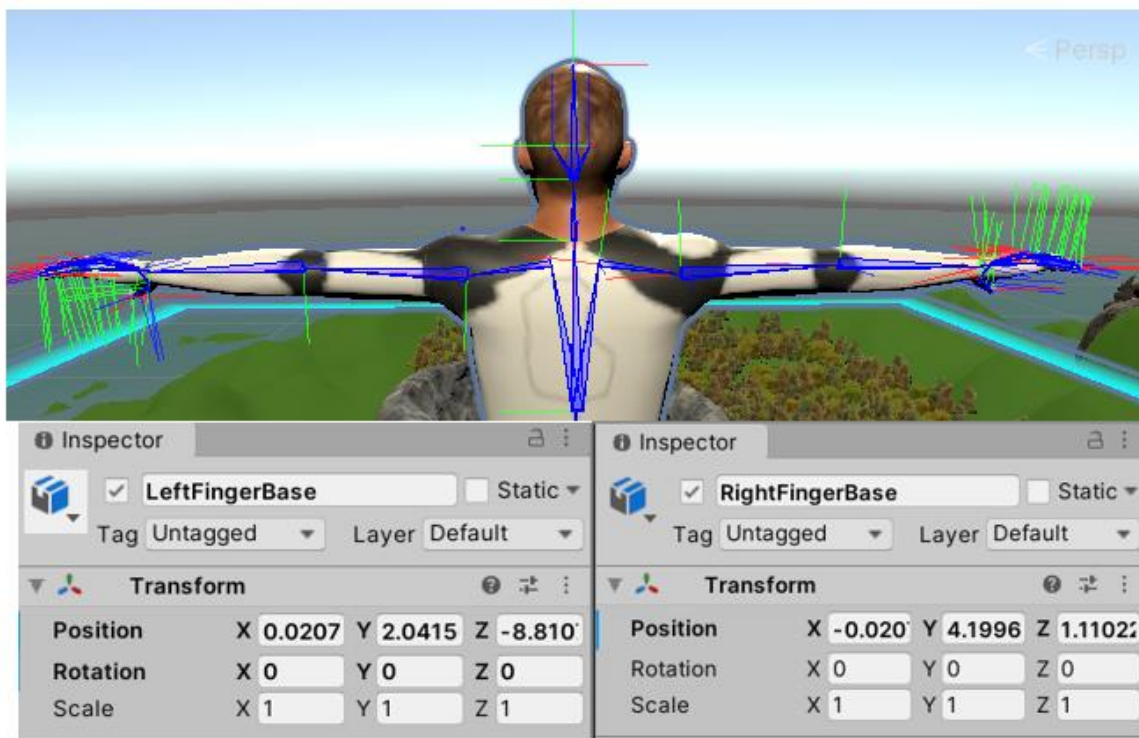


Figure 8.9. Each Bone Axis (Upper-side Picture). Left-hand and Right-hand Position Vectors (Bottom-side picture).

Regarding software, there are other issues to consider. One of them is the range of angle detection. The results of angle calculation (see detail in [Annex A2](#)) show unexpected angle values. The resulting value chart (see Table 8.2) is then analysed to find out the value tendency and evaluate whether a calibration equation is feasible or not. The analysis shows that real values draw a different line, which does not match all expected values or points (Figure 8.10). Thus, in this case, a calibration equation is not feasible.

Right Limb	α	-90	-75	-60	-45	-30	-15	0	15	30	45	60	75	90
	α'	-31	-46	-44	-34	-20	-7,82	1E-06	13	20,56	23,05	28,16	37,04	48
Left Limb	α	-90	-75	-60	-45	-30	-15	0	15	30	45	60	75	90
	α'	-20	-20,5	-20,3	-17	-13,3	-9,27	3E-07	4,369	13,37	16,86	18,92	19,72	18,65

Table 8.2. Upper Limbs Angles Calculations Results. α = Expected Angle. α' = Current Angle in Unity.

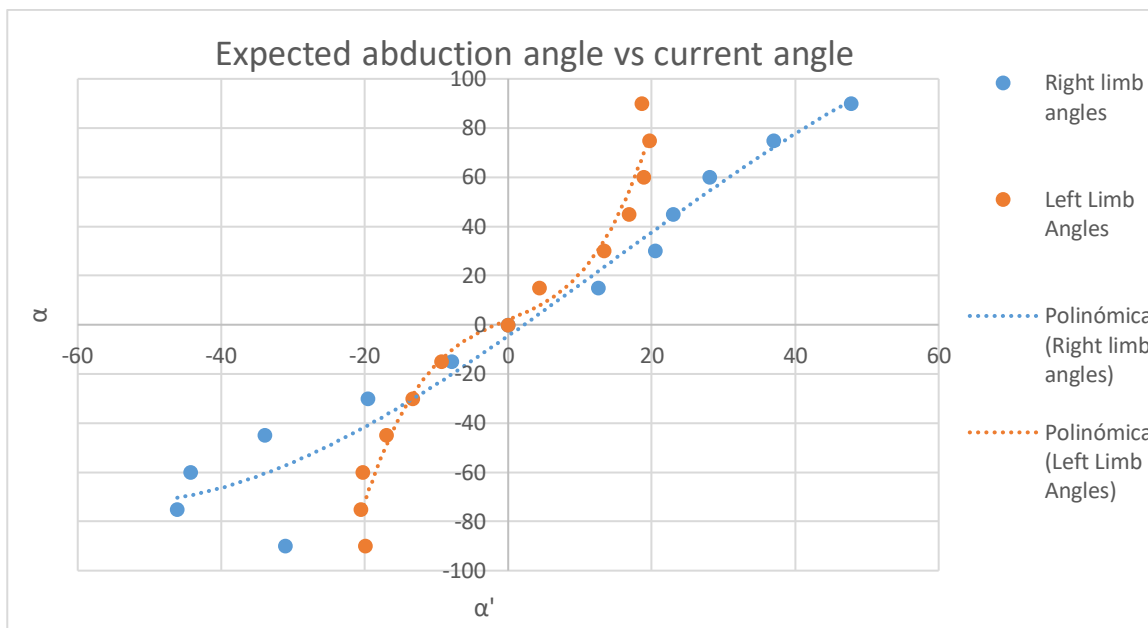


Figure 8.10. Right Limb Angles Calibration Equation: $y = -0,0001x^3 + 0,0064x^2 + 2,0394x - 4,533$.
 Left Limb Angles Calibration Equation: $y = 0,0065x^3 + 0,004x^2 + 1,1976x + 2,0219$.

To sum up, Unity's math system to calculate angles in three-dimensional space regarding the avatar do behave complexly and could behave even more by adding the Z axis. Moreover, developing a calibration equation could lead into unoptimized scripts.

8.3.2. Detection Methodology

After considering all constraints, it is concluded that the first approach to developing the experience by detecting the angles between hand and shoulder is not feasible. In order to execute the experience with ease, the use of *Colliders* and *Triggers* has also been considered.

A *Collider* is a component that gives shape to an object by adding a *Mesh*, which is generally an approximation to the object structure and is invisible during the gameplay. *Colliders* recognize when an object gets collisions and collides. *Triggers* are similar to *Colliders*, but they do not collide.

A *Collider* is needed to create a *Trigger*. In Unity, a *Mesh Collider* component needs to be added to a *GameObject*; once a *Mesh* is created, it needs to be referred. Initially, there are pure forms as cubes and spheres, but several *Meshes* can be imported into the game, as shown in Figure 8.11.

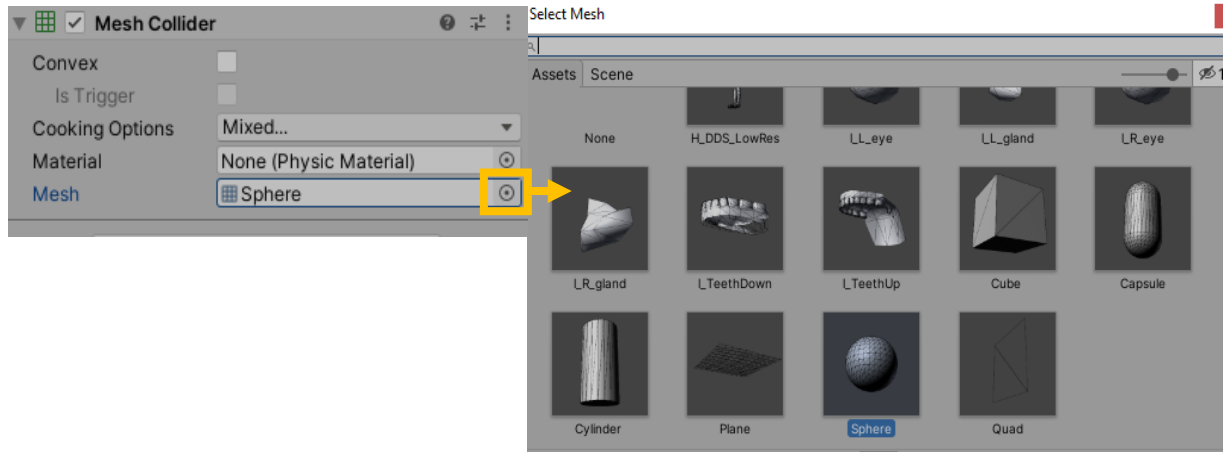


Figure 8.11. Mesh Collider Interface

Once a *Mesh* is chosen, the *Collider* can act as a *Collider* itself or as a *Trigger* by checking the property: *Convex* -> *Is Trigger*.

Consequently, as *Triggers* need an object with a *Collider* function in order to work with, both hands have their *Mesh Colliders*, which will detect the collisions with *Triggers*.

Next step is creating the necessary *Triggers* to allow the user to flap and glide, to do so, three *Triggers* are positioned on each side, six in total: an upper *Trigger*, a mid *Trigger* and a lower one, all of these called "Limits" as shown in the Figure 8.12.

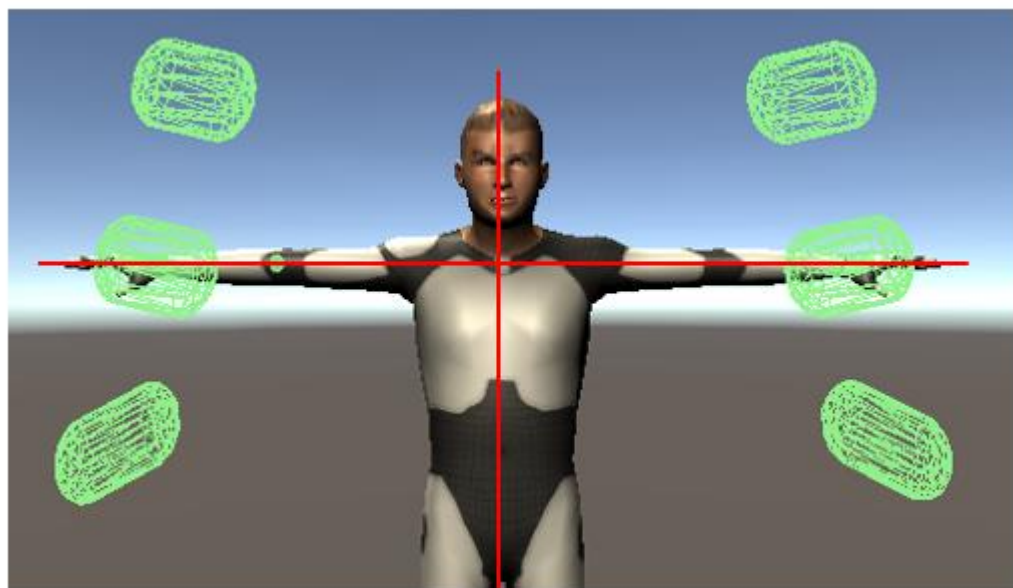


Figure 8.12. Limits Placement in Unity Interface

Upper Limit and Lower Limit are positioned 54° over and below the arm respectively with $\pm 9^\circ$ of tolerance to the user's movement, so a minimum of 45° angle is required to move. And then the mid limit is at 0° with the same tolerance of $\pm 9^\circ$. The tolerance implied into these *Triggers* is three-dimensional so even on the Z axis is allowed some movement, precisely $\pm 9^\circ$ on the Z axis, so the user does not need to execute movements precisely on an (X, Y, 0) plane.

Lastly, all *Triggers* have been tagged to detect them internally. Figure 8.13 shows how to create a *Trigger* tag. The detection of each limit is unique in order to code it later in a script called *Pose* ([Annex A5.1](#)).

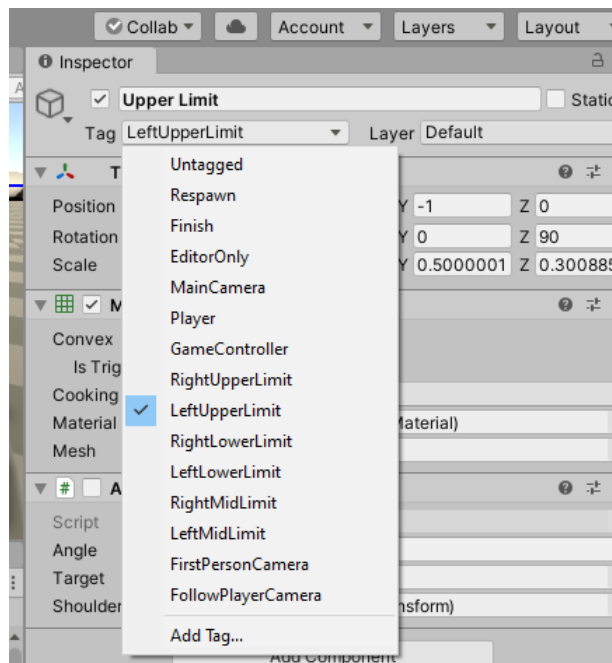


Figure 8.13. Limit's Tags

The *Triggers* are scripted so they do follow the rotation of the hand, it means that regarding that user may not be able to move vertically its arms, it can still do it horizontal or just to try new moves following a vertical local movement getting the hand as a reference.

Once all *Triggers* are set up, *Targets* must be scripted in order to return an answer to their collision. To do it, there are three main functions in C# used:

- *OnTriggerEnter*: Check if a *GameObject* with a *Mesh Collider* has collided with the *Trigger Mesh* and occurs within the first frame.
- *OnTriggerStay*: Check if a *GameObject* has intersected in the previous frame and in the actual frame with a *TriggerObject*.

- *OnTriggerExit*: Checks if the *GameObject* previously entered or stayed has exit the *Mesh Trigger's* zone.

With all these functions established, the script *Pose* has been developed (see more detail in [Annex A5.1](#)). The conditions stated for each movement in *Pose* script are shown in Table 8.3 below.

LIMITS	FLAP	GLIDE	GLIDE LEFT	GLIDE RIGHT
UPPER RIGHT	√*		√	
MID RIGHT		√		
LOWER RIGHT	√*			√
UPPER LEFT	√*			√
MID LEFT		√		
LOWER LEFT	√*		√	

Table 8.3. Limits Requirement for Each Move

*Once entered it is checked if Lower/Upper Limbs have been entered, if not it maintains a variable to storage that upper/lower limits have been activated.

8.4. Avatar Animation

Animations have been used throughout the development of the project. Their purpose is to create a greater sense of immersion, so that the user feels himself moving within the game, along with the avatar.

8.4.1. Animations in Unity

Unity allows the developer to create his own animations or to manage the animations of important models:

- **Own animations:** they consist in customizing one or several *GameObjects' Transform* parameters: position, scale and rotation. These adjustments can be made from the *Animation window*, specifying keyframes, and they will take place during a defined period of time, to end

up conforming the animation. In addition to the *Transform*, some other *GameObject* components can be modified, such as the *Mesh Renderer* or the *RigidBody*. An example of an animation could be a dragon moving its flying wings, repeating the movement continuously.

- **Animations built into imported models:** these animations are usually associated with a name, such as Idle, Walk, Fly, etc.

Each of the animations conform what is called an Animation Clip in Unity.

The first step to manage *GameObject* animations is to create an Animator Controller and associate it with the animated *GameObject*. Thus, from the Animator window, the developer can create a 'State Machine' that organizes the animations into a structured system like a flowchart. In the 'State Machine' the state of the animated *GameObject* defines which animation takes place at any given time. The status of the *GameObject* will vary according to the parameters defined in the game. The Blend Tree is used to mix animations in Unity, mixing animations according to various parameters, such as speed. In each state can be found the animations that are being used to mix with each other and generate movement.

Unity also has an Avatar System, where humanoid characters are assigned to a common internal format.

Each of these pieces, the Animation Clips, the Animator Controller and the Avatar are grouped into a *GameObject* using the Animator Component. This component has a reference to an Animator Controller and (if required) to the Avatar of that model. At the same time, the Animator Controller contains references to the Animation Clips it uses (45). The Animator Component also has a parameter named Root Motion. This parameter can determine the animation to produce a movement that will cause the X, Y, Z axes of the *GameObject* to change. If the animation should not affect the axes, this parameter must be unchecked.

8.4.2. Animations in the Project

Therefore, in this project, the animations have been imported from other models from the Unity Asset Store and readjusted. An Asset called *3rd Person Controller + Fly Mode* has specifically been used, which contains animations for locomotion, jump and flight, among others. Flight animations (Figure 37), which met the requirements for the project, were imported from this Asset and were subsequently modified.

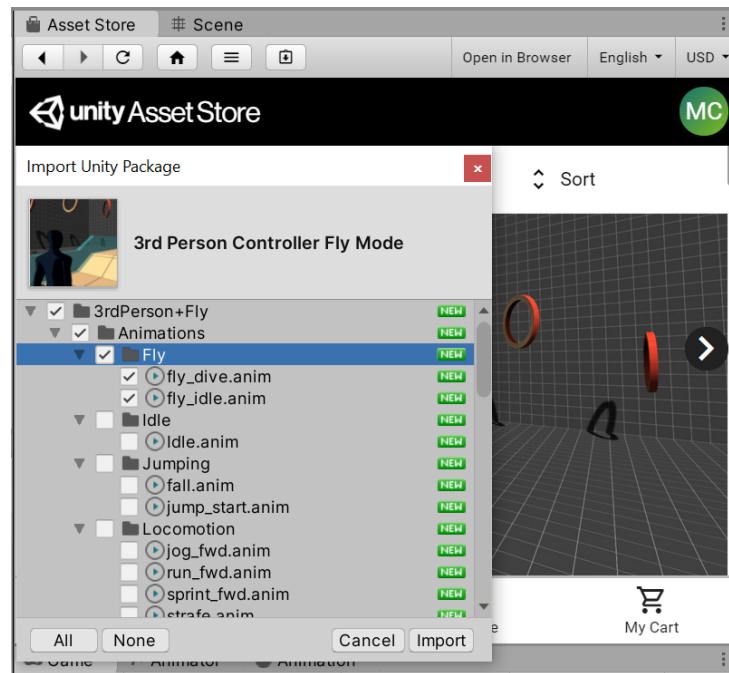


Figure 8.14. Animations Imported from "3rd Person Controller + Fly Mode" Asset in Unity

Fly Idle and *Fly Dive* are the two imported animations.

- *Fly Idle*: consists of a static and cyclic animation that gives a small movement to the character when it is standing in an upright position, so that it is more realistic. In order to repeat the cyclic animation continuously, it must be ensured that the *Loop Time* property is selected.
- *Fly Dive*: consists of a flight animation in which the character is lying.

At this point animations are imported into the Unity project. Then, it is essential to ensure that the character is humanoid. This has to be defined in the Rig section of the character model used for the project. Now the option "Create From This Model" can be selected to create an Avatar from the model (Figure 8.15).

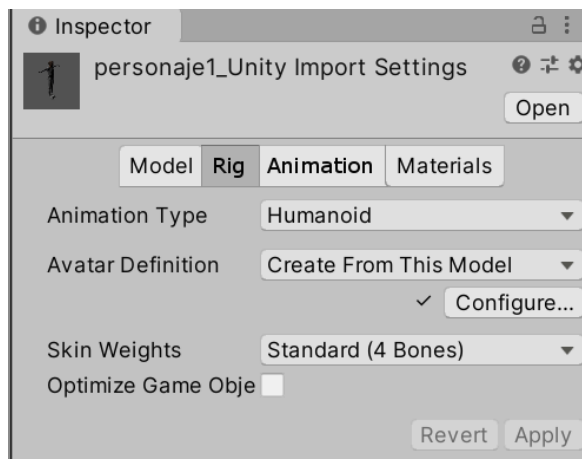


Figure 8.15. Creating an Avatar from a Character's Model

It is now necessary to manage the animations, which means that an Animator Controller must be created, as mentioned in [Section 8.4.1](#). The Animator Controller must be associated with a GameObject created in the Scene, the character. This configuration will allow the developer to control the State Machine, that is, it allows to control when the character has to be in one animation and when in another.

An Animator Component is then added to the character from the Inspector window, and this component is associated with the Animator Controller through the Controller property. The Avatar created previously from the character model has to be dragged into the Avatar property of the Animator Component. The Animator Component is shown in Figure 8.16.

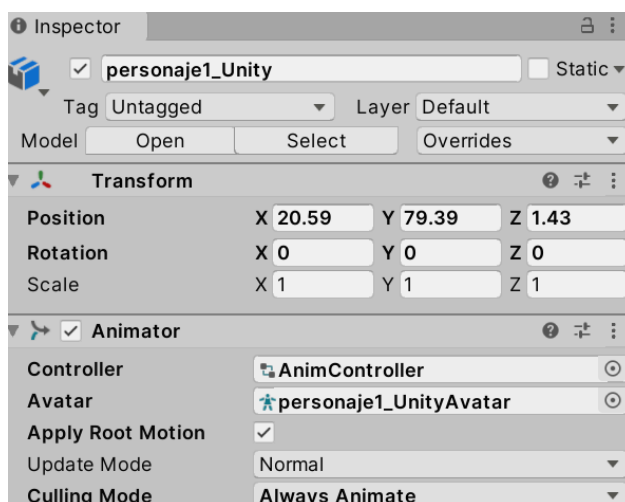


Figure 8.16. Animator Component

Once the appropriate settings are defined, the two imported animations have been modified from the Animation window, selecting any body part or character's Component that needed to be adjusted. The

modification can be carried out using keyframes (marks that indicate from which second to which second the adjustments are registered) or using the recording button from the same Animation window (Figure 40). When using the recording button, the necessary components can be modified directly from the Unity Scene. These amendments get recorded as an animation.

The Fly Dive animation has been modified and renamed as Fly Forward. Changes have been made modifying the Transform of the character's arms and head so that, during the flight, the arms are T-shaped, and the head looks down towards the environment and slightly forward. In the same way, the Mesh of the character ([Annex A.4.1](#)), along with the *Colliders* used to detect the movements, have also been positioned according to the lying character.

Besides, this flight animation has been duplicated to create two more animations (Fly Left and Fly Right) that provide a sense of body rotation, when turning right or left during flight. To create these animations, some parameters have been manipulated, such as the rotation of the character's body and head, and the position of its arms.

In the Fly Idle Animation, only the arms position has been modified.

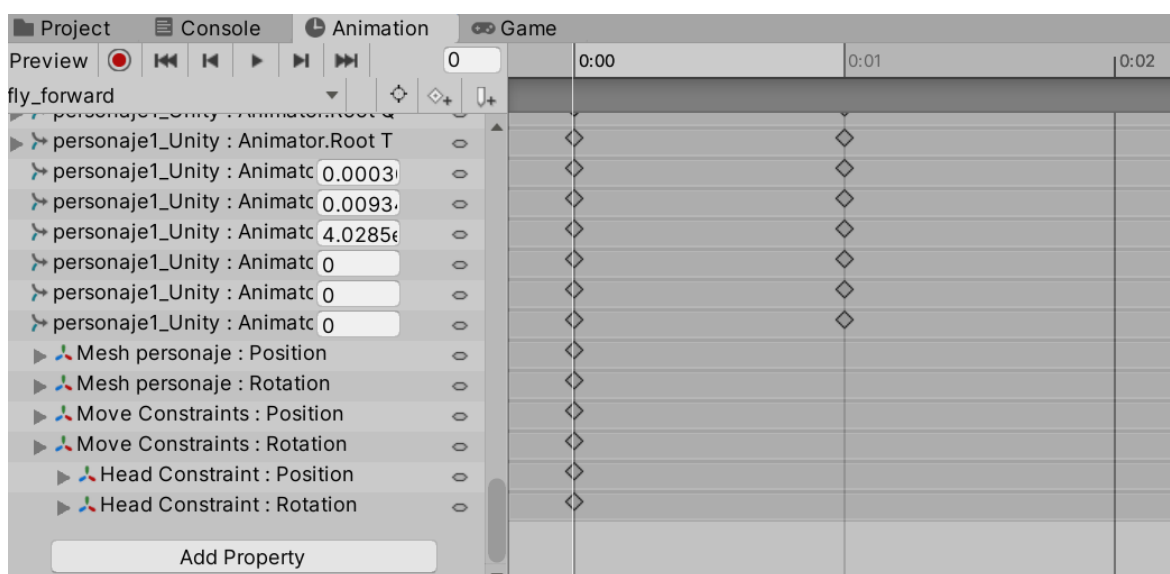


Figure 8.17. Unity's Animation Window

Once the appropriate settings and the necessary animations are set, the parameters that will give way to the different animations from the State Machine should be configured.

From the Animator window, the State Machine allows to associate an animation with each state and define various types of conditions in order to move from one state to another. Once all the animations to be used are available as states within the State Machine, the necessary parameters are created to condition whether the character is in one state or another. These parameters are introduced as a

condition into the transitions (white arrows) that relate the different states, thus restricting the passage from one state to another.

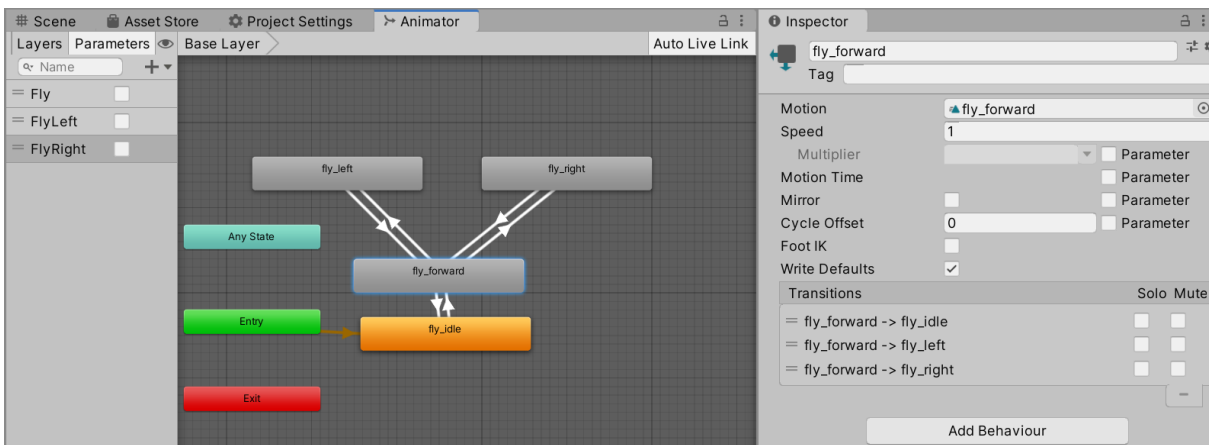


Figure 8.18. Unity's Animator Window

Just as shown in Figure 8.18, in this project, three Boolean parameters have been defined: Fly, Fly Left and Fly Right. These parameters have been introduced into the transitions between the Fly Idle, Fly Forward, Fly Left and Fly Right animations, and have been controlled by a script (see in more detail [Annex A5.2](#)). Table 10 shows how the conditions give way to different animations.

ANIMATION STATE	CONDITION	PREVIOUS CONDITION
FLY IDLE	Fly FALSE	None
FLY FORWARD	Fly TRUE	None
	Fly Left FALSE	Fly TRUE
	Fly Right FALSE	Fly TRUE
FLY LEFT	Fly Left TRUE	Fly TRUE
FLY RIGHT	Fly Right TRUE	Fly TRUE

Table 8.4. Conditions for the Animations

8.5. Movement Translation in Unity

Once movement detection was defined in Unity, the motion of the avatar in the virtual world has been established. A flow diagram (Figure 8.19) is designed in this section in order to define the motion of the avatar.

A flow diagram is a graphic representation of how actions flow within a program. It is used to solve a problem in a structured way. Basic symbols to create flow diagrams are listed below in Table 8.5.









BASIC SYMBOLS	
	Start / End of the program
	Input / Output
	Internal process, operation or activity
	Decision making
	Connector
	Logical AND
	Logical OR
	Program flow

Table 8.5. Flow Diagram's Basic Symbols

The following diagram defines what happens to the avatar once movement from the user is detected, thus defining the avatar's motion depending on the movements of the user in the real world.

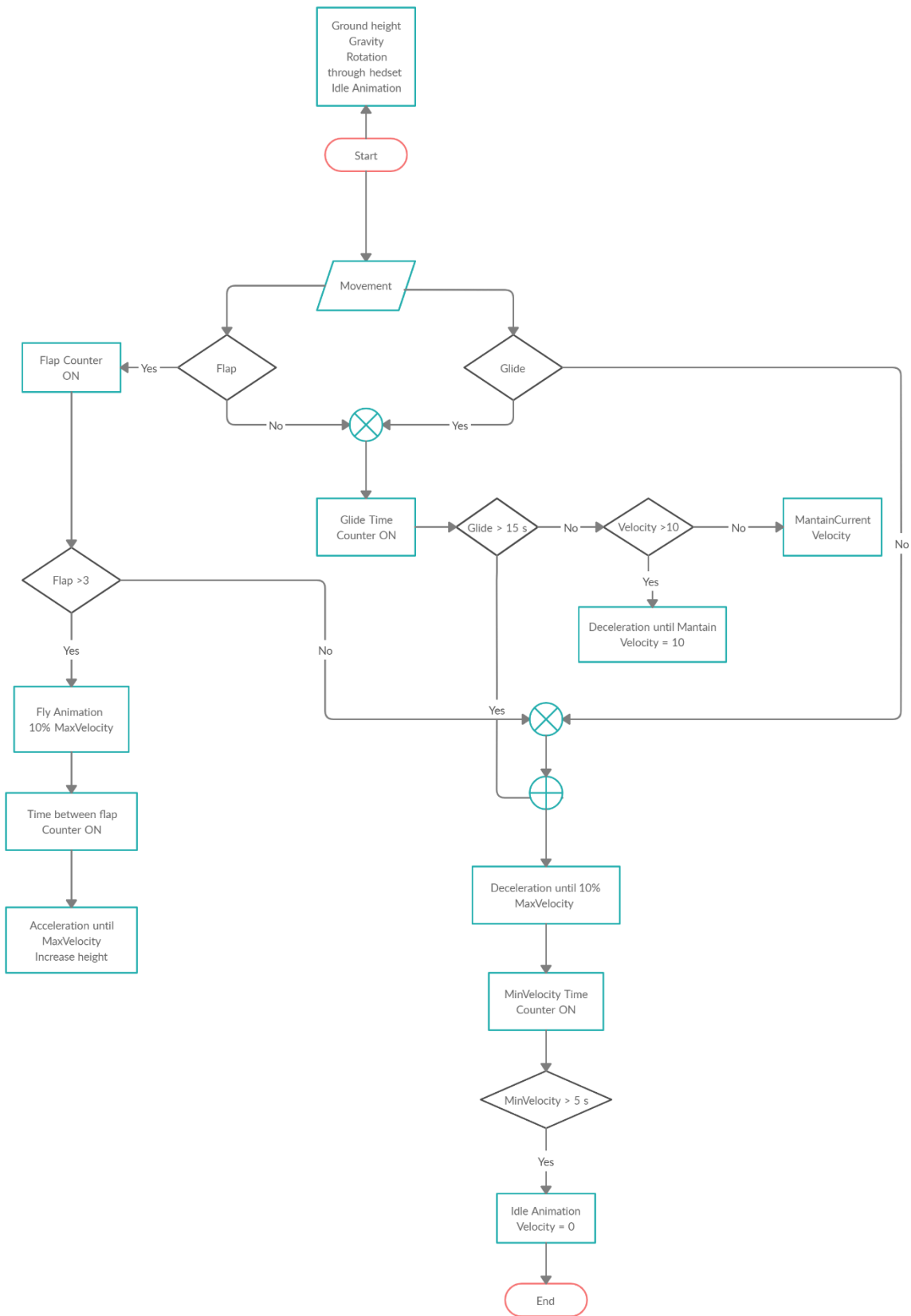


Figure 8.19. Motion of the Avatar in Unity

At the start of the game, rotation movement is defined through the headset position, and it is also defined that the avatar must initiate on ground height and in an upright position (Idle Animation).

When the user starts to flap, a Flap Counter is activated. If flapping is detected more than three times, the Fly Animation starts, provoking the avatar to lay down and see the surroundings from a top view. In addition to the Fly Animation lied down status, an of the avatar, and audio clip -, recorded from a butterfly flap - is played to enhance the sense of immersion while flying ([Annex A4.2](#)).

In this context, the avatar starts moving forward with ten per cent of the maximum speed (previously defined) and will start accelerating and increasing height. Once the maximum speed set by default is reached, another counter is also activated to detect time between flaps and also a new audio clip is played to increase the sense of speed ([Annex A4.2](#).) Depending on the time between detected flap movements, the maximum speed will increase or not, following a twenty per cent increase on the maximum speed. For example, if the user performs the flap movement again in less than one second, the maximum velocity will increase to twice the previous maximum velocity.

However, when the user is gliding instead of flapping, a Glide Time Counter starts, and the avatar maintains its speed when it is lower than 10. If, on the contrary, the speed exceeds 10, it is automatically decreased to 10 and maintained in this figure.

When gliding time measured by the Glide Time Counter is more than 15 seconds, the avatar starts decelerating to ten per cent of the maximum speed. Deceleration also happens when the user is not gliding and flapping has not been detected more than 3 times.

Finally, when the avatar reaches the minimum speed and it lasts more than 5 seconds, the velocity turns zero and the avatar goes back to an upright position (Idle Animation).

8.6. Alternative Tools to Design in Virtual Reality

Due to the global pandemic of Covid-19, the major part of the design was done at home, without Virtual Reality equipment. Nevertheless, HTC Vive controllers' movement and HTC Vive headset were simulated approximately.

8.6.1. Simulate HTC Vive Headset

An alternative to the HTC Vive headset has been found in order to test and recreate the feelings experienced while wearing the headset, so that motion blur and other issues regarding Virtual Reality could be avoided. This alternative is an external software called *RiftCat* (53) that allows to simulate the HTC Vive headset using an android phone. There are various reasons why this software has been used

instead of using settings that allow to develop the app for an android phone. The first reason is that the software works along with SteamVR. *Riftcat* also offers the possibility of working with controllers by using other Android phones and their accelerometers and gyroscopes. However, this last consideration was not executed due to the lack of accuracy of these controllers and the lack of extension range, as it makes impossible to execute the movements of this project. Lastly, with this software in the computer and the *VRidge* app for the Android phone, the experience is almost the same.

This alternative tool demands a setup and some requirements for the computer, and also for the phone (Table 8.6), in order to have a similar experience to the one offered by the real headset.

COMPONENT	MINIMUM REQUIREMENTS	RECOMMENDED REQUIREMENTS
OS	Windows 8.1(AMD) or Windows 7(Nvidia)	Windows 10
PROCESSOR	I5-2500 or more	I5-4590 or more
RAM	4GB or more	8GB or more
GRAPHIC CARD	NVidia GeForce GTX 650 or superior with NVENC support/ AMD Radeon 7750 or superior with the VCE 1.0+ support	NVidia GeForce GTX 970 or superior with NVENC support/ AMD Radeon 290 or superior with the VCE 1.0+ support
MOBILE PHONE	Android 5.0+ o iOS 12.1+ smartphone	Smartphone with Android 5.0+ or iOS 12.1+ and gyroscope
VISOR CARDBOARD	Any compatible device	Any compatible device
CONNECTION	2.4GHz WiFi router or USB 2.0 wire.	5 GHz WiFi router or USB 3.0 wire.

Table 8.6. RiftCat Requirements (53)

8.6.2. Simulate Virtual Reality with Controllers

In order to recreate the flapping movement of the user through the avatar's arms, the idea was to use two mobile phones simulating the controllers, as they have sensors such as gyroscopes or

accelerometers, among others. The intention was to carry out the recreation using the same program used to replicate the headset (*RiftCat*), by downloading an SDK to the mobile phones. However, the result was not very accurate. The controllers were shown in front of the avatar but did not allow to perform the abduction movement previously defined as the base movement of the flap.

At this point, it was decided to change the strategy and try to simulate the movement of the user's arms using a gamepad. This element was implemented into Unity by means of its Input Manager ([Annex A3](#)).

After knowing the mapping of the Gamepads to be used (Figures 8.20 and 8.21), the axes of the joysticks were employed to simulate the arms movement in two dimensions. It is obvious that some degrees of freedom were lost, but the two dimensions of the joystick allowed to simulate the arm abduction movement, as it only involves the X and Y axes, in case it is a perfect movement.

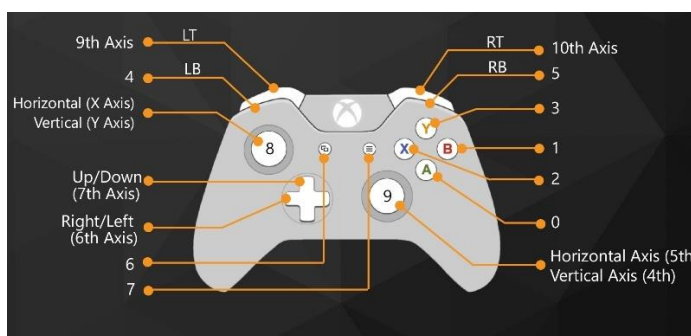


Figure 8.20. Xbox One Controller Mapping for Unity



Figure 8.21. PlayStation 4 Controller Mapping for Unity

Each joystick simulates the movement of an arm, using the Targets on the hands ([Section 8.2](#)) to assign each arm a joystick. Each Target contains a script called *Controller* ([Annex A5.3](#)). This script allows the

player to choose both the Gamepad that will be used and the joystick that will control the chosen Target and, therefore, the respective arm. As shown in Figure 45, the left joystick of the Xbox controller is being used to control the avatar's left arm from its Target.

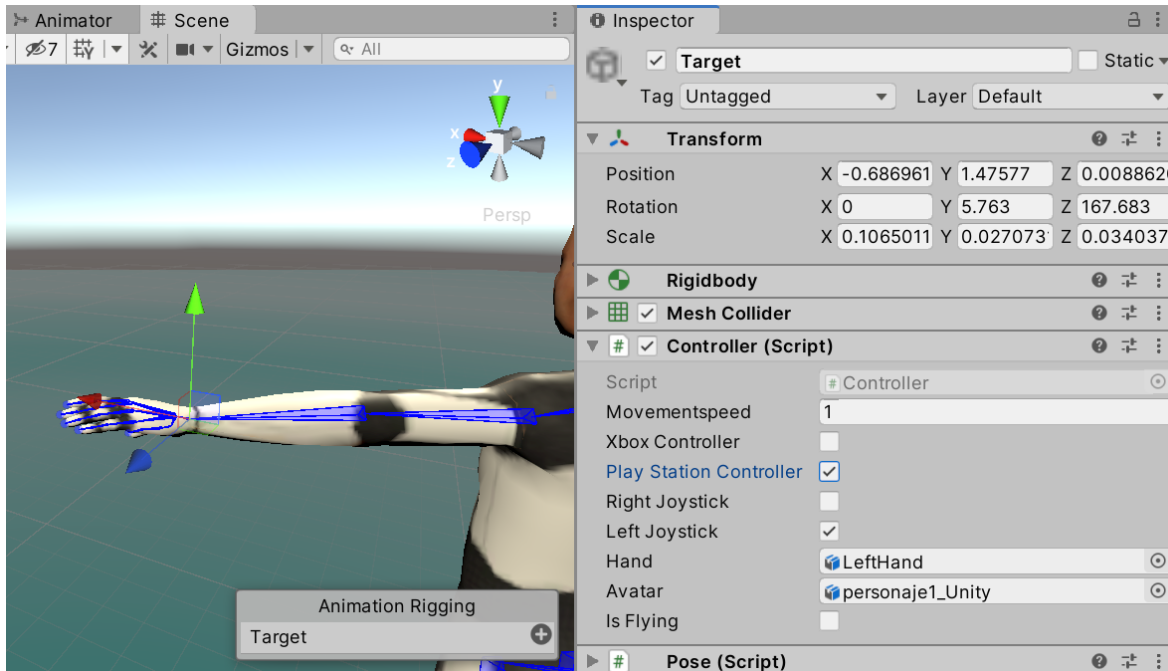


Figure 8.22. Controller Script on Left Target

9. Application Performance

In order to explain how the application operates, the first thing that has been done is to create a use case diagram to understand both the responses of the application to a user's interaction and the goals of these system-user interactions.

9.1. Use of Case Diagram

A use case diagram is a sequence of interests between a system and its actors, in response to an event that initiates a major actor on the same system (54). It describes both the system's provided features and how / by whom they are executed, and the functionality of the system from the user's perspective.

The basic elements employed to represent use case diagrams are shown in Figure 9.1.

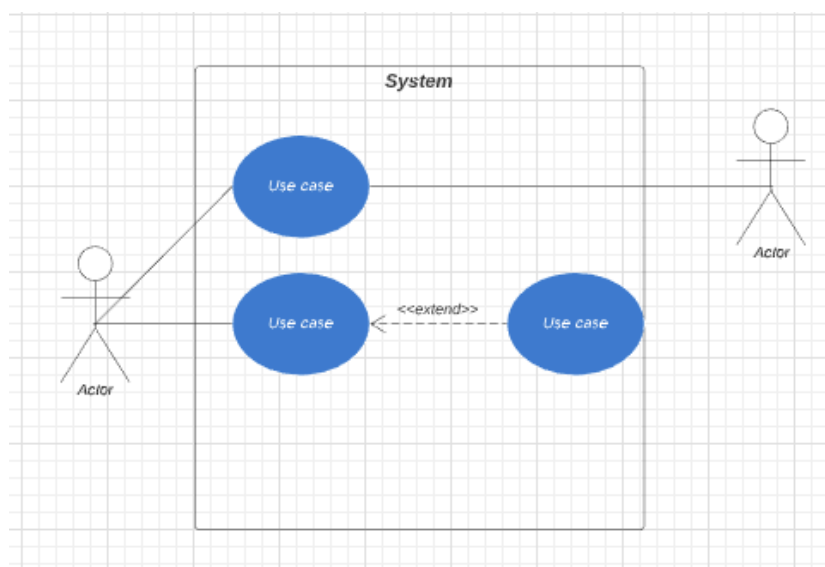


Figure 9.1. Use Case Diagram Basic Elements

- **System:** what you are developing, either a business process, a software or an application. It is represented by a rectangle. Everything inside the rectangle goes into the system. This project's system is a first-person flight module of a VR application.
- **Actor:** someone or something that uses the system to achieve a goal: person, organization, another system or an external device. Actors must be placed outside the system, as they are external objects. Each actor must interact with at least one of the use cases within the system. There are two types of actors:

- Primary actor: initiates the use of the system and should be placed on the left-hand side of the system. In this case, the primary actor of the use case is an older adult living in a nursing home.
- Secondary actor: it is reactionary and should be placed on the right-hand side of the system.
- **Use case:** represents an action that accomplishes some kind of task within the system. It must be placed inside the rectangle because actions occur within the system.
- **Communication relationships:** relationships between the actor and a use case. There are four basic types of relationship explained in Table 9.1.

COMMUNICATION RELATIONSHIPS


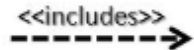


	ASSOCIATION	means a basic communication or interaction.
	INCLUDE	shows dependency between a base use case and an included use case. Every time the base case is executed, the included one is executed as well. The base use case requires an included use case in order to be complete.
	EXTEND	gives the base use case the option to extend its behaviour. When the base use case is executed, the extend use case will happen only if certain criteria are met.
	GENERALIZATION	relationship between general use cases (parent) and specialized use cases (children). Children share the common behaviour of the parent, but each child adds something more on his own.

Table 9.1. Basic Communication Relationships in Use Case Diagrams

In this project, the use case diagram of the first-person flight module represents the interaction between the user and the interface of the VR application, and it is shown in Figure 9.2.

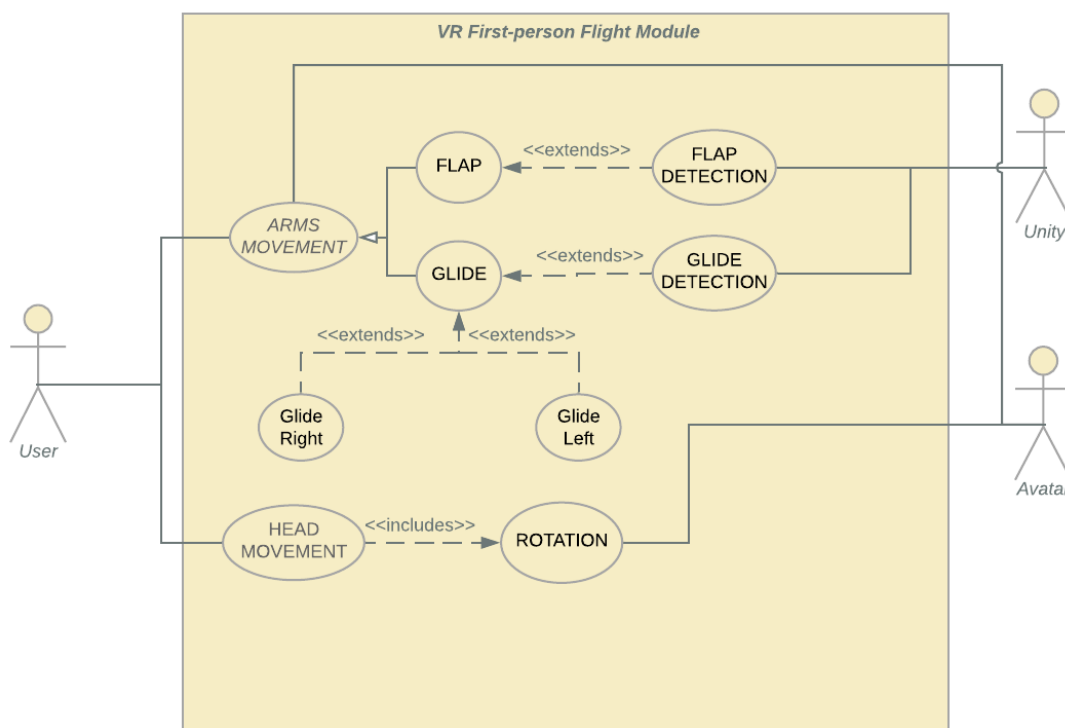


Figure 9.2. Use Case Diagram

As mentioned in [Section 8.6](#), alternatives to the HTC Vive system had to be found, as it could not be accessed. Because of these alternatives, some important variations in the operation of the application have to be considered in this section:

- The avatar’s rotation is not achieved through the simple movement of the user's head because the software used to recreate the Virtual Reality headset is not entirely accurate-. It needs constant readjustment. The left joystick of a Gamepad is used to achieve character rotation and to have a 360º view of the environment.
- The movement of the arms is no longer performed by the user, but it is operated through the joysticks of a Gamepad. To recreate the joysticks’ movement on the avatar's arms, the R1 and L1 buttons on the Gamepad must be held down. Otherwise, the left joystick settings remain for rotation.

Given these variations, the use case diagram representing the interaction between the user and the interface is represented in Figure 9.3.

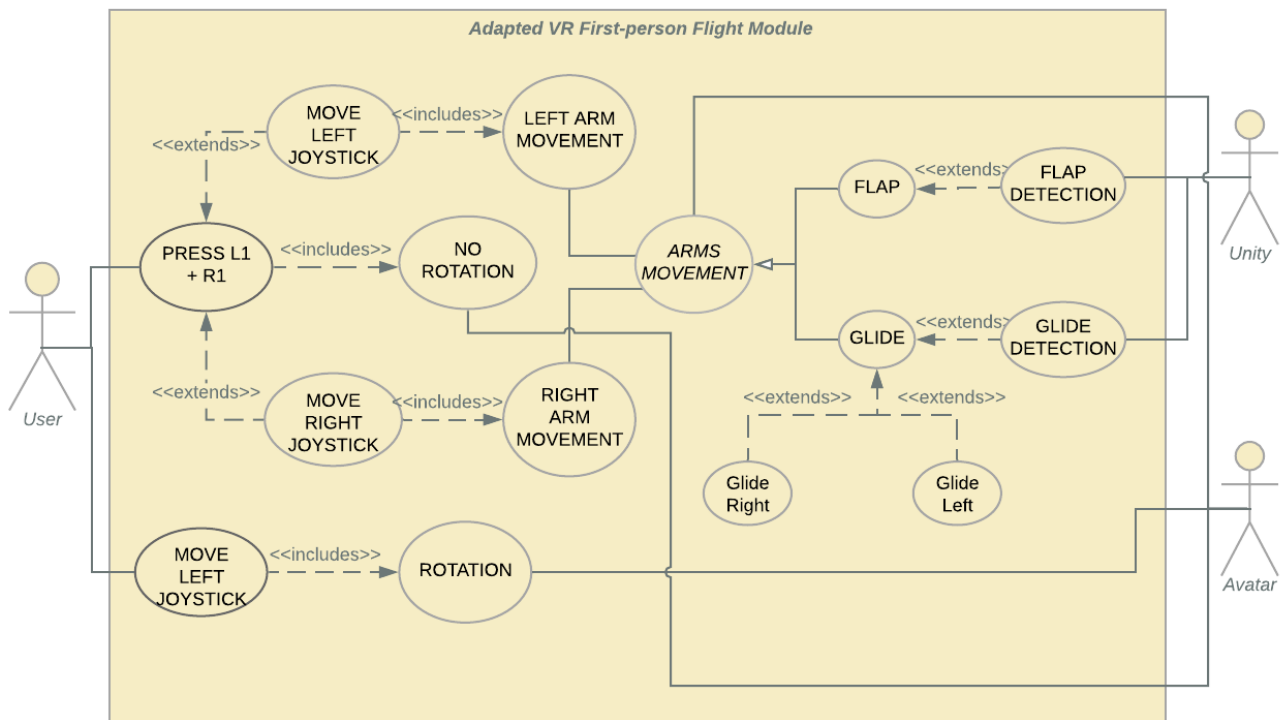


Figure 9.3. Adapted Use Case Diagram

9.2. Use of Case Specification

A use case can also take the format of a structured textual specification, and thus capture the specific details of a use case (55). A *use case diagram* only shows the name of each use case, with no additional information, whereas a *use case specification* details each of the use cases defined.

The use case specification of the previous diagrams is performed in this section. The details specified for each use case are explained below:

- **Actor(s):** name of the actor/actors who interact with the use case.
- **Summary Description:** brief description of the use case. It should include the functionality of the use case and the value added for the user who activates it.
- **Pre-condition:** condition that must be met in order to execute the use case.
- **Basic Path:** description of the interactions' scenario between the actor/s and the use cases.
- **Post-condition:** condition that must be met before the use case is considered done.
- **Related Use Cases:** other use cases that have a relationship with the one described.

The use case specification of the first-person flight module is shown below in two different sections: first the details of the use case diagram in Figure 9.2 are specified, and finally those of the second use case diagram, adapted to the variations (Figure 9.3).

9.2.1. VR First-person Flight Module

In the following section there is a specification of those use cases (UC) directly related to the user that appear in the use case diagram of the VR First-person Flight Module system (Figure 9.2).

USE CASE (UC1)	HEAD MOVEMENT
ACTOR(S)	Primary Actor: User Secondary Actor: Avatar
SUMMARY DESCRIPTION	Move the head around
PRE-CONDITION	Start the application and use HTC Vive controllers
BASIC PATH	The user performs head movement
POST-CONDITION	Includes rotation of the avatar based on the movement of the user's head
USE CASES RELATED	ROTATION

Table 9.2. UC1 Specification

USE CASE (UC2)	ARMS MOVEMENT
ACTOR(S)	Primary Actor: User Secondary Actor: Avatar
SUMMARY DESCRIPTION	Move the arms
PRE-CONDITION	Start the application and use HTC Vive controllers
BASIC PATH	The user moves the arms and so does the avatar in real time
POST-CONDITION	Perform a predefined movement

USE CASES RELATED

FLAP / GLIDE

*Table 9.3. UC2 Specification***USE CASE (UC3)**

FLAP

ACTOR(S)

Primary Actor: User

Secondary Actor: Unity

SUMMARY DESCRIPTION

Do a predefined movement: Flap

PRE-CONDITION

Arms movement

BASIC PATH

The user performs flap movement

POST-CONDITIONUnity *Triggers* detect the Flap movement**USE CASES RELATED**

ARMS MOVEMENT / FLAP DETECTION

*Table 9.4. UC3 Specification***USE CASE (UC4)**

GLIDE

ACTOR(S)

Primary Actor: User

Secondary Actor: Unity

SUMMARY DESCRIPTION

Do a predefined movement: Glide

PRE-CONDITION

Arms movement

BASIC PATH

The user performs glide movement. This movement can be classified in: Glide, Glide Left or Glide Right

POST-CONDITIONUnity *Triggers* detect the Glide movement**USE CASES RELATED**

ARMS MOVEMENT / GLIDE RIGHT / GLIDE LEFT / GLIDE DETECTION

Table 9.5. UC4 Specification

9.2.2. Adapted VR First-person Flight Module

In this section there is a specification of those use cases directly related to the user that appear in the use case diagram of the Adapted VR First-person Flight Module system (Figure 9.3).

VARIED USE CASE (UC1)	MOVE LEFT JOYSTICK
ACTOR(S)	Primary Actor: User Secondary Actor: Avatar
SUMMARY DESCRIPTION	Move the left joystick of the Gamepad
PRE-CONDITION	Start the application and use a Gamepad
BASIC PATH	The user moves the left joystick of the Gamepad
POST-CONDITION	Includes rotation of the avatar based on the movement of the left joystick
USE CASES RELATED	ROTATION

Table 9.6. Varied UC1 Specification

VARIED USE CASE (UC2)	PRESS L1+R1
ACTOR(S)	Primary Actor: User
SUMMARY DESCRIPTION	Press L1 and R1 buttons of the Gamepad
PRE-CONDITION	Start the application and use a Gamepad
BASIC PATH	The user presses L1 and R1 buttons of the Gamepad
POST-CONDITION	Move Right Joystick, Left Joystick or both Includes NO rotation of the avatar based on the movement of the left joystick
USE CASES RELATED	MOVE LEFT JOYSTICK / MOVE RIGHT JOYSTICK / NO ROTATION

Table 9.7. Varied UC2 Specification

VARIED USE CASE (UC3)	MOVE LEFT JOYSTICK
ACTOR(S)	Primary Actor: User Secondary Actor: Avatar
SUMMARY DESCRIPTION	Move the left joystick of the Gamepad
PRE-CONDITION	L1 + R1 buttons pressed
BASIC PATH	The user uses the left joystick to perform movement on the left arm of the avatar in real time
POST-CONDITION	Includes avatar's left arm movement
USE CASES RELATED	PRESS L1 + R1 / LEFT ARM MOVEMENT

Table 9.8. Varied UC3 Specification

VARIED USE CASE (UC4)	MOVE RIGHT JOYSTICK
ACTOR(S)	Primary Actor: User Secondary Actor: Avatar
SUMMARY DESCRIPTION	Move the right joystick of the Gamepad
PRE-CONDITION	L1 + R1 buttons pressed
BASIC PATH	The user uses the right joystick to perform movement on the right arm of the avatar in real time
POST-CONDITION	Includes avatar's right arm movement
USE CASES RELATED	PRESS L1 + R1 / RIGHT ARM MOVEMENT

Table 9.9. Varied UC4 Specification

VARIED USE CASE (UC5)	ARMS MOVEMENT
ACTOR(S)	Avatar

SUMMARY DESCRIPTION	Move the arms of the avatar
PRE-CONDITION	Move left, right or both joysticks of the Gamepad
BASIC PATH	The user moves avatar’s arms in real time by moving the left and right joysticks of the gamepad
POST-CONDITION	Perform a predefined movement
USE CASES RELATED	LEFT ARM MOVEMENT / RIGHT ARM MOVEMENT / FLAP / GLIDE

Table 9.10. Varied UC5 Specification

VARIED USE CASE (UC6)	FLAP
ACTOR(S)	Primary Actor: Avatar Secondary Actor: Unity
SUMMARY DESCRIPTION	Do a predefined movement: Flap
PRE-CONDITION	Arms movement
BASIC PATH	The avatar performs flap movement
POST-CONDITION	Unity <i>Triggers</i> detect the Flap movement
USE CASES RELATED	ARMS MOVEMENT / FLAP DETECTION

Table 9.11. Varied UC6 Specification

VARIED USE CASE (UC7)	GLIDE
ACTOR(S)	Primary Actor: Avatar Secondary Actor: Unity
SUMMARY DESCRIPTION	Do a predefined movement: Glide
PRE-CONDITION	Arms movement

BASIC PATH	The avatar performs glide movement. This movement can be classified in: Glide, Glide Left or Glide Right
POST-CONDITION	Unity <i>Triggers</i> detect the Glide movement
USE CASES RELATED	ARMS MOVEMENT / GLIDE RIGHT / GLIDE LEFT / GLIDE DETECTION

Table 9.12. Varied UC7 Specification

10. Environmental Impact Analysis

The environmental impact of this project is considered to be low. The main source that causes an impact in environment is the electrical energy consumed by all the electronic devices involved in the project. Other sources to considered may be the waste of fuel generated by the Google cardboard provider, when the corresponding tool was sent to each member of the team. Apart from that, there is no other source of energy consumed that could have an environmental impact during the execution of the project. Furthermore, another point to consider is the savings in energy produced by *telerehabilitation* in this concept, as this project in a future allows patients to take their rehabilitation routine from home, the fuel consumption involving transport from living place to the rehab location can be saved as the data needed to foresee the evolution of the patient can be done via Internet. Moreover, another impact considered besides the energy consumption is the ability of freeing up in health care centers(56)(57).

Apart from energy consumption, it has been considered that the only possible electronic waste derived from this project is caused by the useful life of each device, in case they are not taken to any recycling point. Under this consideration, plastic waste from each device, as well as heavy metals contained within all electronic circuits, may impact negatively on the environment by increasing pollution (58).

Despite this last consideration, the WEEE Directive (Waste Electrical and Electronic Equipment Directive) of the European law states recycling measures for electronic devices (59)(60). This law allows customers to return small electronic devices to electronic related shops with no need to buy any other product. It is also mandatory for all electronic devices to be marked with a symbol, as shown in Figure 10.1, to warn the customer that the device in question cannot go to the normal waste bin. An important indicator is the black line on the bottom of this symbol. This black line indicates that the electronic device was launched to the market after 2005, whereas the absence of the line indicates that the good was manufactured between 2002 and 2005.



Figure 10.1. The WEEE Symbol

All electronic devices of the project have been checked to have this symbol.

11. Improvement Proposals and Future Applications

Intrinsically, this project is part of a bigger one, the root or mother project. In this sense, the project's future application is already known: be part of a rehabilitation application for the elderly. However, it is acknowledged that this application may be used for any other application, in which copying the flying movement is desired.

In addition to the future project, which seeks to contain various modules apart from the flying one, there are still some proposed improvements for this application. These improvements are mostly because this first-person flight module has been developed during the COVID-19 worldwide pandemic, making the development more difficult.

Some of the improvements are listed below:

- Create a calibration event at the beginning of the game, so that the avatar matches the arm's length of each patient, creating a greater sense of immersion.
- Create a discrete interface on the screen, while running the application, in which the patient may see at any time where he is situated in the virtual world. This interface may be a small map of the environment used to ease the location of the patient, similar to the one in Figure 11.1.



Figure 11.1. Minimap of the World in Warcraft Game

- Try to develop an angle detection methodology with VR controllers, instead of player's skeleton, to optimize code.
- Adjust velocity and other parameters according to HTC Vive system to avoid motion blur.
- Optimize textures of the environment and other terrain features to raise the immersion level of the application.

- Give the caregiver or physiotherapist the option of recording each patient's activity and adjusting the movements required (angles), in case the patient suffers of an illness that affects his mobility.

Conclusions

This project consisted in the development of a Virtual Reality application intended for future use in the rehabilitation field. Specifically, for the elderly to improve their performance and engagement in regular physical activity.

Through this application, the elderly may be able to execute a variety of exercises with no need to stay in a clinical environment, thus making this activity more enjoyable. Besides, the application is engaged in nature and sounds with the purpose of improving the patients' feeling about exercise, as well as their well-being.

The import of the natural environment from the previous project has been carried out successfully, adding some sounds related to a flying game and thus enhancing immersion.

Comparing the initially established goals and the results obtained, it can be stated that the main objectives have been achieved. Both the operation of the VR application and the objectives set by Dr Stavros have been successfully accomplished: the avatar moves according to the user's movement in real time, the game starts with ten per cent of the maximum speed and at ground height, and it has acceleration and deceleration mechanisms. In addition, some customization - such as animations - have been introduced, which differentiate whether the user is performing the defined flying movements or not. Thanks to these animations, the user has a top view of the natural environment while performing the flight movements.

The results of this project show that the user's movements are correctly reproduced on the avatar. At first, it was possible to recreate the user's movements on the avatar using the HTC Vive Virtual Reality equipment. However, with the emergence of the COVID-19 pandemic, the strategy had to be changed because this equipment was no longer accessible. From then on, the user's movement was simulated with the joysticks of a Gamepad. This alternative mechanism can simulate the flying action of the user in two dimensions without problems.

Now that the project is concluded, it cannot be said that all goals have been reached, as VR equipment was needed to thoroughly test all steps taken. However, the measures taken are considered to be reliable enough to be able to work with Virtual Reality. Furthermore, the application may still have some range of improvement.

Budget

In this section the economic analysis of the project is presented by calculating the budget required. To calculate the budget of this project,

Direct Cost

Direct costs are divided into salary, social security and the value of the equipment and material needed to develop the project.

For the salary calculation it has been considered that two engineers have worked with a net salary of 25€ per hour, therefore multiplied by the number of hours that have been carried over the project, considering 100 days approximately and a total amount of 6h per day. In addition to the salary in Spain the cost associated with social security is a 23.6% of the Salary, being a total budget of 44,160€ as shown in Table B.1.

SALARY

	Quantity	Time(h)	Cost/hour(€/h)	Cost (€)
ENGINEERS	2	600	25	30,000

SOCIAL SECURITY

	Nº of workers	Salary (€)	Contribution base (%)	Cost (€)
SOCIAL SECURITY	2	30,000	23,60%	14,160
TOTAL				44,160

Table B.1. Salary and S.S Direct Costs

The rest of the direct costs related to the equipment are shown in the following Table B.2.

EQUIPMENT

ITEM	Quantity(ut)	Cost/unit(€/ut)	Cost (€)
MONITOR	1	1,200	1,200
PC	1	400	400
LAPTOP	2	800	1,600
HTC VIVE	1	1,000	1,000
"UNITY 3D"	2	0	0
"VISUAL STUDIO"	2	0	0
"RIFT CAT"	1	15	15
GAME CONTROLLERS	2	60	120
GOOGLE CARDBOARD	2	10	20
TOTAL			4,355

Table B.2. Costs Related to Equipment

In total as direct costs, they raise up to a total of 48,515€.

Indirect Cost

The indirect costs associated to this project are those related to installation and expenses of the corresponding installation shown in following Table B.3.

INSTALLATION EXPENSES

	Months	Cost/month (€)	Cost (€)
RENT	5	600	3,000
INTERNET	5	20	100
LIGHT	5	80	400
WATER	5	40	200
GAS	5	30	150
TOTAL			3,850

Table B.3. Installation Costs

Total Cost

To sum up all costs, the following Table B.4 establishes the total cost of the project.

Concept	Cost (€)
Salary	30,000
Social Security	14,160
Equipment	4,355
Total Direct Costs	48,515
Installations expenses	3,850
Total Indirect Costs	3,850
TOTAL	52,365

Table B.4. Total Costs

Bibliography

1. De Bruin, E. et al. Einsatz der virtuellen realität für das training der motorischen kontrolle bei älteren. einige theoretische überlegungen. A: *Zeitschrift für Gerontologie und Geriatrie*. 2010, Vol. 43, núm. 4, p. 229-234. ISSN 09486704. DOI 10.1007/s00391-010-0124-7.
2. Hwang, J. i Lee, S. The effect of virtual reality program on the cognitive function and balance of the people with mild cognitive impairment. A: *Journal of Physical Therapy Science*. 2017, Vol. 29, núm. 8, p. 1283-1286. ISSN 09155287. DOI 10.1589/jpts.29.1283.
3. *Mynd Virtual Reality | Virtual Reality for Seniors* [en línia]. Disponible a: <https://www.myndvr.com/>.
4. Jones, T., Moore, T. i Choo, J. The impact of virtual reality on chronic pain. A: *PLoS ONE*. 2016, Vol. 11, núm. 12, p. 1-10. ISSN 19326203. DOI 10.1371/journal.pone.0167523.
5. Tashjian, V.C. et al. Virtual Reality for Management of Pain in Hospitalized Patients: Results of a Controlled Trial. A: *JMIR Mental Health*. 2017, Vol. 4, núm. 1, p. e9. ISSN 2368-7959. DOI 10.2196/mental.7387.
6. Li, A. et al. *Virtual reality and pain management: current trends and future directions*. 2011. 2011. DOI 10.2217/pmt.10.15.
7. *Whitepapers, Articles, Books Archives - C*.
8. Yates, M., Kelemen, A. i Sik Lanyi, C. Virtual reality gaming in the rehabilitation of the upper extremities post-stroke. A: *Brain Injury*. Informa Healthcare, 2016, Vol. 30, núm. 7, p. 855-863. ISSN 1362301X. DOI 10.3109/02699052.2016.1144146.
9. Levin, M.F. What is the potential of virtual reality for post-stroke sensorimotor rehabilitation? A: *Expert Review of Neurotherapeutics*. Taylor & Francis, 2020, Vol. 20, núm. 3, p. 195-197. ISSN 17448360. DOI 10.1080/14737175.2020.1727741.
10. Kim, J.H., Thang, N.D. i Kim, T.S. 3-D hand motion tracking and gesture recognition using a data glove. A: *IEEE International Symposium on Industrial Electronics*. 2009, p. 1013-1018. DOI 10.1109/ISIE.2009.5221998.
11. The Franklin Institute. *History of Virtual Reality | The Franklin Institute*.
12. Dom, B. *History of VR - Timeline of Events and Tech Development*. 2018. 2018.
13. *Virtual Reality Systems*. 1993. 1993. DOI 10.1016/c2009-0-22372-5.
14. *Estalagmita - Wikipedia, la enciclopedia libre*.
15. *Oculus Rift - Wikipedia*.
16. *Revit y Realidad Virtual_ un experimento interesante, y hasta ahí*.

17. Wikipedia. *Immersion (virtual reality) - Wikipedia*. 2016. 2016.
18. Tsyktor, V. *What Is Semi-Immersive Virtual Reality?* - CyberPulse. 2019. 2019.
19. *What Is Non-Immersive Virtual Reality?* - CyberPulse.
20. *Process Simulation Software in Oil and Gas Market to Witness Massive Growth, Emerging Technology Research Report with Players Aspen Technology, Inc.*
21. *The Great War told through VR (Virtual Reality)_ the innovative immersive experience of the Museum of Monte San Michele _ WSA.*
22. Elden, M. Implementation and initial assessment of VR for scientific visualisation: Extending Unreal Engine 4 to visualise scientific data on the HTC Vive. A: *MSc Thesis*. 2017, p. 108.
23. Gerardi, M. et al. Virtual Reality Exposure Therapy for Post-Traumatic Stress Disorder and Other Anxiety Disorders. A: *Current psychiatry reports*. 2010, Vol. 12, p. 298-305. DOI 10.1007/s11920-010-0128-4.
24. Rizzo, A. et al. Virtual reality posttraumatic stress disorder (PTSD) exposure therapy results with active duty OIF/OEF service members. A: *International Journal on Disability and Human Development*. 2011, Vol. 10. DOI 10.1515/IJDHD.2011.060.
25. Henderson, A., Korner-Bitensky, N. i Levin, M. Virtual Reality in Stroke Rehabilitation: A Systematic Review of its Effectiveness for Upper Limb Motor Recovery. A: *Topics in Stroke Rehabilitation* [en línia]. Taylor & Francis, 2007, Vol. 14, núm. 2, p. 52-61. DOI 10.1310/tsr1402-52. Disponible a: <https://doi.org/10.1310/tsr1402-52>.
26. Maier, M. et al. Effect of Specific Over Nonspecific VR-Based Rehabilitation on Poststroke Motor Recovery: A Systematic Meta-analysis. A: *Neurorehabilitation and Neural Repair*. 2019, Vol. 33, núm. 2, p. 112-129. ISSN 15526844. DOI 10.1177/1545968318820169.
27. De Rooij, I.J.M., Van de Port, I.G.L. i Meijer, J.-W.G. Effect of Virtual Reality Training on Balance and Gait Ability in Patients With Stroke: Systematic Review and Meta-Analysis. A: *Physical Therapy*. 2016, Vol. 96, núm. 12, p. 1905-1918. ISSN 0031-9023. DOI 10.2522/ptj.20160054.
28. Liu, Q. et al. The Effects of Viewing an Uplifting 360-Degree Video on Emotional Well-Being among Elderly Adults and College Students under Immersive Virtual Reality and Smartphone Conditions. A: *Cyberpsychology, Behavior, and Social Networking*. 2020, Vol. 23, núm. 3, p. 157-164. ISSN 21522723. DOI 10.1089/cyber.2019.0273.
29. Levin, M.F. What is the potential of virtual reality for post-stroke sensorimotor rehabilitation? A: *Expert Review of Neurotherapeutics* [en línia]. Taylor & Francis, 2020, Vol. 20, núm. 3, p. 195-197. ISSN 17448360. DOI 10.1080/14737175.2020.1727741. Disponible a: <https://doi.org/10.1080/14737175.2020.1727741>.
30. Ke, L. et al. Virtual reality for stroke rehabilitation (Review) SUMMARY OF FINDINGS FOR THE MAIN COMPARISON. A: *Virtual Reality for stroke rehabilitation (Review)*. 2015, Vol. 80, núm. 2, p. 57-62. ISSN 03678318. DOI 10.1002/14651858.CD008349.pub4.www.cochranelibrary.com.

31. Turolla, A. et al. Virtual reality for the rehabilitation of the upper limb motor function after stroke: A prospective controlled trial. A: *Journal of NeuroEngineering and Rehabilitation*. Journal of NeuroEngineering and Rehabilitation, 2013, Vol. 10, núm. 1, p. 1. ISSN 17430003. DOI 10.1186/1743-0003-10-85.
32. Huang, Q. et al. Evaluating the effect and mechanism of upper limb motor function recovery induced by immersive virtual-reality-based rehabilitation for subacute stroke subjects: Study protocol for a randomized controlled trial. A: *Trials*. 2019, Vol. 20, núm. 1, p. 1-9. ISSN 17456215. DOI 10.1186/s13063-019-3177-y.
33. Dockx, K. et al. Cochrane Library «Virtual reality for rehabilitation in Parkinson ' s disease». A: *Cochrane database of systematic reviews*. 2016, núm. 12. DOI 10.1002/14651858.CD010760.pub2.www.cochranelibrary.com.
34. Maggio, M.G. et al. Virtual reality in multiple sclerosis rehabilitation: A review on cognitive and motor outcomes. A: *Journal of Clinical Neuroscience*. Elsevier Ltd, 2019, Vol. 65, p. 106-111. ISSN 15322653. DOI 10.1016/j.jocn.2019.03.017.
35. Shema, S.R. et al. Research Report Clinical Experience Using a 5-Week Virtual Reality to Enhance Gait in an. A: . 2014, Vol. 94, núm. 9, p. 1319-1326. DOI 10.2522/ptj.20130305.
36. Howett, D. et al. Differentiation of mild cognitive impairment using an entorhinal cortex-based test of virtual reality navigation. A: *Brain*. 2019, Vol. 142, núm. 6, p. 1751-1766. ISSN 14602156. DOI 10.1093/brain/awz116.
37. Montana, J.I. et al. The Benefits of emotion Regulation Interventions in Virtual Reality for the Improvement of Wellbeing in Adults and Older Adults: A Systematic Review. A: *Journal of Clinical Medicine*. 2020, Vol. 9, núm. 2, p. 500. ISSN 2077-0383. DOI 10.3390/jcm9020500.
38. *HTC Vive, Análisis*.
39. BARNARD, D. *Degrees of Freedom (DoF): 3-DoF vs 6-DoF for VR Headset Selection*. 2019. 2019.
40. VRS. *Virtual Reality Society - Latest Virtual Reality News Headset Reviews* [en línia]. 2018. 2018. Disponible a: <https://www.vrs.org.uk/>.
41. Rvdm88. *HTC Vive Lighthouse Chaperone tracking system Explained*. 2015. 2015.
42. User Guide. A: *SpringerReference*. 2011, DOI 10.1007/springerreference_27988.
43. *SteamVR*.
44. *Unity (game engine) - Wikipedia*.
45. Unity3D. *Unity - Manual: Unity User Manual (5.6)*. 2017. 2017.
46. *Unify Community Wiki*. 2012. 2012.
47. Valve Corporation. *SteamVR Unity Plugin*. 2019. 2019.

48. Microsoft. *Herramientas de desarrollo de juegos de Unity | Visual Studio*.
49. *CÓMO USAR VISUAL STUDIO CON UNITY3D – Artículos y tendencias sobre soluciones tecnológicas*.
50. Microsoft. *Visual Studio 2017 System Requirements | Microsoft Docs*. 2018. 2018.
51. Dube, C. i Tapson, J. Kinematics design and human motion transfer for a humanoid service robot arm. A: . 2009,
52. Gilman, S., Dirks, D. i Hunt, S. Measurement of head movement during auditory localization. A: *The Journal of the Acoustical Society of America*. 1979, Vol. 11, p. 37-41. DOI 10.3758/BF03205429.
53. *VRidge - Play PC VR on your Cardboard*.
54. *Use case diagram - Wikipedia*.
55. Visual, P. *What is Use Case Specification?* 2019. 2019.
56. Sheehy, L. et al. Home-based virtual reality training after discharge from hospital-based stroke rehabilitation: A parallel randomized feasibility trial. A: *Trials*. *Trials*, 2019, Vol. 20, núm. 1, p. 1-9. ISSN 17456215. DOI 10.1186/s13063-019-3438-9.
57. Bai, J. i Song, A. Development of a Novel Home Based Multi-Scene Upper Limb Rehabilitation Training and Evaluation System for Post-Stroke Patients. A: *IEEE Access*. *IEEE*, 2019, Vol. 7, p. 9667-9677. ISSN 21693536. DOI 10.1109/ACCESS.2019.2891606.
58. Gaidajis, G., Angelakoglou, K. i Aktsoğlu, D. E-waste: Environmental Problems and Current Management Engineering Science and Technology Review. A: *Journal of Engineering Science and Technology Review* [en línia]. 2010, Vol. 3, núm. 1, p. 193-199. Disponible a: www.jestr.org.
59. Barrena Medina, A. Directiva 2012/19/UE del Parlamento Europeo y del Consejo de 4 de julio de 2012 sobre residuos de aparatos eléctricos y electrónicos (RAEE). Refundición. (DOUE L 197/38, de 24 de Julio de 2012). A: *Actualidad Jurídica Ambiental*. 2012, núm. 16, p. 19-20. ISSN 1989-5666.
60. Parliament, T.H.E.E. et al. L 174/88. A: . 2011, p. 88-110.

Annex

A1. Animation Rigging Package

The *Animation Rigging Package* contains different components that have been used in this project. More in-depth explanation will follow in this section.

A1.1. Two Bone IK Constraint

This component controls and manipulates two forward kinematic bones into an inverse kinematic expression, by using a Target and a Hint, which work as if they were the hand and elbow, respectively.

The properties found in these components are (Figure A1.1):

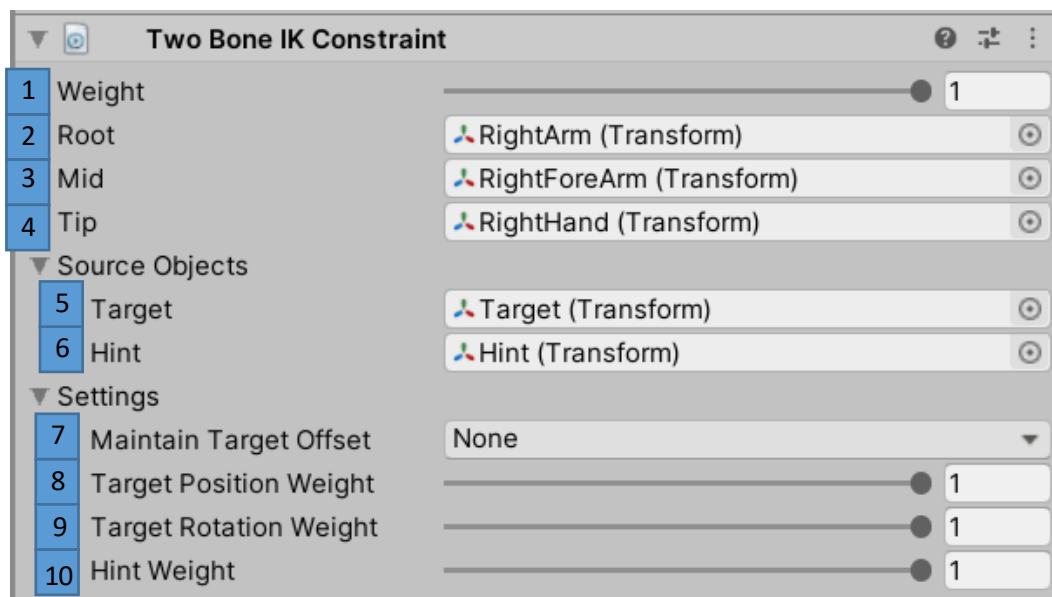


Figure A1.1. Two Bone IK Constraint Interface

- 1) **Weight:** It sets the influence on the constrained GameObjects. Weight is set within a value from 0 to 1.
- 2) **Root:** The root of constrained GameObjects, being the last bone of the two available to move.
- 3) **Mid:** The middle of the constrained GameObjects.
- 4) **Tip:** The last constrained object in the hierarchy.
- 5) **Target:** The GameObject created to act as an origin of IK chain movement.
- 6) **Hint:** (Optional) GameObject set to control the middle node.

- 7) **Maintain Target Offset:** Allows to maintain position, rotation or both offsets between the target and the hint.
- 8) **Target Position Weight:** The amount of influence the target's position has on the tip GameObject, being 1 fully and 0 none.
- 9) **Target Rotation Weight:** The amount of influence the target's rotation has on the tip GameObject, being 1 fully and 0 none.
- 10) **Hint Weight:** The influence of the hint on the constrained GameObjects hierarchy, being 1 fully influences and 0 none influenced.

In order to setup Two Bone IK, first it is necessary to identify which part corresponds to each property within the components, in Figure A1.1 right hand has been already setup. The reason behind this selection comes from understanding that the movement caption is originated at the hand (Right Hand) and related to the avatar's hand via the Target. After the hand gets another position, the immediate part of the arm gets its movement transmission being it the Right ForeArm and then the elbow - which is the Hint in this case -, followed by the upper limb (Right Arm). With all properties set, they need to be placed in the right position over the Scene. To do so the Target is placed at the hand and the Hint at the elbow, even if there's no hierarchy within the avatar, the point of connection between the forearm and the arm is got (Figure A1.2).

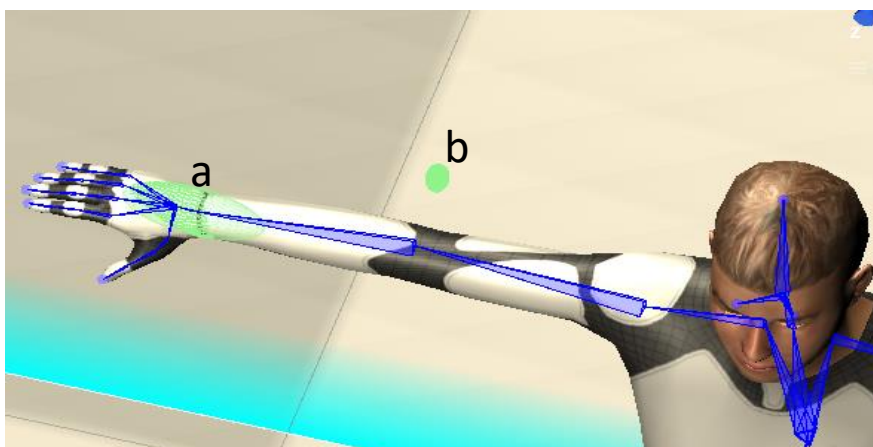


Figure A1.2. a) Target. b) Hint.

In Figure A1.2 Hint is positioned behind the elbow; this is made so moves are more realistic.

A1.2. Multi-Parent Constraint

It moves and rotates a GameObject as if it was the child of another one, by adding the benefit of choosing which of 3DoF can influence as a parent and also not getting affected by scale.

Properties found in the component are the following ones from Figure A1.3:

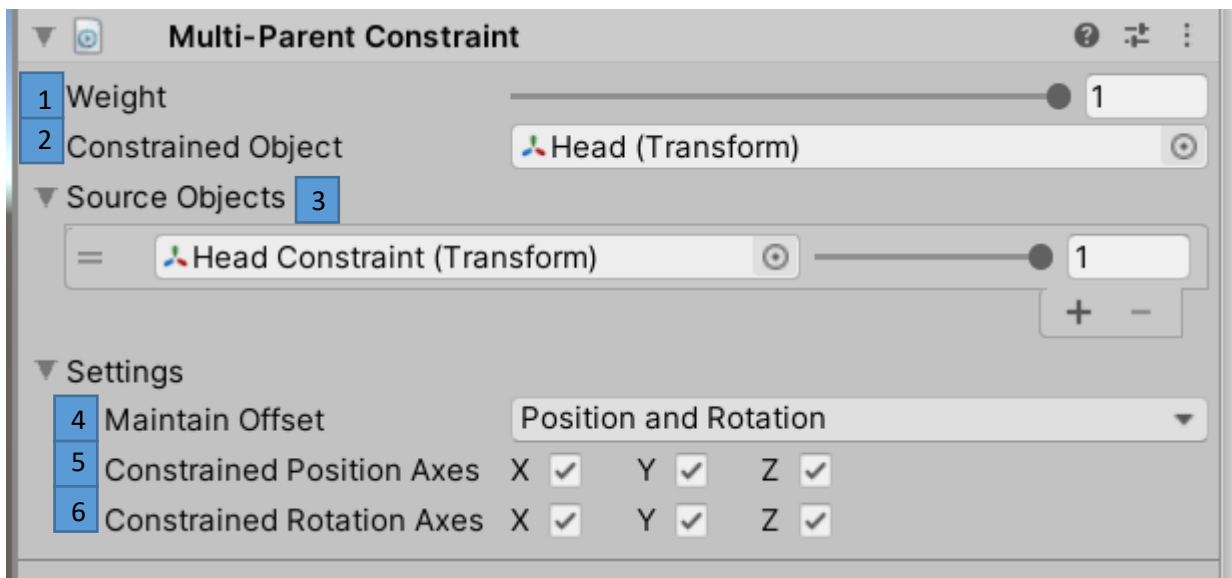


Figure A1.3. Multi-Parent Constraint Interface

- 1) **Weight:** It sets the influence on the constrained GameObjects. Weight is set within a value from 0 to 1.
- 2) **Constrained Object:** GameObject affected by the Source Object.
- 3) **Source Object:** The GameObject or list of GameObjects that will influence the position and orientation of the Constrained GameObject. In case a list is used, the order of the list will affect the influence, also by adjusting the weight of each Source Object the influence can be setup.
- 4) **Maintain Offset:** Allows to maintain position, rotation or both offsets from the constrained Object to the Source one.
- 5) **Constrained Position Axes:** Check if X, Y or Z axes are allowed to control the corresponding position of the Constrained Object.
- 6) **Constrained Rotation Axes:** Check if X, Y or Z axes are allowed to control the corresponding rotation of the Constrained Object.

To get Multi-parent constraint component work in this project, as it is shown in Figure A1.3, the main GameObject that will work as a reference in both position and rotation is *Head Constraint*, thus *Head* of the Avatar will be linked to it as the Constrained Object.

A1.3. Rig

The Rig Component works as a parent of all rig constraints, like Two Bone IK or Multi-Parent Constraints. These ones are like its children in a hierarchy (Figure A1.4). Its main use is to compile all the Rig Components defined within its local hierarchy (found in the Hierarchy Window), and consequently generate the Animation.

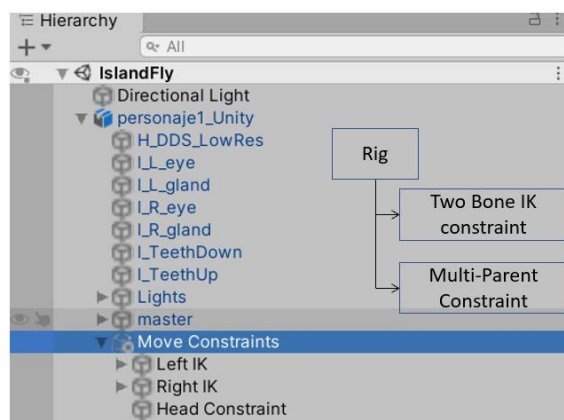


Figure A1.4. Rig Hierarchy Scheme

Therefore, a Rig Constraint can be selected in the Rig Builder as a Rig Layer (Figure A1.5). In this project just one layer has been used to set the arms movement rigging and the head movement of the avatar. More than one could be used in case different Animation behaviours are wanted to be available. To make everything work, a Rig Component is required to be under the GameObject holding the animator.

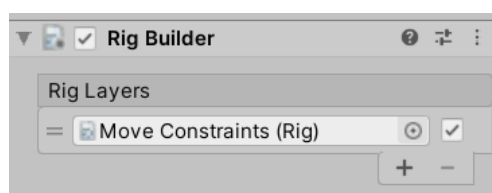


Figure A1.5. Rig Layer Interface within Rig Builder

The Rig Component has also a Weight property that allows to enable or disable its use as shown in the figure below.

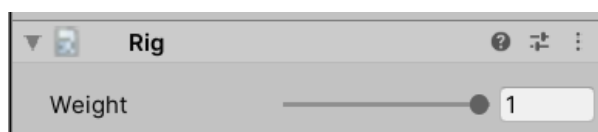


Figure A1.6. Rig Component Interface

A1.4. Rig Builder

The Rig Builder Component has the same relevance as an Animator component. In fact, they both work alongside, and this component needs an Animator Component in the GameObject to work with.

To make this Component work usefully, Rigs need to be created and then added to the Rig Builder, as Rig Layers. By doing so, different Rigs can be applied to a GameObject as shown in Figure A1.5.

The Rig Builder has been used to attach the Rig Component, in which all the inverse kinematics are applied.

A1.5. Bone Renderer

The Bone Renderer Component is a dev tool used to easily identify all the bones that form the GameObject attached to it by making them visible and selectable (Figure A1.7), but not visible while running the application.

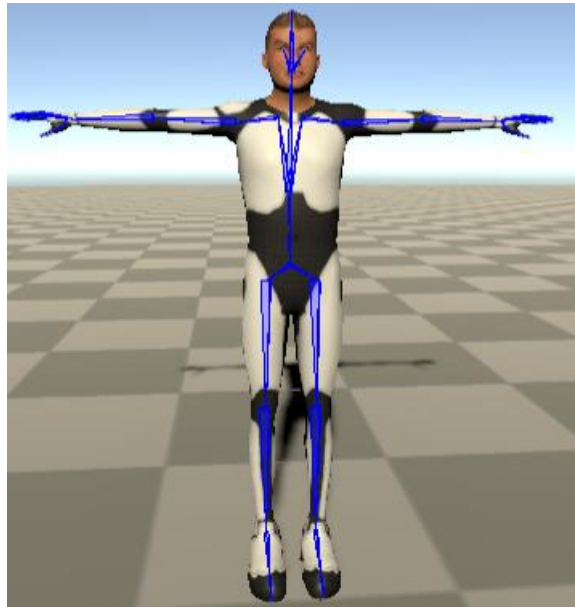


Figure A1.7. Skeleton of the Avatar by Bone Renderer Component

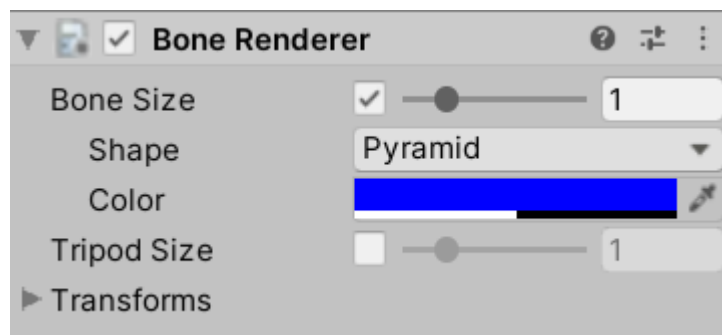


Figure A1.8. Bone Renderer Component Interface

Bone Renderer properties are:

- Bone Size: This allows to make the visible bones bigger or smaller to ease their identification.
- Shape: Line, Pyramid or Box shapes can be applied to bones in order to see them differently.
- Tripod Size: Allows to see all the axes of each component.
- Transforms: All GameObjects desired to form the skeleton are drag into this property.

This component has been used to associate Targets and Hints in their proper position with accuracy. It also allows to drag bone components into the other Rig Constraints.

A2. Angle Calculations

The following method was followed per each side to do angle calculations, thus testing the viability of getting the movement according to angle restrictions.

First, an empty object was created at 0° angle from the shoulder, so that a reference could be made by taking the X position of this Object. It matches the radius of the inner circle that the arm can do, then with the radius and the desired angles, by applying simple trigonometry and with the Equations 1 and 2, the X and Y positions got are shown in Table A2.1.

$$y = r * \sin \alpha \quad \text{Eq 1.}$$

$$x = r * \cos \alpha \quad \text{Eq 2.}$$

α	0°	15°	30°	45°	60°	75°	90°
x	0,49844	0,4814	0,4316	0,3524	0,2492	0,1290	0
y	0	0,1290	0,2492	0,3524	0,4316	0,4814	0,4984

Table A2.1. X, Y Position Results from Angle Definition

Once all the positions were calculated, different GameObjects were created for each angle, on both sides, and also their negative counterpart (-15°, -30°, etc). As a result, the Scene looked as Figure A2.1.

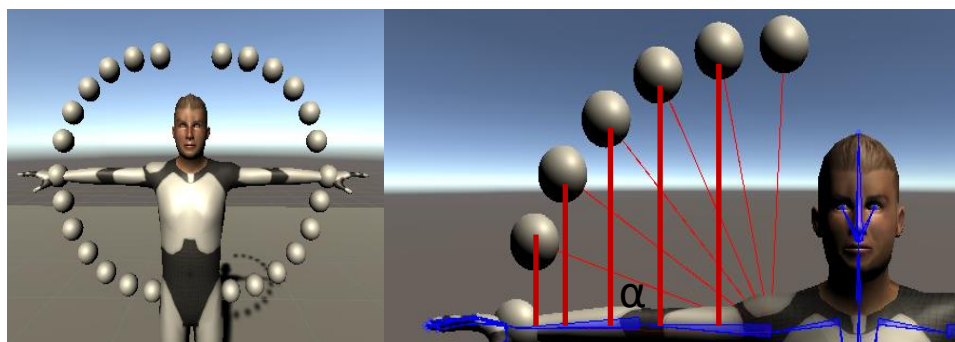


Figure A2.9. Angles Display

After all the angles were set, a script was developed to display the angle that the hand was detecting. Also, a red ray was added to draw a line from the hand's Target to the shoulder. Thus, it was visually

ensured that the angle of the Target approximately matched each of the GameObject spheres set up with each angle from Table A2.1.

The script used is the following one:

```
public class angle: MonoBehaviour
{
    public float Angle;

    public GameObject target;

    [SerializeField]

    private Transform shoulder;

    void Start()
    {
    }

    void Update()
    {
        Vector3 direction = shoulder.position - transform.position;

        Debug.DrawRay(transform.position, direction, Color.red);

        Vector3 TransformPosition = transform.localPosition;

        Vector3 ShoulderPosition = shoulder.localPosition;

        Angle = Vector3.Angle(TransformPosition, ShoulderPosition) ;
    }
}
```

Once the script and the angles were created, the hand and the Target got moved during the gameplay to test the angles displayed. Then, the angles were calculated by Unity and the results were related to the angles expected.

A3. Alternative to HTC Vive Controllers

As explained in [Section 8.6.2](#), an alternative to the HTC Vive controllers was found to recreate the movement of the user's arms. The strategy followed was to simulate the two-dimensional movement of the arms using a Gamepad implemented in the Unity Input Manager (Figure A3.1). This was done thanks to the mapping of the different Gamepads used (Figures 8.20 and 8.21).

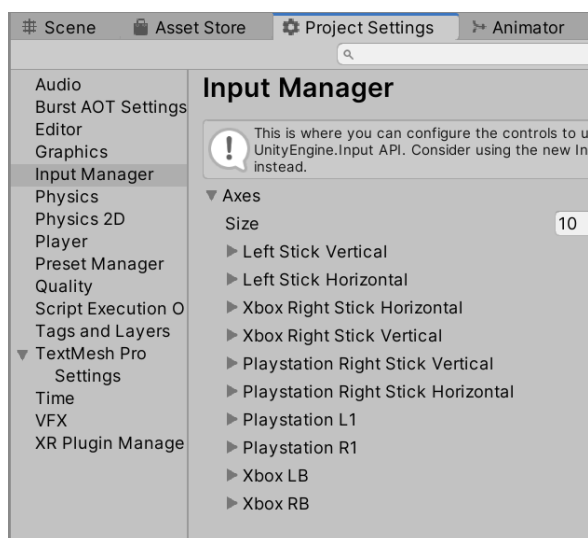


Figure A3.10. Unity's Input Manager

* To access the Input Manager, go to Edit -> Project Setting -> Input Manager

For each of the necessary axes or buttons of the Gamepads, there is a defined element in the Input Manager, indicating its properties. In the case of the left joystick, the axes are only defined once. This fact is because the axes of the left joystick are the same for the two Gamepads implemented. The other elements are named after the Gamepad they belong to, and then the button or axis they represent.

The axes of both joysticks on each Gamepad were defined, as well as two buttons on each Gamepad: RB and LB for the Xbox control and L1 and R1 for the PlayStation 4 control. These buttons' combinations were necessary to differentiate the purpose of the joysticks' movement: the left joystick is used for the avatar's rotation until R1 and L1, or RB and LB are pressed, which causes the joysticks to control the movement of the avatar's arms (mentioned in [Section 9.1](#)).

Three examples of axis and button implementation in the Input Manager are shown in the following Figures A3.2, A3.3 and A3.4.

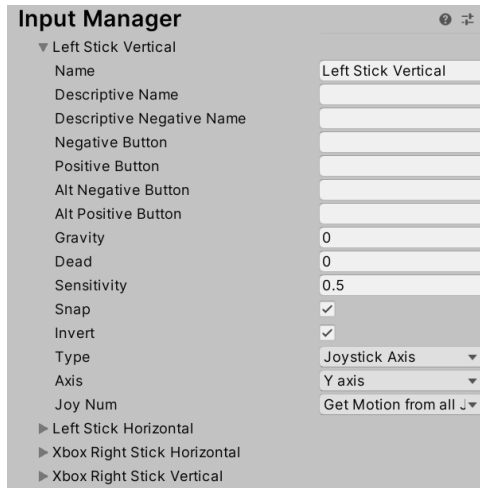


Figure A3.11. Left Joystick Vertical Axis Implementation in Unity's Input Manager

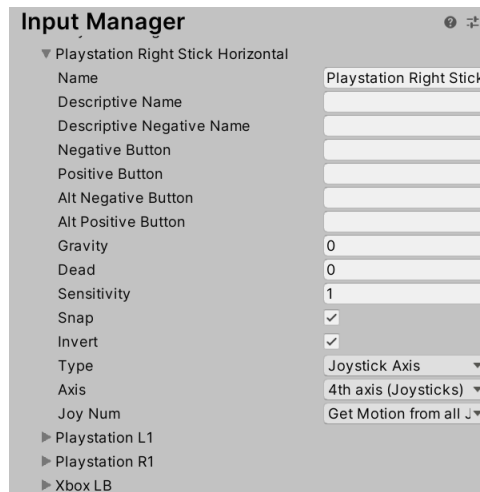


Figure A3.12. PlayStation 4 Right Joystick Horizontal Axis Implementation in Unity's Input Manager

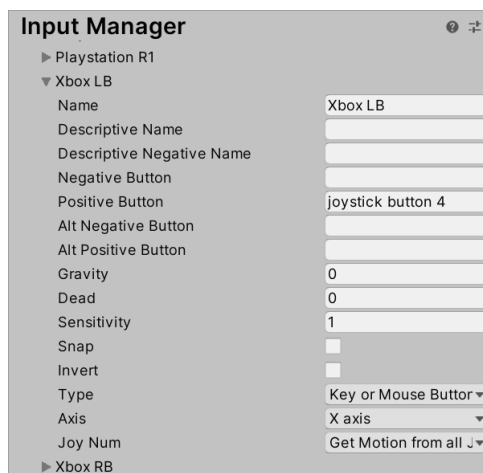


Figure A3.13. Xbox LB Button Implementation in Unity's Input Manager

A4. Immersion

A4.1. Character Conditions

In order to get the user immersed in the environment, a key factor is to sense the environment as real, or as real as possible. To get this condition achieved, a *Mesh Collider* has been created with the character design that allows the application to detect its place and avoid trespassing the terrain, and any other object in the Scene that has another *Collider* component. By accomplishing this, when the character collides with a tree, a mountain or another object, it will slow down the speed and hit this object, unless it is avoided.

A4.2. Audio Files

Once the movement is started, the flying sensation needs to be addressed not only as a motion sense but also with an auditory stimulus. Two different sounds have been added to the character. These sounds are:

- Flapping sound of a butterfly
- Wind sound

The first sound has been modified and intensified, but it is subtle enough not to overlap the rest of the sounds in the environment. These sounds are played only when the flapping movement is done.

The second sound is the sound of wind. It is played when the character reaches a minimum speed of 10 u/s. When this speed is achieved, the user has to sense that it has increased his velocity, not only with the special references but also with the sound of the wind.

A5. Scripts

A5.1. Pose

In this script named *Pose*, movement detection is done by *Triggers*. For each *Trigger's* tag, there is a Boolean value that returns *true* or *false*, depending on where the hands are. In other words, this script checks whether each movement is being done or not.

This script has been used twice, one for each hand.

The following variables have been related to each movement:

- UpperOK and LowerOK are crucial for flapping.
- Glide is a unique variable set for gliding.
- GlideLeft and GlideRight are set to detect in what direction the user is going to glide
- Brake and AboutToBrake are variables set in case another action is desired to include in the application.
- GlideUpper and GlideLower are spare variables, set in case another action should be included in the application.

The *Pose* script is called in *FlyMovement* script to check flap and glide actions.

```
public class Pose: MonoBehaviour
{
    public bool UpperOK;
    public bool LowerOK;
    public bool GlideLeft;
    public bool GlideRight;
    public bool GlideUpper;
    public bool GlideLower;
    public bool AboutToBrake;
    public bool Brake;
    public bool Glide;
    void OnTriggerEnter(Collider obj)
```

```
{
  if (obj.tag == "RightUpperLimit" || obj.tag == "LeftUpperLimit")
  {
    UpperOK = true;
  }
  if (obj.tag == "RightLowerLimit" || obj.tag == "LeftLowerLimit")
  {
    LowerOK = true;
  }
}

void OnTriggerStay(Collider obj2)
{
  if (obj2.tag == "RightUpperLimit" || obj2.tag == "LeftLowerLimit")
  {
    GlideLeft = true;
  }
  if (obj2.tag == "RightLowerLimit" || obj2.tag == "LeftUpperLimit")
  {
    GlideRight = true;
  }
  if (obj2.tag == "RightUpperLimit" || obj2.tag == "LeftUpperLimit")
  {
    GlideUpper = true;
  }
  if (obj2.tag == "RightLowerLimit" || obj2.tag == "LeftLowerLimit")
  {
```



```
    GlideLower = true;
}
if (obj2.tag == "RightMidLimit" || obj2.tag == "LeftMidLimit")
{
    Glide = true;
}
if (obj2.tag == "RightHand" || obj2.tag == "LeftHand")
{
    Brake = true;
}
}
void OnTriggerExit(Collider obj3)
{
    if (obj3.tag == "RightUpperLimit" || obj3.tag == "LeftUpperLimit")
    {
        UpperOK = false;
    }
    if (obj3.tag == "RightLowerLimit" || obj3.tag == "LeftLowerLimit")
    {
        LowerOK = false;
    }
    if (obj3.tag == "RightUpperLimit" || obj3.tag == "LeftLowerLimit")
    {
        GlideLeft = false;
    }
    if (obj3.tag == "RightLowerLimit" || obj3.tag == "LeftUpperLimit")
```

```
{
  GlideRight = false;
}
if (obj3.tag == "RightUpperLimit" || obj3.tag == "LeftUpperLimit")
{
  GlideUpper = false;
}
if (obj3.tag == "RightLowerLimit" || obj3.tag == "LeftLowerLimit")
{
  GlideLower = false;
}
if (obj3.tag == "RightMidLimit" || obj3.tag == "LeftMidLimit")
{
  Glide = false;
}
if (obj3.tag == "RightHand" || obj3.tag == "LeftHand")
{
  Brake = false;
}
}
}
```

A5.2. FlyMovement

The backbone script of this project is called *FlyMovement*. This script is used to move the avatar around the Scene and allows the avatar to:

- Accelerate
- Decelerate

- Increase Altitude/Height
- Decrease Altitude/Height
- Activate Animations
- Activate Sounds

Firstly, variables are stated. Initially, variables regarding acceleration and deceleration are created. Secondly, movement Booleans are generated to check each movement by calling out the script *Pose* of each hand's Target. Finally, other variables are created to avoid bugs and errors in flap movement, followed by speed variables and audio clips.

After stating all variables, function *Start* gets the Controller Component – which allows the player to move -, the Animation Component and initializes some other variables that will be used later.

Once *void Start* is executed, the function *void Update* is executed for each frame. The first method executed inside this function is *Hands*; this method checks the *Pose* script to detect hands position, thus movement. After having movement detected, if the flap is detected, it gets a Boolean value of *false*, to avoid errors, and another variable called *IsFlapping* gets a Boolean value of *true*. The reason why this is done is that once the action of flapping is done, it is not done anymore until a few seconds later. However, the state of the avatar flapping lasts over time.

Then a brief counter to check the action of flapping starts, and so if the flap movement is done at least three times, the Fly Animation is set to *true* in order to start the Flying movement. The rotation of the headset also sets the movement direction.

After performing the flapping movement more than three times, a variable called *IsFlying* gets a *true* value and starts the function *Accel*. His function allows the patient to accelerate. In case the flap has not been detected for more than three times and, otherwise, glide takes place as the central movement, the function *MaintainVelocity* is called in order to keep the avatar's speed on the same value. If *MaintainVelocity* is active for more than 15 seconds, the avatar starts to decelerate.

Lastly but not least, gravity is constantly applied to the player.

FlyMovement script is defined as follows:

```
public class FlyMovement : MonoBehaviour
{
    // Movement variables

    public float maxSpeed; // The maximum speed
```

```
private float MaxSpeed; // Maximum speed to return

public float timeZeroToMax = 10f;

public float timeMaxToOne = 6f;

float accelRate;

float decelRate;

public float forwardVelocity;

public float transpeed = 8f;

float rotation = 0;

public float rotspeed = 50f;

public float gravity = 3f;

public float MaxHeight;

Vector3 moveDir = Vector3.zero; // Vector created for altitude restrictions

CharacterController controller;

Animator anim; // Idle and Flying animations call-outs

private Controller hands;

private float xAxis;

//Movement conditions

public bool flap;

public bool IsFlying;

public bool GlideRight;

public bool GlideLeft;

public bool Glide;

private bool Upper;

private bool Lower;

//Objects to detect movement

public GameObject RightTarget;
```

```
public GameObject LeftTarget;

public GameObject Camara;

private float CamY;

private float CamZ;

Pose RHand;

Pose LHand;

private float Flaptime; //How many times flap has been done

public bool Fly; //Variable for script controller

private bool IsFlapping;

// Counters used for different purposes

private float Internalflaptime = 0f;

private bool Startcounter;

public float Counter;

public float counter;

public float timeMinVel;

//Variables to locate different speeds

float Speed100;

float Speed80;

float Speed60;

float Speed40;

float Speed20;

//Sound Effects

public AudioClip FlapSound;

public AudioClip WindSound;
```

```
void Start()
{
    controller = GetComponent<CharacterController>();
    anim = GetComponent<Animator>();
    MaxSpeed = maxSpeed;
    forwardVelocity = 0.1f * maxSpeed;
    xAxis = 0f;
    FlapTime = 0f;
    counter = 5f;
    Speed100 = maxSpeed * 2;
    Speed80 = maxSpeed * 1.8f;
    Speed60 = maxSpeed * 1.6f;
    Speed40 = maxSpeed * 1.4f;
    Speed20 = maxSpeed * 1.2f;

    CamY = Camara.transform.localPosition.y ;
    CamZ = Camara.transform.localPosition.z;

    if (counter != 0)
    {
        Counter = counter;
    }
    else
    {
        Counter = 5f;
        counter = 5f;
    }
}
```

```
}  
  
private void OnEnable()  
{  
}  
  
void Update()  
{  
    Hands(); //Set movement conditions  
  
    // Fly movement  
  
    if (Input.GetButton("Playstation R1") && Input.GetButton("Playstation L1"))  
    {  
    }  
  
    else  
    {  
        rotation = Input.GetAxis("Left Stick Horizontal") * rotspeed * Time.deltaTime;  
        transform.Rotate(0, rotation, 0);  
    }  
  
  
    if (flap == true)  
    {  
        flap = false;  
  
        Glide = false;  
  
        GlideLeft = false;  
  
        GlideRight = false;  
  
        Flaptime += 1;  
  
        IsFlapping = true;  
  
        AudioSource.PlayClipAtPoint(FlapSound, transform.position);  
    }  
}
```

```
if (Flaptime > 3)
{
    Camara.transform.localPosition = new Vector3(Camara.transform.localPosition.x, -2.2f, -
    0.6f);

    anim.SetBool("Fly", true);

    Fly = true; //Flapping Control Fly in Controller Script

    IsFlying = true;

    Startcounter = true;

    moveDir = new Vector3(0, 0, 1);

    moveDir *= forwardVelocity;

    moveDir = transform.TransformDirection(moveDir);

    transform.position = transform.position + Camara.transform.forward;
}
}

if (IsFlying == true)
{
    Accel();

    Internalflaptime += Time.deltaTime;

    if (Internalflaptime > counter + 1.5f)
    {
        IsFlying = false;

        Internalflaptime = 0f;
    }
}

else if ((Glide == true || GlideLeft == true || GlideRight == true) && flap == false)
```



```
{  
    MaintainVelocity();  
    float MaintainGlide = 0f;  
    MaintainGlide += Time.deltaTime;  
    if (MaintainGlide > 15f && IsFlying == false)  
    {  
        Decel();  
    }  
}  
else  
{  
    Decel();  
}  
moveDir.y -= gravity * Time.deltaTime; // Gravity applied constantly  
controller.Move(moveDir * Time.deltaTime);  
float height = Mathf.Clamp(transform.position.y, 5, MaxHeight);  
transform.position = new Vector3(transform.position.x, height, transform.position.z);  
}  
  
void LateUpdate()  
{  
    IsFlapping = false; // With this line, the movement of flap is always false at the end of each frame  
    // after executing all the code to avoid bugs  
}  
  
void Hands()
```

```
{  
    RHand = RightTarget.GetComponent<Pose>();  
  
    bool RightUpper = RHand.UpperOK;  
  
    bool RightLower = RHand.LowerOK;  
  
  
    LHand = LeftTarget.GetComponent<Pose>();  
  
    bool LeftUpper = LHand.UpperOK;  
  
    bool LeftLower = LHand.LowerOK;  
  
  
    //Following parameters include static positions of hands  
  
    // Parameters to glide on different directions  
  
    bool LGlideRight = LHand.GlideRight;  
  
    bool RGlideRight = RHand.GlideRight;  
  
    bool RGlideLeft = RHand.GlideLeft;  
  
    bool LGlideLeft = LHand.GlideLeft;  
  
    // Both Hands up  
  
    bool RGlideUpper = RHand.GlideUpper;  
  
    bool LGlideUpper = LHand.GlideUpper;  
  
    // Both Hands down  
  
    bool LGlideLower = LHand.GlideLower;  
  
    bool RGlideLower = RHand.GlideLower;  
  
    // Glide straight forward  
  
    bool RGlide = RHand.Glide;  
  
    bool LGlide = LHand.Glide;  
  
    // Both Hands collide;  
  
    bool Brake = LHand.Brake;
```

```
if (RightUpper == true && LeftUpper == true)
{
    Upper = true;
}

if (RightLower == true && LeftLower == true)
{
    Lower = true;
}

if (Upper == true && Lower == true)
{
    Upper = Lower = false;
    flap = true;
}

// Gliding conditions

if (RGlideRight == true && LGlideRight == true)
{
    GlideRight = true;
}

if (RGlideRight == false && LGlideRight == false)
{
    GlideRight = false;
}

if (RGlideLeft == true && LGlideLeft == true)
```

```
{
    GlideLeft = true;
}
if (RGLideLeft == false && LGLideLeft == false)
{
    GlideLeft = false;
}
if (RGLide == true && LGLide == true)
{
    Glide = true;
}
if (RGLide == false && LGLide == false)
{
    Glide = false;
}
}
void Accel()
{
    accelRate = maxSpeed / timeZeroToMax;
    forwardVelocity += accelRate * Time.deltaTime;
    forwardVelocity = Mathf.Min(forwardVelocity, maxSpeed);
    moveDir.y += transpeed * Time.deltaTime;
    if (Startcounter == true && IsFlying == true && forwardVelocity >= MaxSpeed)
    {
        AudioSource.PlayClipAtPoint(WindSound, transform.position);
        Counter += Time.deltaTime;
    }
}
```

//Depending on how much time it takes for the player to do the flapping movement, different maximum speeds is set

```
if (0 < Counter && Counter <= 1f && IsFlapping == true)
{
    Counter = 0f;
    maxSpeed = Speed100;
    IsFlapping = false;
}

if (1 < Counter && Counter <= 2f && IsFlapping == true)
{
    Counter = 0f;
    maxSpeed = Speed80;
    IsFlapping = false;
}

if (2 < Counter && Counter <= 3f && IsFlapping == true)
{
    Counter = 0f;
    maxSpeed = Speed60;
    IsFlapping = false;
}

if (3 < Counter && Counter <= 4f && IsFlapping == true)
{
    Counter = 0f;
    maxSpeed = Speed40;
    IsFlapping = false;
}
```

```
if (Counter > counter)
{
    Counter = 0f;
    maxSpeed = MaxSpeed;
    IsFlapping = false;
}
}

if (Startcounter == false)
{
    Counter = 0f;
}
}

void Decel()
{
    decelRate = -maxSpeed / timeMaxToOne;
    forwardVelocity += decelRate * Time.deltaTime;
    forwardVelocity = Mathf.Max(forwardVelocity, 0.1f * maxSpeed);
    if (forwardVelocity <= 0.1f * maxSpeed+0.1f)
    {
        timeMinVel += Time.deltaTime;
    }
    if (timeMinVel > 5f)
    {
        Startcounter = false;
        forwardVelocity = 0f;
        Fly = false; // Flapping Control in Controller script
    }
}
```

```
anim.SetBool("Fly", false); // Idle Animation

FlapTime = 0f;

timeMinVel = 0f;

    }

}

void MaintainVelocity()
{
    if (forwardVelocity > 10f)
    {
        decelRate = -maxSpeed / timeMaxToOne;
        forwardVelocity += decelRate * Time.deltaTime;
        forwardVelocity = Mathf.Max(forwardVelocity, 10);
    }

    if (forwardVelocity < 10f)
    {
        float MaintainVel = forwardVelocity;
        forwardVelocity = MaintainVel;
    }
}

void Rotation()
{
    if (GlideLeft == true)
    {
        rotation = -10f * rotspeed * Time.deltaTime;
        anim.SetBool("FlyLeft", true);
        transform.Rotate(0, rotation, 0);
    }
    else if (GlideRight == true)
    {
        rotation = 10f * rotspeed * Time.deltaTime;
```

```

        anim.SetBool("FlyRight", true);
        transform.Rotate(0, rotation, 0);
    }
    else
    {
        anim.SetBool("FlyRight", false);
        anim.SetBool("FlyLeft", false);
    }
}
}
}

```

A5.3. Controller

This script results from looking for an alternative to move the avatar's arms, due to lack of access to the HTC Vive Virtual Reality system. As explained on [Annex A3](#), the script called *Controller* makes the joysticks on the Gamepad - controlled by the user - to provoke the avatar's arms to move. This movement happens with a precondition: hold down the R1 + L1 buttons on the Gamepad.

This script has been used twice, one for each hand, locating it on each hand's Target.

In the beginning, it is necessary to understand the variables created for this script. There are public Boolean variables from which the user can specify what Gamepad he is using (Xbox or PlayStation4) and can also specify what joystick he wants to use to move the arm (target) and in what script it is located. There are also variables regarding the position of the arms and a variable that calls the *FlyMovement* script.

Once the Start method has been passed, and if the R1 and L1 buttons are pressed, the Update method checks that the avatar flies according to the *FlyMovement* script. Depending on whether the avatar is flying or not, one method or another is activated. If the avatar is not flying, the method that causes the avatar's arms to move in an upright position - the so-called *FlappingControl* method - is activated. On the contrary, if the avatar is flying, the method that causes the avatar to lay down -the so-called *FlappingControlFly* method- is activated instead.

The *Controller* script is shown here below:

```

public class Controller : MonoBehaviour
{

```




```
public float movementspeed = 1.0f;

public bool XboxController;

public bool PlayStationController;

public bool RightJoystick;

public bool LeftJoystick;

private Transform thisTransform;

Vector3 startPos;

public GameObject hand;

private Vector3 Pos;

private float X;

private float Y;

private float Z;

public GameObject avatar;

FlyMovement Fly;

public bool IsFlying;

// Start is called before the first frame update

private void Start()

{

    thisTransform = transform;

    startPos = thisTransform.localPosition;

    X = startPos.x;

    Z = startPos.z;
```

```
Y = startPos.y;
}

void Update()
{
    Fly = avatar.GetComponent<FlyMovement>();

    IsFlying = Fly.Fly;

    if (Input.GetButton ("Playstation R1") && Input.GetButton ("Playstation L1") ||
        Input.GetButton ("Xbox RB") && Input.GetButton ("Xbox LB"))
    {
        if (IsFlying == false)
        {
            FlappingControl();
        }

        if (IsFlying == true)
        {
            FlappingControlFly();
        }
    }
}

void FlappingControl()
{
    if (PlayStationController == true)
    {
```

```
    if (RightJoystick)
    {
        Vector3 moveRightStick = Vector3.zero;

        moveRightStick.y = Input.GetAxis("Playstation Right Stick Vertical");

        this.transform.localPosition = new Vector3(hand.transform.position.x + moveRightStick.x
        - (0.8f), (hand.transform.position.y + moveRightStick.y - (1.8f)), Z);
    }

    else if (LeftJoystick)
    {
        Vector3 moveLeftStick = Vector3.zero;

        moveLeftStick.y = Input.GetAxis("Left Stick Vertical");

        this.transform.localPosition = new Vector3(hand.transform.position.x + moveLeftStick.x,
        hand.transform.position.y + moveLeftStick.y, Z);
    }
}

if (XboxController == true)
{
    if (RightJoystick)
    {
        Vector3 moveRightStick = Vector3.zero;

        moveRightStick.y = -Input.GetAxis("Xbox Right Stick Vertical");

        this.transform.localPosition = new Vector3(X, (hand.transform.localPosition.y +
        moveRightStick.y), Z);
    }
}
```

```
else if (LeftJoystick)

{

    Vector3 moveLeftStick = Vector3.zero;

    moveLeftStick.y = Input.GetAxis("Left Stick Vertical") * 0.75f;

    this.transform.localPosition = new Vector3(X, (hand.transform.localPosition.y +
moveLeftStick.y +1.4f ), Z);

}

}

}

void FlappingControlFly()

{

    if (PlayStationController == true)

    {

        if (RightJoystick)

        {

            Vector3 moveRightStick = Vector3.zero;

            moveRightStick.y = Input.GetAxis("Playstation Right Stick Vertical");

            this.transform.position = new Vector3(hand.transform.position.x + 1f + moveRightStick.x
+ 0.8f, (hand.transform.position.y + moveRightStick.y), Z);

        }

        else if (LeftJoystick)

        {

            Vector3 moveLeftStick = Vector3.zero;
```

```
        moveLeftStick.y = Input.GetAxis("Left Stick Vertical");

        this.transform.localPosition = new Vector3(hand.transform.position.x + moveLeftStick.x -
0.8f, (hand.transform.position.y + moveLeftStick.y), Z);

    }

}

if (XboxController == true)

{

    if (RightJoystick)

    {

        Vector3 moveRightStick = Vector3.zero;

        moveRightStick.y = -Input.GetAxis("Xbox Right Stick Vertical");

        Pos = new Vector3(0f, Y + (hand.transform.localPosition.y + moveRightStick.y), 0f);

        this.transform.localPosition = Pos;

    }

    else if (LeftJoystick)

    {

        Vector3 moveLeftStick = Vector3.zero;

        moveLeftStick.y = Input.GetAxis("Left Stick Vertical");

        Pos = new Vector3(-0.6869611f, Y + (hand.transform.localPosition.y + moveLeftStick.y), 0f);

        this.transform.localPosition = Pos;

    }

}

}
```

}



