Virtual Reality Multitracking Interaction with 3D Characters for Immersive Training Applications

Salva Kirakosian

A thesis presented for the degree of
Diploma of Science in Electrical and Computer Engineering

Πολυτεχνείο Κρήτης

School of E.C.E.
Technical University of Crete
Greece

29.09.2017

Virtual Reality Multitracking Interaction with 3D Characters for Immersive Training Applications

Salva Kirakosian

Abstract

This thesis implements a Virtual Reality (VR) Multitracking interaction with 3D Characters resulting in an Immersive dance training application using the Occulus Rift HMD system and the Leap Motion hand tracking Controller. The player, who is placed in the main scene, is trained by the 3D white light scanned human character in a virtual environment which is designed and developed in Unity3D. Realistic animations were applied to the character using a complex Inverse Kinematics (IK) system applied to the rigged 3D Human Model. Characters are rigged with custom skeletons using 3D geometrical bones and IK solvers. Motion Capture data were retargetted so that animations of the 3D character are produced. The Motion Capture data, derived from the tracking device Kinect v2, were applied to the rigged 3D model to create realistic dance animations. Rendering optimization techniques such as Baked GI, were applied in the scene to reduce the GPU rendering load and to increase the frame rate per second (FPS) of the VR Application.

The project features a tutorial guide process, so that the users familiarize themselves with the user interface (UI) being prepared for their VR dance training. During this process, the users are allowed to modify the sound volume and restart the training through the Settings UI panel. The 3D character is standing just below the UI tutorial guide in front of the users guiding them in relation to the training functionality. This procedure allows the users to practice their actions by interacting with UI panels using gestures such as thumb and waving hands, combining touches and pinch actions. Users with VR experience can skip this pre-training procedure and continue with the main Dance Tutorial, where they can select from the main UI tutorial panel, one out of three basic options. The training process starts with the first option named "Basic Steps" of the Salsa Latin dance which includes the "Basic A", the "Side Steps" and the "Right Turn" dance routine. By selecting the "Interaction" option, the users can practice dance by leading the 3D model to execute the "Side Steps" and the "Turn" animation. Finally, the users can select the last "Dance" option, where they can improvise and dance, leading the 3D character to basic "Side Steps" and "Turn" while they execute the basic steps they have learned, during the first option.

Acknowledgements

It is never late to make changes in our life, changes that make us realize the pleasure of living. We have to believe in ourselves that we can make from nothing everything. Give love to them that surround us, be honest, do not create fear in our mind for every backward step we make, move on when we are ready. Thanks to all friends that was next to me during my stay in Chania. For all the people that made my life in this city be like a holiday. Especially to Christos Birakis and Katerina Mania, Emmanuel Maravelakis and his Lab for the White Light scanning procedure and Jose Alberto Rodriguez a famous Latin dancer and instructor for being a 3D model in our project.

Thanks to my family that believe in me and never loose their faith.

I love you mama

Contents

Chapter 1

Introduction

1.1 Scope

The world is acknowledging and accepting the concept of Virtual Reality for fun but also for training. Imagine attending lectures in a amphitheater at the Technical University of Crete sitting in your room, or watching the game of your favorite sport from the VIP section from your home. All this is now possible and achievable with Virtual Reality (VR). For those who thought VR was only confined to gaming, it is already being applied to innovative and brilliant training applications such as in Medical Science, Science and Nuclear Reality, Military, Entertainment, Fashion, Business and Marketing, Journalism, Real Estate and Architecture, Sports, Public Speaking and Investigating Crime Scenes, to name a few.

Nowadays, the proliferation of social media and the frequent use of mobile devices have affected the socialization of people. People can not be easily in contact with other people that they don't know or even dance with them. For this reason, the presented VR application offers a chance to the user to learn dancing routines in a simulated world, and subsequently be able to make a first step towards socializing in a real world dance community. The goal of this project was to create an immersive interactive training application offering dance tutorials so that users learn basic steps of the Salsa Latin dance. All of the above will be explained in greater detail in the following chapters.

This thesis implements a Virtual Reality (VR) Multitracking interaction with 3D Characters resulting in an Immersive dance training application using the Occulus Rift HMD system and the Leap Motion hand tracking Controller. The player, who is placed in the main scene, is trained by the 3D white light scanned human character in a virtual environment which is designed and developed in Unity3D. Realistic animations were applied to the character using a complex Inverse Kinematics (IK) system applied to the rigged 3D Human Model. Characters are rigged with custom skeletons using 3D geometrical bones and IK solvers. Motion Capture data were retargetted so that animations of the 3D character are produced. The Motion Capture data, derived from the tracking device Kinect v2, were applied to the rigged 3D model to create realistic dance animations. Rendering optimization techniques such as Baked GI, were applied in the scene to reduce the GPU rendering load and to increase the FSP of the VR Application.

The project features a tutorial guide process, so that the users familiarize themselves with the user interface (UI) being prepared for their VR dance training.

During this process, the users are allowed to modify the sound volume and restart the training through the Settings UI panel. The 3D character is standing just below the UI tutorial guide in front of the users guiding them in relation to the training functionality. This procedure allows the users to practice their actions by interacting with UI panels using gestures such as thumb and waving hands, combining touches and pinch actions. Users with VR experience can skip this pre-training procedure and continue with the main Dance Tutorial, where they can select from the main UI tutorial panel, one out of three basic options. The training process starts with the first option named "Basic Steps" of the Salsa Latin dance which includes the "Basic A", the "Side Steps" and the "Right Turn" dance routine. By selecting the "Interaction" option, the users can practice dance by leading the 3D model to execute the "Side Steps" and the "Turn" animation. Finally, the users can select the last "Dance" option, where they can improvise and dance, leading the 3D character to basic "Side Steps" and "Turn" while they execute the basic steps they have learned, during the first option.

1.2 Thesis Outline

This thesis is divided into seven main chapters based on their content. The first chapter is an introduction to the thesis. The following chapter contains general background information and knowledge, to help the reader understand the subject of this thesis. A detailed description of the development of the project follows, along with a description on how the different parts of the application, were put together. The final chapter explains how the system was evaluated based on users' feedback. Our conclusions and suggested future work are included in the end of the last chapter. The structure of the thesis, is explained in greater detail, in the following paragraphs.

Chapter 2 – Background: This chapter acts as a prologue, to explain the fundamental principles of Virtual Reality (VR). It defines the term and the background of Virtual Reality, i.e. what exactly VR is and how people experience it. Afterwards, a background analysis of Head Mounted Display (HMD) technology is presented, followed by a description of existing HMDs in 2017 and their most important features are explained. We, then, present the most common Motion Capture techniques and Game Engines as well as 3D modeling and animation methodologies. At the end, we introduce general info about the Salsa music and dance.

Chapter 3 – Cross platform Game Engine Software: The third chapter presents the software platforms employed for the development of the dance training application. A detailed description of the utilized platform Unity 3D is provided including its most important components, focusing on programming geometry behaviors, UIs and integration of multiple hardware components. A brief description of two industry-standard 3D modeling software, e.g., 3Ds Max and Motion builder, is offered used for editing and creating the 3D models required for our project.

Chapter 4 – User View: In this section, a detailed description of the dance training is offered. Initially, a description of the pre-tutorial process is analyzed, where users adapt to the VR environment and the interaction with the UI panels. The main tutorial comes next where users can select from the Main UI tutorial panel one out of three basic options, starting with the first option "Basic Steps" of the Salsa Latin dance which includes the "Basic A", the "Side Steps" and the "Right

Turn". By selecting the next "Interaction" option, the users can practice dance by leading the 3D model to execute the "Side Steps" and the "Turn" animation. Finally, the last "Dance" option is described, where they can improvise and dance, leading the 3D character to basic "Side Steps" and "Turn" while they execute the basic steps, during the first option. At the end of this section, we describe the design and implementation of the 3D Virtual Scene and the minimum PC requirements needed for the best VR experience.

Chapter 5 – Implementation: In this section, a detailed description of the UI implementation and environmental modeling is put forward. In this section, the implementation of the main 3D Scene and tutorials are explained. Next, the development of the animations is analyzed as well as how the interaction with the UI is handled. A detailed description follows related to the gestures that are used in this project for intuitive interaction, which are the "Thumb Up" and "Waving hand" gestures. Next, it is shown how the interaction between the user and the training application itself, is achieved. The realistic dance animations of the 3D humanoid character are also explained, which are implemented based on motion capture applying In-versed Kinematics to the rigged 3D model. Finally, a description of the Artificial Intelligence (AI) navigation offered in order for the 3D Character to follow the user, is included.

Chapter 6 - Evaluation: In this chapter, a description of the procedure that is used to explore the users' level of immersion or presence as well as potential motion sickness symptoms, is described. Users complete standard questionnaires after the VR training experience has occurred. Users also offer feedback in relation to their overall impression after using the dance training application signifying perceptual as well as usability issues. Finally, the results of the evaluation process are presented at the end of this section.

Chapter 7 – Conclusions & Future Work: This chapter contains the main conclusions, as well as hints about potential future work that could possibly extend this thesis. Finally, certain technological suggestions and feedback of best practice are communicated to future VR developers.

Chapter 2

Background

This chapter aims to offer to the reader background knowledge relevant to this thesis. A definition of Virtual Reality (VR) is provided. A brief history of Head Mounted Displays (HMD) follows from 1930 till today. HMDs at the present time are described in order to show how much they evolved since the previously available devices a few years ago. Game Engines which provide the software framework for the creation and development of 3D interactive applications, are also discussed in this chapter. Furthermore, the reason why the modeling and animation are such important elements of Virtual Reality experiences, is described in detail.

2.1 The Reality-Virtuality Continuum

There is a fair amount of confusion about the differences between Augmented Reality (AR) and Virtual Reality (VR). In news reports and in the blogosphere, these terms are not always used distinctly. A conceptual framework established more than 35 years ago may be useful to establish clarity [1],[2].

In the early 1990s, researchers Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino introduced a concept called the Reality-Virtuality (RV) continuum (Milgram, 1994). While the researchers originally designed the Reality-Virtuality continuum to address mixed reality and the display technologies of the era, the original framework is still quite useful. They defined mixed reality environments as those in which "real world and virtual world objects are presented together...". Their definition of mixed reality served as an umbrella term that encompassed both Virtual and Augmented Reality technologies.

The term 'mixed reality' has since largely fallen out of use. Since the companies in this space have staked their branding and functional identities in one medium or the other, there has been little reason to use the term.

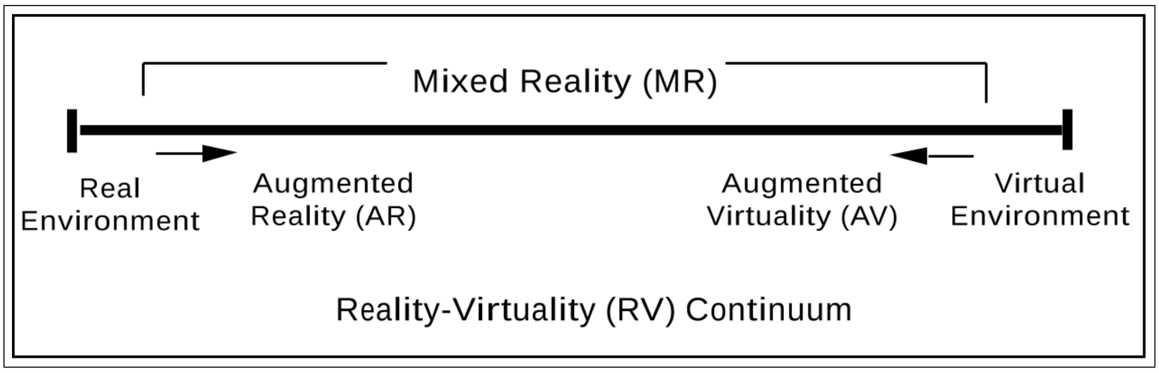


Figure 2.1: The Reality-Virtuality Continuum

According to Milgram et al., there are many variations of technology-altered forms of reality beyond the four listed in Figure (2.1).

They chose, however, to highlight four prominent versions in what they called their "simplified representation of a[n] RV Continuum".

Differences Between Augmented Reality and Virtual Reality

- First, there is unadorned, unaltered reality with a capital R. A real environment is "any environment consisting solely of real objects, and includes whatever might be observed when viewing a real-world scene either directly in person, or through…some sort of video display." Note that simply viewing a real environment through digital means, such as through a phone, tablet or computer does not make it virtual.

- Second, there is Augmented Reality which consists of a primarily real environment with digital and virtual data, images and objects superimposed or layered upon the real world. An example of Augmented Reality companies would be Magic Leap and Blippar.

- Third, there is also Augmented Virtuality (a term not currently in use but descriptive of the technologies on the horizon), which consists of spaces that are primarily virtual with some real objects, images, and data introduced into the virtual world. As an aside, if virtual objects are superimposed or layered upon real environments as in the case of augmented reality, how should we describe real objects introduced into primarily virtual environments as in the case of Augmented Virtuality?

- Fourth, the Milgram definition of virtual environments/Virtual Reality includes both immersive virtual worlds as well as those that are only monitor-based so long as the simulations consist "solely of virtual objects." This definition is cumbersome and outdated because Virtual Reality environments can now be launched through a variety of hardware technologies.

For the purpose of the current thesis, a Virtual Reality immersive application was implemented, displayed on a modern HMD (the Oculus Rift) to simulate a 3D dance training environment.

2.2 Virtual Reality

Virtual Reality (VR) has been notoriously difficult to define over the years. Many people take "virtual" to mean fake or unreal, and "reality" to refer to the real world. This results in an oxymoron. The actual definition of virtual, however, is "to have the effect of being such without actually being such". The definition of "reality" is "the property of being real", and one of the definitions of "real" is "to have concrete existence". Using these definitions "Virtual Reality" means "to have the effect of concrete existence without actually having concrete existence", which is exactly the effect achieved in a good Virtual Reality system. There is no requirement that the virtual environment match the real world. Inspired by these considerations, for the virtual dance lessons simulator we adapt the following definition:

Virtual Reality is the use of computer technology to create the effect of an interactive three-dimensional world in which the objects have a sense of spatial presence.

In this definition, "spatial presence" means that the objects in the environment effectively have a location in three-dimensional space relative to and independent of user's position. It should be noted that this is an effect, not an illusion. The basic idea is to present the correct cues to the user perceptual and cognitive system so that his brain interprets those cues as objects "out there" in the three-dimensional world. These cues have been surprisingly simple to provide using computer graphics: simply render a three-dimensional object (in stereo) from a point of view which matches the positions of users' eyes as they move about. If the objects in the environment interact with the user then the effect of spatial presence is greatly heightened. The immersive environment can be similar to the real world in order to create a lifelike experience grounded in reality or sci-fi. Some applications the sense of immersion is highly desirable. The main point of Virtual Reality, and the primary difference between conventional three-dimensional computer graphics and Virtual Reality is that in Virtual Reality the user is working with things as opposed to pictures of things.

Virtual Reality is an artificial environment that is created with software and presented to the user in such a way that the user suspends belief and accepts it as a real environment. On a computer though, Virtual Reality is primarily experienced through two of the five senses: sight and sound. Virtual Reality means creating immersive, computer-generated environments that are so convincing users that they will react the same way they would in real life. The idea is to block out the sensory input from the outside and use the visual and auditory cues to make the virtual world seem more real. While the concept is simple, actually building Virtual Reality systems has proven difficult to do, until recently.

2.2.1 VR Immersion

Immersion is basically an unique experience that is connected with the world of Virtual Reality. Over here the user whole exploring the three dimensional world of Virtual Reality will simply immerse into this make believe world as the real world. It is basically a feeling of involvement of the user in the virtual world intelligently designed by experts.

They have the power to interact with this world. This unique combinations where the user can immerse as well interact with the simulations is known as Telepresence. This is devise by the famous computer scientist Jonathan Steuer. Thus the user forgets about his real world scenario, forgets his present identity, situation and life and is immersed in a world of imagination, adventure and exploration. He/She gets more focused about this newly created identity inside the Virtual Reality world. Immersion is made up of two main components as stated by Jonathan Steuer. They are:

- Depth of Information
- Breadth of Information

While a user is using simulations and interaction between the user and the virtual environment takes place then some amount of quality of data are received in the signals. These information are termed as Depth of Information. Depth of information can necessarily include anything and everything starting from the resolution of the display unit, the graphics quality, the effectiveness of the audio and video etc.

Jonathan Steuer also defines breadth of information as a number of sensory dimensions presented simultaneously. Any virtual environment can be designated as having a wider breadth of information whenever it stimulates all the human senses. The user should get fully focused onto the new identity and world he explores. The audio and visual effects are the mostly researched area in creating a good virtual environment. These are considered as the main factors that can stimulate user's all sensory organs. The sense of touch is been given more and more priority as it has become the dominating factor to stimulate a human. Those systems that allow the users to interact through touch are known as Baptic Systems.

It is also necessary from the users' perspective to explore the life –sized virtual environment fully and effectively. This will prove how effective is Immersion. The perspectives also need to be changed seamlessly. Say for example there is a room in the virtual environment and it contains a pedestal right in the middle of the room then the pedestal should be visible to the user from any angle of the room. The point of view will also move accordingly wherever the angle of the user is changing. However Dr. Frederick Brooks, a legend in VR technology and theory comments that displays must project a frame rate of at least 20 – 30 frames per second in order to produce an efficient user experience. [3]

2.2.2 Frame Rate

Frame Rate, commonly measured and referred to as frames-per-second (FPS), is the frequency at which a hardware device is able to draw or capture consecutive images , called frames. The term is usually used when describing technical specifications of film and video cameras, computer graphics and motion capture systems.It is also a measure of performance of games and 3d applications. In video, film computer graphics and even more in Virtual Reality, Frame Rate output is critical, as it decides whether consecutive frames will be perceived by the brain as separate images or not, thus giving the desired illusion of motion. Although the human visual system can theoretically process 1000 different images a second, studies show that untrained eyes can hardly tell any difference above 60fps or 100fps, depending on the display device and usage. For example Virtual Reality devices usually require a higher than usual fps rate to create the pleasant illusion of smooth motion.Finally, it is essential to note that achieving high frame rates is usually not an easy task , as it requires hardware with substantial processing power and thoughtful application programming.

In our project we used a minimal polygonal design displaying a low polygon environment, achieving a frame rate of more than **100 FSP** in order to offer a super realistic motion flow to the end user with a very low rate of lag while interacting (dancing) with the 3D character.

2.3 Head Mounted Displays (HMDs)

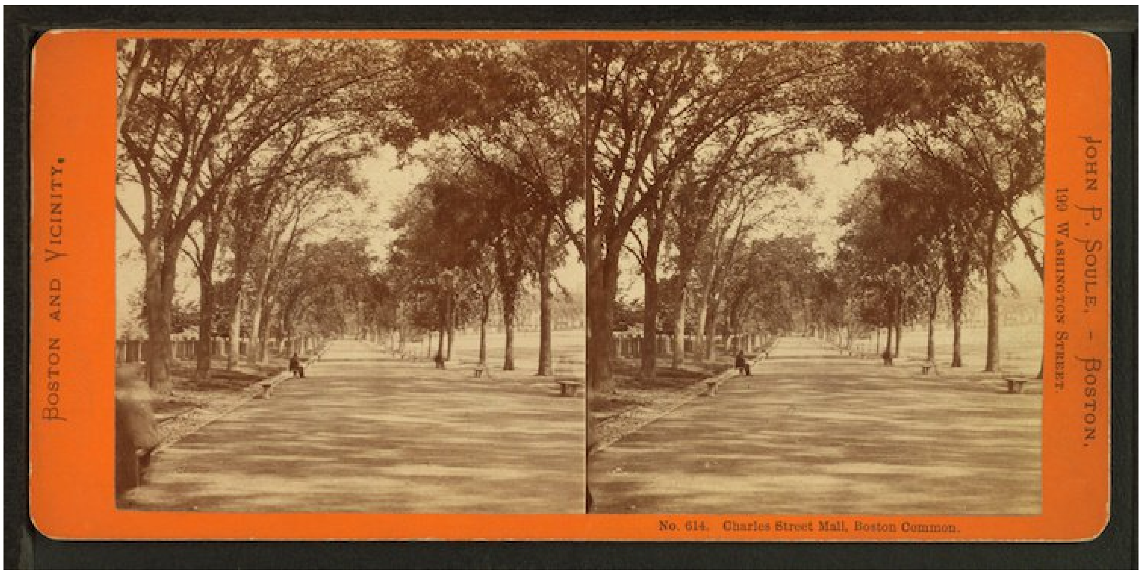
Virtual Reality has beginnings that preceded the time that the concept was coined and formalized. In this detailed history of Virtual Reality we look at how technology has evolved and how key pioneers have paved the path for Virtual Reality as we know it today.

2.3.1 History of HMDs

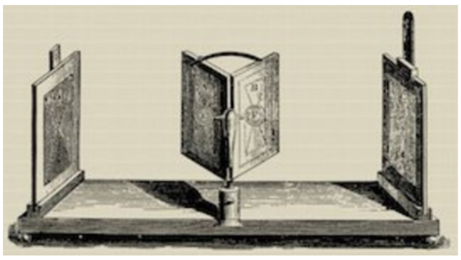
Early attempts at Virtual Reality

Panoramic paintings. If we focus more strictly on the scope of Virtual Reality as a means of creating the illusion that we are present somewhere we are not, then the earliest attempt at virtual reality is surely the 360-degree murals (or panoramic paintings) from the nineteenth century. These paintings were intended to fill the viewer's entire field of vision, making them feel present at some historical event or scene.

1838 - Stereoscopic photos & viewers. In 1838 Charles Wheatstone's research demonstrated that the brain processes the different two-dimensional images from each eye into a single object of three dimensions. Viewing two side by side stereoscopic images or photos through a stereoscope gave the user a sense of depth and immersion. The later development of the popular View-Master stereoscope (patented 1939), was used for "virtual tourism". The design principles of the Stereoscope is used today for the popular Google Cardboard and low budget VR Head Mounted Displays for mobile phones.



Over time mankind has been slowly but surely creating ever richer ways to stimulate our senses. Things really began to take off in the 20th century, with advent of electronics and computer technology.



1930s – Science fiction story predicted VR. In the 1930s a story by science fiction writer Stanley G. Weinbaum (Pygmalion's Spectacles) contains the idea of a pair of goggles that let the wearer experience a fictional world through holographics, smell, taste and touch. In hindsight the experience Weinbaum describes for those wearing the goggles are uncannily like the modern and emerging experience of Virtual Reality, making him a true visionary of the field.

PYGMALION'S SPECTACLES

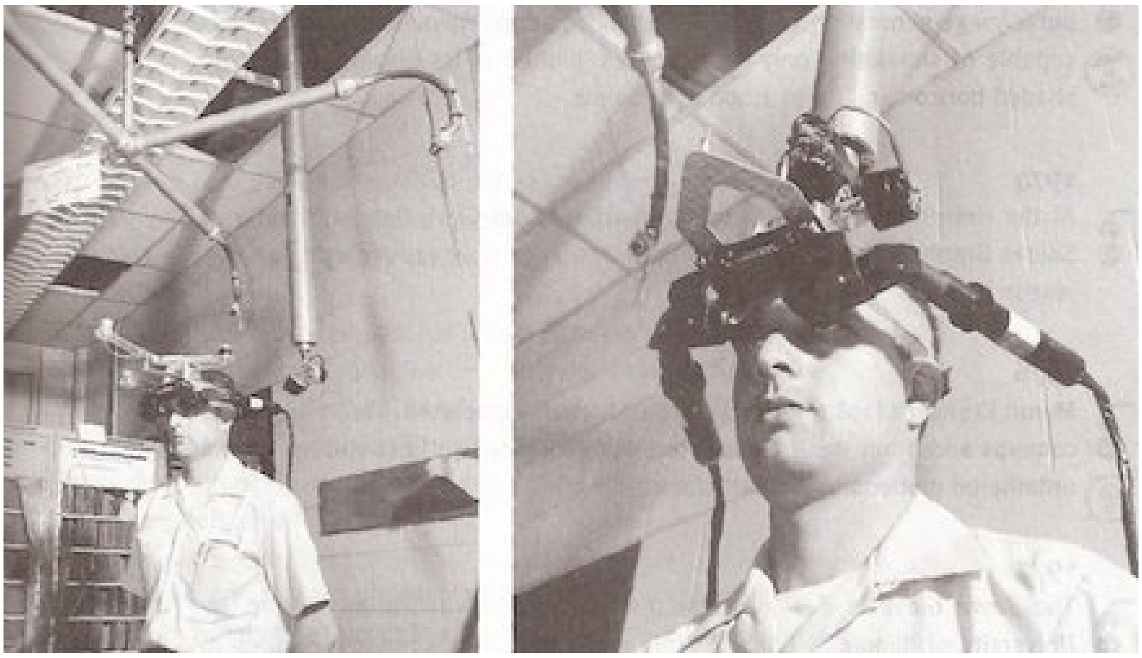
By STANLEY G. WEINBAUM

Author of "The Black Flame," "A Martian Odyssey," etc.

© 1935 by Continental Publications, Inc.

Unbelieving, still gripping the arms of that unseen chair, Dan was staring at a forest

1968 – Sword of Damocles. In 1968 Ivan Sutherland and his student Bob Sproull created the first VR / AR Head Mounted Display (Sword of Damocles) that was connected to a computer and not a camera. It was a large and scary looking contraption that was too heavy for any user to comfortably wear and was suspended from the ceiling (hence its name). The user would also need to be strapped into the device. The computer generated graphics were very primitive wireframe rooms and objects.



1993 – SEGA announce new VR glasses. Sega announced the Sega VR headset for the Sega Genesis console in 1993 at the Consumer Electronics Show in 1993.

The wrap-around prototype glasses had head tracking, stereo sound and LCD screens in the visor. Sega fully intended to release the product at a price point of about $200 at the time, or about $322 in 2015 money. However, technical development difficulties meant that the device would forever remain in the prototype phase despite having developed 4 games for this product. This was a huge flop for Sega.

1999 – The Matrix. In 1999 the Wachowski siblings' film The Matrix hits theatres. The film features characters that are living in a fully simulated world, with many completely unaware that they do not live in the real world. Although some previous films had dabbled in depicting Virtual Reality, such as Tron in 1982 and Lawnmower Man in 1992, The Matrix has a major cultural impact and brought the topic of simulated reality into the mainstream.

2.3.2 Virtual Reality in the 21st century.

The first fifteen years of the 21st century has seen major, rapid advancement in the development of Virtual Reality. Computer technology, especially small and powerful mobile technologies, have exploded while prices are constantly driven down. The rise of smart-phones with high-density displays and 3D graphics capabilities has enabled a generation of lightweight and practical Virtual Reality devices. The video game industry has continued to drive the development of consumer Virtual Reality unabated. Depth sensing cameras sensor suites, motion controllers and natural human interfaces are already a part of daily human computing tasks. Recently companies like Google have released interim Virtual Reality products such as the Google Cardboard in Figure (2.2a), a DIY headset that uses a smartphone to drive it. Companies like Samsung have taken this concept further with products such as the Galaxy Gear, which is mass produced and contains "smart" features such as gesture control.

Developer versions of final consumer products have also been available for a few years, so there has been a steady stream of software projects creating content for the immanent market entrance of modern Virtual Reality.

It seems clear that 2016 was a key year in the Virtual Reality industry. Multiple consumer devices that seem to finally answer the unfulfilled promises made by Virtual Reality in the 1990s are already available to market since that time. These include the pioneering Oculus Rift (Figure 2.2b), which was purchased by social media giant Facebook in 2014 for the staggering sum of $2BN. An incredible vote of confidence in where the industry is set to go. The Oculus Rift release in 2016 competes with products from Valve corporation and HTC (Figure 2.2c), Microsoft as well as Sony Computer Entertainment. These heavyweights are sure to be followed by many other enterprises, should the market take off as expected.

Many current generation HMDs come with features such as positional-tracking, head-tracking, hand-tracking and even eye-tracking. The more sophisticated the HMD, the more of the above technologies are incorporated, providing users remarkable simulation experiences.

In our project, we developed our application based on the Oculus Rift CV1, which works perfectly with most industry standard game engines, especially with Unity3D.

(a) Google cardboard (b) Oculus Rift (c) HTC Vive

Figure 2.2: VR Devices in 2017

Positional tracking. This technology allows the HMD and/or the computer to have knowledge of real-time positioning of a user in real 3D space. This can be very useful for certain applications or games, as someone could move in the virtual world by simply walking or moving around in reality. This is achieved by constantly measuring distance and angles between fitted sensors inside the HMD in respect to one or more base stations located in a room. Then, mathematical calculations translate the measurements to the exact position of the user in the room (Figure 2.3).

Our game makes use of the positional tracking in setting the transform of the user in the virtual environment through his real-world movements.

Orientation tracking. Orientation tracking is one of the most important features of an HMD. This is what allows the user to look-around a virtual world by simply moving/tilting the head as we normally do in our lives, without the need of separate controllers, thus giving the illusion of presence in the virtual environment. A head-tracking system typically consists of components such as accelerometers, gyroscopes and magnetometers,which are also used in nowadays smart-phones. An external stationary sensor may also be used (see Figure 2.3).

Head tracking as well as positional tracking has been employed in the presented dance training application. The movements and the rotation of the player's head is used to move and rotate the Camera in the VR 3D scene.

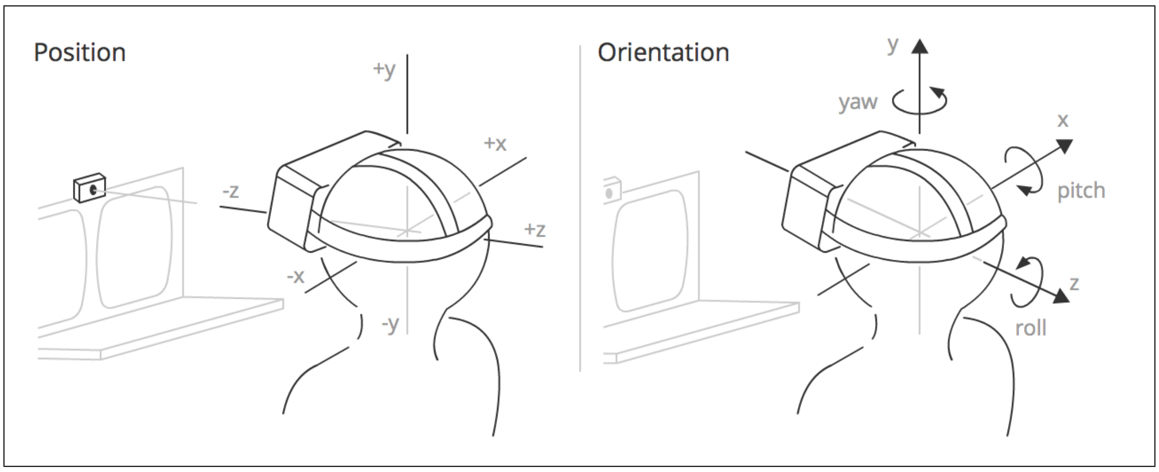


Figure 2.3: Position and Orientation Tracking

2.4 Motion Capture

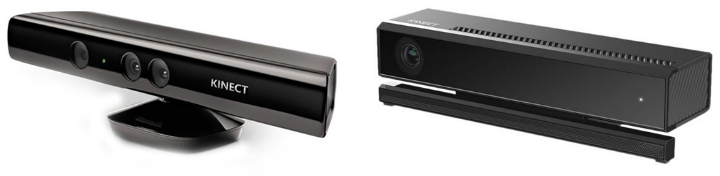
The use of motion capture for computer character animation is relatively new, having begun in the late 1970's, and only now beginning to become widespread.

Motion capture is the recording of human body movement (or other movement) for immediate or delayed analysis and playback. The information captured can be as general as the simple position of the body in space or as complex as the deformations of the face and muscle masses. Motion capture for computer character animation involves the mapping of human motion onto the motion of a computer character. The mapping can be direct, such as human arm motion controlling a character's arm motion, or indirect, such as human hand and finger patterns controlling a character's skin color or emotional state.

The idea of copying human motion for animated characters is, of course, not new. To get convincing motion for the human characters in Snow White, Disney studios traced animation over film footage of live actors playing out the scenes. This method, called rotoscoping, has been successfully used for human characters ever since. In the late 1970's, when it began to be feasible to animate characters by computer, animators adapted traditional techniques, including rotoscoping. At the New York Institute of Technology Computer Graphics Lab, Rebecca Allen used a half-silvered mirror to superimpose videotapes of real dancers onto the computer screen to pose a computer generated dancer for Twyla Tharp's "The Catherine Wheel." The computer used these poses as keys for generating a smooth animation. Rotoscoping is by no means an automatic process, and the complexity of human motion required for "The Catherine Wheel," necessitated the setting of keys every few frames. As such, rotoscoping can be thought of as a primitive form or precursor to motion capture, where the motion is "captured" painstakingly by hand. [4]

2.4.1 Methods and Systems

Motion tracking or motion capture started as a photogrammetric analysis tool in biomechanics research in the 1970s and 1980s, and expanded into education, training, sports and recently computer animation for television, cinema, and video games as the technology matured. Since the 20th century the performer has to wear markers near each joint to identify the motion by the positions or angles between the markers. Acoustic, inertial, LED, magnetic or reflective markers, or combinations of any of these, are tracked, optimally at least two times the frequency rate of the desired motion. The resolution of the system is important in both the spatial resolution and temporal resolution as motion blur causes almost the same problems as low resolution. Since the beginning of the 21st century and because of the rapid growth of technology new methods were developed. Most modern systems can extract the silhouette of the performer from the background. Afterwards all joint angles are calculated by fitting in a mathematic model into the silhouette. For movements you can't see a change of the silhouette, there are hybrid Systems available who can do both (marker and silhouette), but with less marker. A range of suits are now available from various manufacturers and base prices range from $1,000 to $80,000 USD. [5]

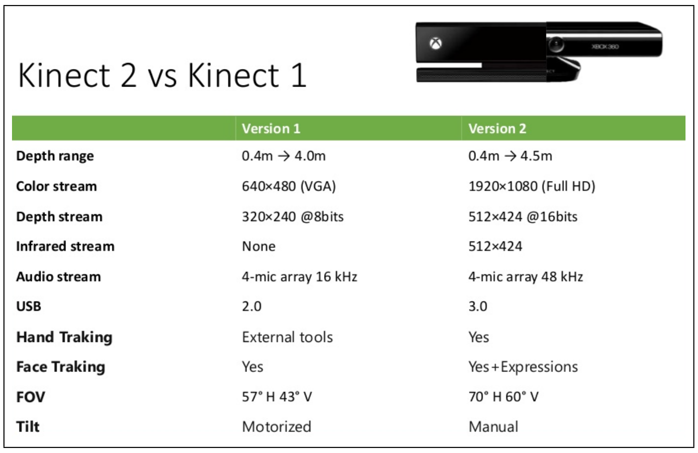


(a) Kinect v1 (b) Kinect v2

Figure 2.4: Microsoft Kinect Devices

Optical Systems

Kinect (codenamed Project Natal during development) is a line of motion sensing input devices by Microsoft for Xbox 360 (Figure 2.4a) and Xbox One (Figure 2.4b) video game consoles and Microsoft Windows PCs that uses Depth Sensor. Based around a webcam-style add-on peripheral, it enables users to control and interact with their console/computer without the need for a game controller, through a natural user interface using gestures and spoken commands. [6]



Kinect 2 vs Kinect 1

	Version 1	Version 2
Depth range	0.4m → 4.0m	0.4m → 4.5m
Color stream	640×480 (VGA)	1920×1080 (Full HD)
Depth stream	320×240 @8bits	512×424 @16bits
Infrared stream	None	512×424
Audio stream	4-mic array 16 kHz	4-mic array 48 kHz
USB	2.0	3.0
Hand Traking	External tools	Yes
Face Traking	Yes	Yes+Expressions
FOV	57° H 43° V	70° H 60° V
Tilt	Motorized	Manual

Figure 2.5: Version2 and Version1 comparison

Depth sensor - An infrared projector and a monochrome CMOS (complimentary metal-oxide semiconductor) sensor work together to ”see” the room in 3-D regardless of the lighting conditions.

For the purpose of this thesis we started to use the Kinect V1 because of its direct compatibility with the animation software we used to create and edit the dance motions. After the upgrade we made to Kinect V2 we realized that there was no way to import real-time tracking data from the device to the 3D animation modeling software. The only way was to have a middle software such as the Kinect animation studio created from Antonio Carlos Furtado which is freeware software recording the raw data out of the Kinect V2 and store the motions we take separately. After a record we take only then we import the animation to Editor software.

2.4.2 Hand Tracking

Hand-tracking describes the process of constantly capturing users’ hands and movements in real 3D space. Although this technology is still making its first steps in the Virtual Reality field, HMD manufacturers are beginning to adopt it, as it opens up new horizons and opportunities for VR users and developers. It is very obvious that being able to interact with a virtual environment by simply using your hands will bring the realism and interactivity of Virtual Reality to a whole new level. It usually works by allowing the user to see a 3D computer generated replica of his hands in the virtual world, according to their current position and rotation in real 3D space. For example, a user could point his (real) finger to touch a virtual button, thrust his hand forward to punch a virtual enemy or make a gesture to trigger an event.

There is two different way we cant have a hand-tracking movements:

- **The first technique** requires the user constantly hold special controllers with his hands, which act as positional-tracking devices for each individual hand. A similar process to positional-tracking is followed by measurements taken between the controllers and base stations in a room. Then, the position of each hand can be calculated in relativity to the head and room and be projected in the virtual world.

Figure 2.6: Vive and Oculus Touch controllers

- **The second technique** involves a design similar to eye-tracking which includes image capturing and image processing. One or more camera sensors located on the HMD point towards the users hands , capturing their movement in several frames per second. Then advanced mathematical and image processing algorithms analyze each image/frame, producing a 3d representation of how the device sees the hands.

Figure 2.7: Leap Motion IR Controller

For this project, we selected the second technique using the Leap Motion device as it can mount to the HMD, thus, allowing the users hands to be free for the dance interaction. The Leap Motion device allow users to make realistic touch movements while interacting with the UI and 3D character. Generally the Kinect and the LeapMotion are both tracking devices. In the current project we needed to use both tracking devices. The reason is that the Kinect has minimum tracking distance at 60cm so it was not possible to track finger movements accurately especially when the user turns around and hands are not visible to tracking device. Because of the Kinect's size, it was not possible for it to be mounted the HMD. For this reason, the Leap Motion was mounted onto the HMD so that tracking data are independent of the orientation and position of the user thanks to the tiny size of this device.

2.5 3D Computer Graphics

2.5.1 3D Rendering

Converting information about 3D objects into a graphics image that can be displayed is known as rendering. It usually requires considerable memory and processing power. It is the process of adding realism to computer graphics by adding three-dimensional qualities such as light, shadows and variations in color and shade. This process is usually performed using 3D computer graphics software. There are many rendering methods that have been developed, each one appropriate for specific applications. There is the non photo-realistic rendering, which gives the effect of painting, drawing or cartoons, and the rendering methods aiming to achieve high photo-realism. Another categorization is suitability for real time rendering and non real time rendering. Non real time rendering is implemented in non interactive media such as films and video. In this case, the rendering process can take huge amounts of time. That is because non real time rendering has the advantage of very high quality even with limited processing power due to the absence of real time response, which makes the time for the rendering process not considerable. A method suitable for non real time rendering is ray tracing, which simulates the path of a single light ray as it would be absorbed or reflected by various objects in the scene. Real time rendering is implemented in interactive media such as games and simulations. The calculations and the display are happening in real time. The primary goal is to achieve an as high as possible degree of photo-realism at an acceptable rendering speed. This is 24 frames per second, as that is the minimum the human eye needs to see to successfully create the illusion of movement.

2.5.2 Global Illumination (GI)

Global illumination (shortened as GI) or indirect illumination is a general name for a group of algorithms used in 3D computer graphics that are meant to add more realistic lighting to 3D scenes. Such algorithms take into account not only the light which comes directly from a light source (direct illumination), but also subsequent cases in which light rays from the same source are reflected by other surfaces in the scene, whether reflective or not (indirect illumination).

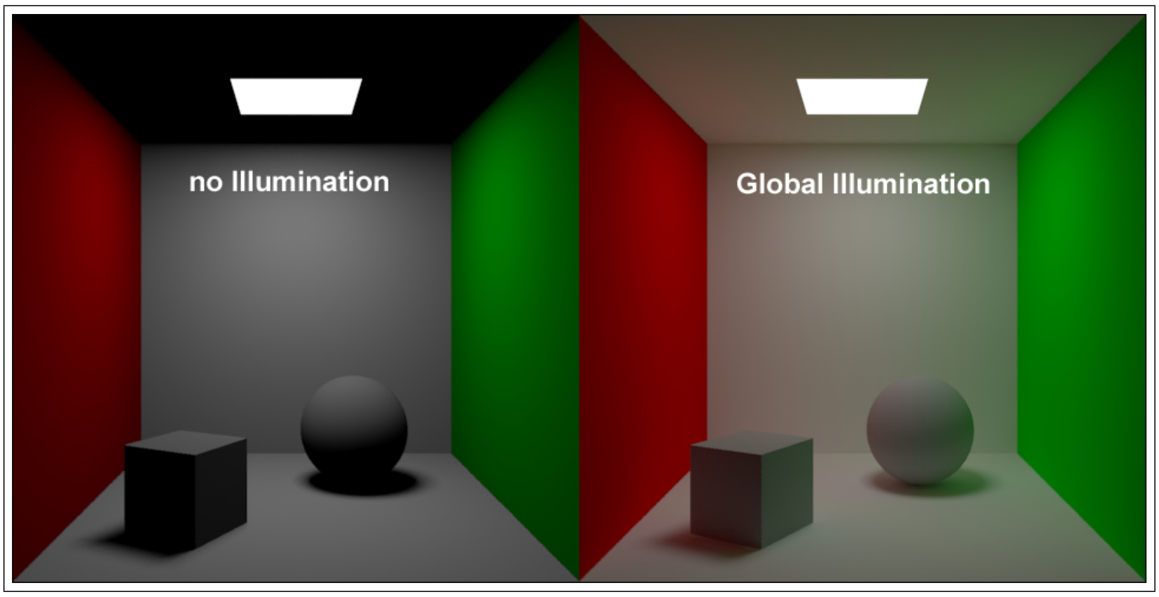


Figure 2.8: Global Illumination effect

Images rendered using global illumination algorithms often appear more photo-realistic than images rendered using only direct illumination algorithms. However, such images are computationally more expensive and consequently much slower to generate. One common approach is to compute the global illumination of a scene and store that information with the geometry, e.g., radiosity. That stored data can then be used to generate images from different viewpoints for generating walkthroughs of a scene without having to go through expensive lighting calculations repeatedly.

Radiosity, ray tracing, beam tracing, cone tracing, path tracing, Metropolis light transport, ambient occlusion, photon mapping, and image based lighting are examples of algorithms used in global illumination, some of which may be used together to yield results that are not fast, but accurate. These algorithms model diffuse inter-reflection which is a very important part of global illumination; however most of these (excluding radiosity) also model specular reflection, which makes them more accurate algorithms to solve the lighting equation and provide a more realistically illuminated scene, see Figure (2.8).

In our project we applied Global Illumination on the decoration boxes appearing in the 3D environment. We applied baked lights for optimization so that we improve the FPS of the game play.

2.6 Game Engines

Unity 3D

Unity programming environments supports both 2D and 3D app development, which is quite unusual for a game engine. That said, Unity was really designed for 3D games with 2D support bolted on afterwards; the 2D features were initially just for building menus and other 2D screens needed in a 3D game, to avoid the need for an external tool. The features were quite generic and developers started building games with them; probably due to the broad cross-platform support. To their credit, Unity has supported this and continue to invest in the area.

Three development languages are officially supported: C#, UnityScript (basically JavaScript with type annotations) and Boo. The last of these, Boo is not widely used and probably best avoided. The Unity's development kit community has widely adopted C# and the majority of plugins and examples use it. If anyone prefers JavaScript and only has a very simple project in mind then UnityScript is a good option. After starting using plugins written in C# that potentially need to call back into UnityScript code, issues will probably come up with compilation order.

Additionally, Unity has a lot of great futures such as:

- Strong community of asset and plugin creators – there are a lot of free and reasonable priced content available.
- Visual editing tools are excellent and the editor can be extended with plugins.
- It supports a wide range of asset formats and converts automatically to optimal formats for the target platform.
- It supports a very wide range of platforms: mobile, desktop, web and console.
- Deployment to multiple platforms is very easy to manage.

- The 3D engine produces high quality results without any complex configuration.
- There is a free license that covers the majority of features.
- Paid licenses are very affordable for most professional developers, available on subscription for $75 per platform currently (some platforms are free).

On the other hand, there are some consequences, such as:

- Collaboration is difficult. Unity has an expensive asset server product to help teams collaborate. If somebody does not use it, sharing code and assets between team members can be painful. The best option is to enable and use external source control but there are several binary files that cannot be merged and updating assets often causes them to break things in scenes, losing connections to scripts and other objects.
- Performance is not great until very recently Unity ran almost entirely in a single thread and made almost no use of the extra cores in most mobile devices – this is improving in Unity 5. The compilers are not at all well optimized for the ARM processors in almost all mobile devices – Unity have decided to transpire to C++ and use LLVM to get a more optimized build rather than solve this problem directly in future releases.
- The engine source code is not available. Even paying users do not get to see the Unity source code, which means if users come across a bug in the engine they have to wait for them to fix it or work around it. It is always going to be more critical for users than it is for them. This also limits the ways in which user can extend or customize the engine.

Unreal Engine

Unreal is one of the most popular game engines to develop high-end triple-A titles for years now. Gears of War, Batman: Arkham Asylum, Mass Effect, and many other blockbuster games were developed with this engine. Below are its pros and cons.

Some of the best features of Unreal Engine are:

- With so many developers using it, Unreal offers the largest community support. Several lifetime hours of video tutorials and assets are available.
- Best support and update mechanism of all engines, with a new tool introduced with each new update.
- There is widest range of easy to maneuver tools up under its sleeve. There are few tools that can be maneuvered even by a school kid.

- Compatible with diverse operating platforms including iOS, Android, Linux, Mac, Windows, and most game consoles.
- The new licensing terms of $19 a month and a 5 percent royalty only if user's game makes over $5,000 make Unreal Engine 4 much more competitive than it had been in the past.

On the contrary, there are some developers who complain a lot about the unfriendly tools that involve a bit of a higher learning curve.

CryEngine

This game engine has received praise for beautiful graphics output. If somebody has a knack for pretty game visuals, this can be the ideal game engine for him. But this powerful game engine has its problems, too. Some of the pros are:

- CryEngine makes the game ambience pretty with its artist-level programming capability in its Flowgraph tool.
- It has the most powerful audio tool, Fmod, inside it, so sound designers love this engine as well as programmers.
- The game engine also offers the easiest A.I. coding of any tech on the market.
- For a beginning developer, UI scale form comes handy.

On the other hand, there are some cons too:

- The free version of the game engine lacks proper customer support.
- Being relatively new to the industry, the engine is yet to find a robust community.
- Learning curve is pretty challenging for a starter.

There are a lot of game engines that offer a plethora of features to users such as: HeroEngine, Rage Engine, Project Anarchy, GameSalad, GameMaker: Studio, App Game Kit, Cocos 2D and more, but these are not going to be presented in the context of this thesis.

Unreal Engine and Unity are currently ahead of the competition as the two most popular game engines available to the public. This is due to the fact that they both succeed in providing high-end graphics, a large variety of usable tools, great support for platforms and devices, without compromising usability and efficiency. It is important to note that these 2 Game Engines offer a large community support, which is also something that has to be considered when choosing the right Game Engine. CryEngine is also great and powerful engine with remarkable capabilities, however its complicated structure and smaller community excluded it from our consideration.

In conclusion, taking into account the advantages and disadvantages of each engine, Unity proved to be the ideal choice for this project, mainly due to its efficiency, large community support and ease of use.

2.7 Character Animation

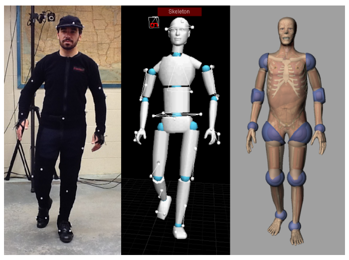
Animation in 3D applications usually happens in two primary ways (**Keyframe animation** and **Motion Capture**). In major productions, both may be used.

Keyframe animation - Keyframe animation, or keyframing, is the most well-known and oldest style of animation. In fact, there are examples of frame-by-frame animation dating all the way back to 1600 B.C. Egypt! Modern keyframing techniques date back to the early cartoons created by animation pioneers like Winsor McCay and Walt Disney. What may surprise you is that keyframing techniques have not changed much since the early 1900's - most of the basic principles still apply today. What has changed is that 3D software packages have made keyframing much easier to accomplish, meaning a broader scope of artists can learn how to animate.

Keyframing is essentially changing the shape, position, spacing, or timing of an object in successive frames, with major changes to the object being the key frames. In traditional 2D animation, each frame is usually drawn by hand. When frames are shown in succession, as in a movie, the slight differences in each frame of animation create the illusion of motion. 3D software packages make keyframe animation easier by interpolating, or "tweening," the in-between frames. When animating a falling ball, for example, one key frame might be of the ball in mid-air, the next key frame may be the ball touching the ground, and the key frame after that would be the ball squishing down as the impact deforms its shape. All of the in-between frames are then calculated by the software automatically, including the squish at the bottom, making actual process of animation a matter of creating a few great key frames.

Motion Capture - Motion capture, or mocap, was first used sparingly due to the limitations of the technology, but is seeing increased acceptance in everything from video game animation to CG effects in movies as the technique matures. Whereas keyframing is a precise, but slow animation method, motion capture offers an immediacy not found in traditional animation techniques. Mocap subjects, usually actors, are placed in a special suit containing sensors that record the motion of their limbs as they move. The data is then linked to the rig of a 3D character and translated into animation by the **3D software** such us (**Autocad 3DSMax, MotionBuilder, Maya**).

There are a couple downsides to motion capture which make it difficult for beginning 3D animators to learn. Firstly is the cost of mocap technology, which can run several thousands or even tens of thousands of dollars whitch use markers (see Figure 2.9a) while more cheaper technologies exist like capture artist motions without markers using depth sensor cameras (see Figure 2.9b). This means that most new 3D artists must learn to incorporate this animation style by importing mocap data into a project from a commercially available mocap library.


(a) Marker based mocap

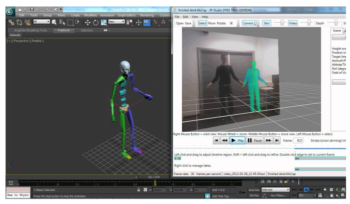

(b) Marker less

Figure 2.9: Motion Capture of Character

The other downside to mocap is that the end-result is often far from perfect; mocap animation usually requires clean-up from key-frame artists to make it look more realistic, especially if the character being animated does not have an anatomy or proportions similar to those of a human.

In this project, we used both techniques applying the key frame animations to the UI components of the 3D environment and Motion Capture animations to capture human motions.

Inverse Kinematics

Most animation is produced by rotating the angles of joints in a skeleton to predetermined values. The position of a child joint changes according to the rotation of its parent and so the end point of a chain of joints can be determined from the angles and relative positions of the individual joints it contains. This method of posing a skeleton is known as **Forward Kinematics (FK)**.

However, it is often useful to look at the task of posing joints from the opposite point of view - given a chosen position in space, work backwards and find a valid way of orienting the joints so that the end point lands at that position. This can be useful when you want a character to touch an object at a point selected by the user or plant its feet convincingly on an uneven surface. This approach is known as **Inverse Kinematics (IK)** and is supported in Mecanim for any humanoid character with a correctly configured Avatar.[7]

2.8 Latin Dance

2.8.1 Introduction

In many styles of salsa dancing, as a dancer shifts their weight by stepping, the upper body remains level and nearly unaffected by the weight changes. Weight shifts cause the hips to move. Arm and shoulder movements are also incorporated. The Cuban Casino style of salsa dancing involves significant movement above the waist, with up-and-down shoulder movements and shifting of the ribcage.

The arms are used by the "lead" dancer to communicate or signal the "follower," either in "open" or "closed" position. The open position requires the two dancers to hold one or both hands, especially for moves that involve turns, putting arms behind the back, or moving around each other, to name a few examples. In the closed position, the leader puts the right hand on the follower's back, while the follower puts the left hand on the leader's shoulder.

In the original Latin American form, the forward/backward motion of salsa is done in diagonal or sideways with the 3-step weight change intact.

In some styles of salsa, such as the New York style, the dancers remain mostly in front of one another (switching places), while in Latin American styles, such as Cuban style, the dancers circle around each other, sometimes in 3 points. This circular style is inspired by Cuban Son, specifically to the beat of son montuno in the 1920s. However, as it is a popular music, it is open to improvisation and thus it is continuously evolving. Modern salsa styles are associated and named to the original geographic areas that developed them. There are often devotees of each of these styles outside of their home territory. Characteristics that may identify a style include: timing, basic steps, foot patterns, body movement, turns and figures, attitude, dance influences and the way that partners hold each other. The point in a musical bar music where a slightly larger step is taken (the break step) and the direction the step moves can often be used to identify a style.

Incorporating other dance styling techniques into salsa dancing has become very common, for both men and women: shimmies, leg work, arm work, body movement, spins, body isolations, shoulder shimmies, rolls, even hand styling, acrobatics and lifts.

Latin American styles originate from Puerto Rico, Cuba and surrounding Caribbean islands.

2.8.2 Salsa Styles

Salsa's roots are based on different genres such as Puerto Rican rhythms, Cuban Son, specifically to the beat of Son Montuno in the 1920s. However, as it is a popular music, it is open to improvisation and thus it is continuously evolving. New modern salsa styles are associated and named to the original geographic areas that developed them. There are often devotees of each of these styles outside of their home territory. Characteristics that may identify a style include: timing, basic steps, foot patterns, body rolls and movements, turns and figures, attitude, dance influences and the way that partners hold each other. The point in a musical bar music where a slightly larger step is taken (the break step) and the direction the step moves can often be used to identify a style.

- Afro-Latino style
- Colombian / Cali style
- Cuban style / Casino
- Los Angeles style
- **New York style (on2)**

2.8.3 Rythm

For salsa, there are four types of clave rhythms, the 3-2 and 2-3 Son claves being the most important, and the 3-2 and 2-3 Rumba claves. Most salsa music is played with one of the Son claves, though a Rumba clave is occasionally used, especially during Rumba sections of some songs. As an example of how a clave fits within the 8 beats of a salsa dance, the beats of the 2-3 Son clave are played on the counts of 2, 3, 5, the "and" of 6, and 8.

There are other aspects outside of the Clave that help define salsa rhythm: the cowbell, the Montuno rhythm and the Tumbao rhythm.

The cowbell rhythm emphasizes the "on-beats" of salsa: 1, 3, 5 and 7 while the conga rhythm emphasizes the "off-beats" of the music: 2, 4, 6, and 8. Some dancers like to use the strong sound of the cowbell to stay on the Salsa rhythm. Alternatively, others like to use the conga rhythm to create a jazzier feel to their dance since strong "off-beats" are a jazz element.

Tumbao is the name of the rhythm that is played with the conga drums. It sounds like: "cu, cum.. pa... cu, cum... pa". Its most basic pattern is played on the beats 2, 3, 4, 6, 7 and 8. Tumbao rhythm is helpful for learning to dance contra-tiempo ("On2"). The beats 2 and 6 are emphasized when dancing On2, and the Tumbao rhythm heavily emphasizes those beats as well.

The Montuno rhythm is a rhythm that is often played with a piano. The Montuno rhythm loops over the 8 counts and is useful for finding the direction of the music. By listening to the same rhythm, that loops back to the beginning after eight counts, one can recognize which count is the first beat of the music.

The basic Salsa dance rhythm consists of taking three steps for every four beats of music. The odd number of steps creates the inherent syncopation to the Salsa dancing and ensures that it takes 8 beats of music to loop back to a new sequence of steps. Different styles employ this syncopation differently. For "On1" dancers this rhythm is described as "quick, quick, slow, quick, quick, slow." For "On2" dancers this rhythm is "quick, quick, quick, pause, quick, quick, quick, pause." In all cases, only three steps are taken in each 4-beat measure (or 6 total over 8 beats).[8]

Chapter 3

Cross platform Game Engine Software

Since the system as well as the demo training VR application are built using the Unity game engine, certain basic theory concepts about 3D graphics, utilized in this thesis, must be presented. 3D computer graphics (in contrast to 2D computer graphics) are graphics that utilize a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images. 3D computer graphics creation falls into three basic phases. The process of forming a computer model of an object's shape known as 3d modelling, the placement and movement of objects within a scene as well as the computer calculations that, based on light placement, surface types, and other qualities, generate the image. This image generating process is called the rendering process.

3.1 Unity Game Engine

For the purposes of this project, the Unity game engine has been selected to provide the development environment. Unity was selected because of its ease of use, the numerous online guides/tutorial, the ability to create 2D graphics for user interfaces, the familiar scripting language (C#), the Unity's Asset Store, where it is easy to find objects without having to create everything from scratch, the thriving and supportive community and last but not least the ability to use the full range of Game Engine tools and programming capabilities for free. The main components/windows of Unity Game Engine see Figure 3.1: Hierarchy window, Project window, Console window, Scene window, Game window, Inspector window,Toolbar, and most importantly, Scripting mechanisms. A more detailed description of each component follows.

3.1.1 Hierarchy

The Hierarchy window (see Figure 3.2 contains a list of every GameObject in the current Scene. Some of these are direct instances of Asset files (like 3D models), and others are instances of Prefabs, which are custom objects that make up most of the game. As objects are added in and removed from the Scene, they appear and disappear from the Hierarchy as well.

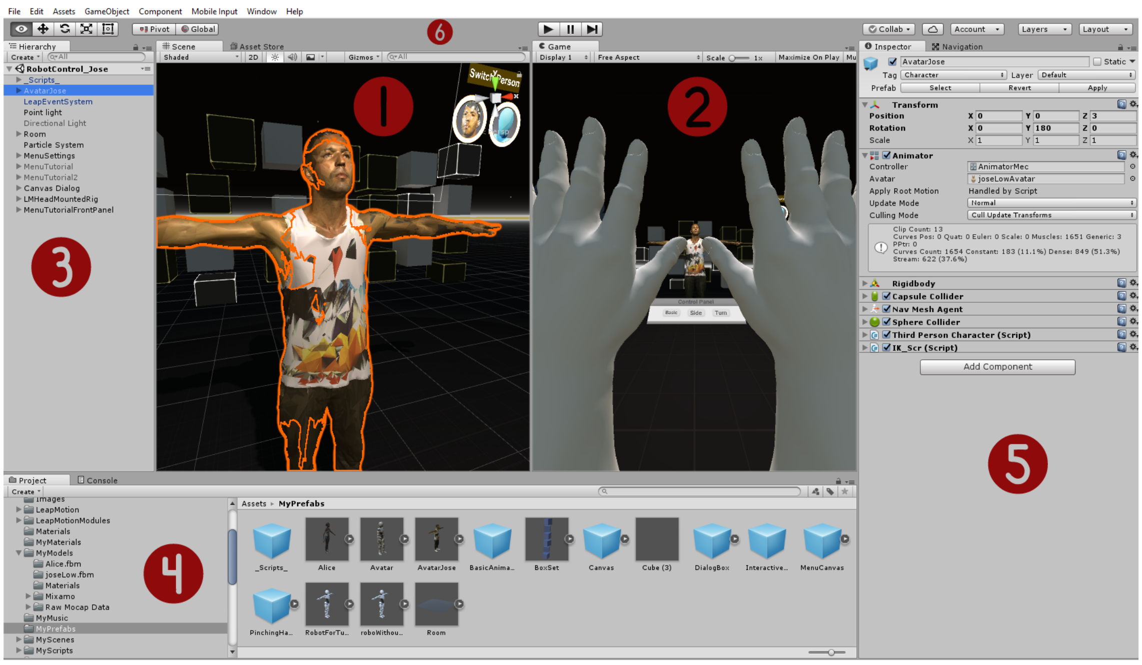


Figure 3.1: Main Layout of Unity Engine: 1.Scene, 2.Game, 3.Hierarchy, 4.Project, 5.Inspector, 6.Toolbar

By default, objects are listed in the Hierarchy window in the order they are created. Re-ordering of objects can be done easily by dragging them up or down, or by making them "child" or "parent" objects. For instance, in Figure 3.2 the Object 1 is the parent object and Object 2, Object 3 are children of it.

3.1.2 Project

In this window (see Figure 3.3), the user can access and manage the assets that belong to his project. It consists of two panels. The left panel of the browser shows the folder structure of the project as a hierarchical list. When a folder is selected from the list by clicking, its contents will be shown in the panel to the right panel. The user can click the small triangle to expand or collapse the folder, displaying any nested folders it contains.

The individual assets are shown in the right panel as icons that indicate their type (script, material, sub-folder, etc.). The icons can be re-sized using the slider at the bottom of the panel; they will be replaced by a hierarchical list view if the slider is moved to the extreme left. The space to the left of the slider shows the currently selected item, including a full path to the item if a search is being performed.

Just above the panel is a "breadcrumb trail" that shows the path to the folder currently being viewed. The separate elements of the trail can be clicked for easy navigation around the folder hierarchy. When searching, this bar changes to show the area being searched (the root Assets folder, the selected folder or the Asset Store) along with a count of free and paid assets available in the store, separated by a slash.

There is an option in the General section of Unity's Preferences window to disable the display of Asset Store hit counts if they are not required.

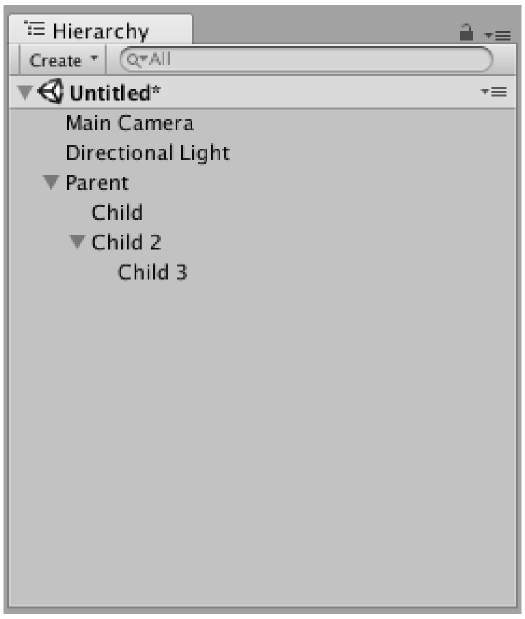


Figure 3.2: Hierarchy Window

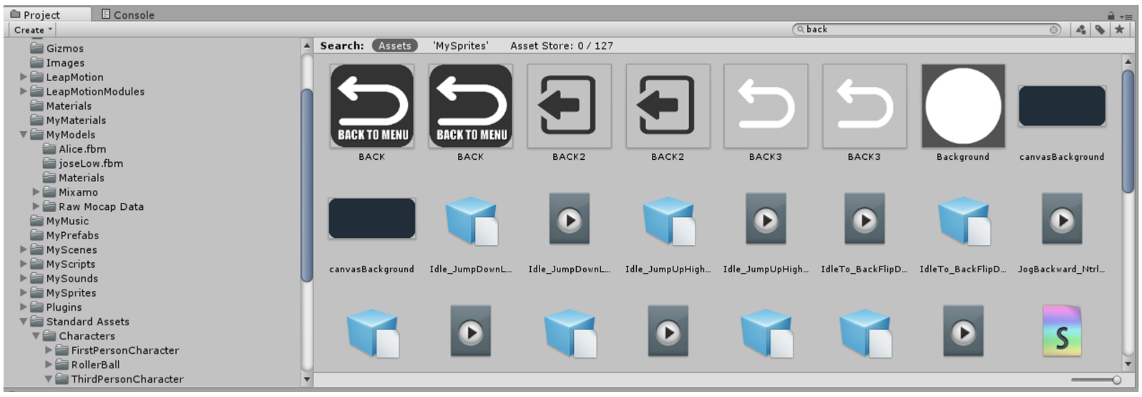


Figure 3.3: Project Window

3.1.3 Console

The Console Window shows errors, warnings and other messages generated by Unity. To aid with debugging, the user can also show his own messages in the Console using the implemented functions of Unity (Debug.Log, Debug.LogWarning and Debug.LogError).

The toolbar of the console window has a number of options that affect how messages are displayed. The Clear button removes any messages generated from user's code but retains compiler errors. The user can also arrange for the console to be cleared automatically whenever he runs the game by enabling the Clear on Play option. There is also the opportunity to change the way messages are shown and updated in the console. The Collapse option shows only the first instance of an error message that keeps recurring. This is very useful for runtime errors, such as null references, that are sometimes generated identically on each frame update. The Error Pause option will cause playback to be paused whenever Debug.LogError is called from a script. Finally, there are two options for viewing additional information about errors. The Open Player Log and Open Editor Log items on the console tab menu access Unity's log files which record details that may not be shown in the console.

3.1.4 Scene

The Scene window (see Figure 3.1 1.Scene) is the interactive view into the world that the user is creating. Scene View can be used to select and position scenery, characters, cameras, lights, and all other types of Game Objects. Being able to Select, manipulate and modify objects in the Scene View are some of the first skills somebody will need to begin his first steps in Unity.

The Scene Gizmo is in the upper-right corner of the Scene View. This displays the Scene View Camera's current orientation, and allows the user to quickly modify the viewing angle and projection mode.

In order to Move, Rotate, Scale, or Transform individual GameObjects, the user can use the four Transform tools in the toolbar (see Figure 3.1 6.Toolbar). Each has a corresponding Gizmo that appears around the selected GameObject in the Scene view. To alter the Transform component of the GameObject, the user can use the mouse to manipulate any Gizmo axis, or type values directly into the number fields of the Transform component in the Inspector.

3.1.5 Game

The Game window (see Figure 3.1 2.Game) is rendered from the Camera in user's game. It is representative of the final, published game. It is required for the user to use one or more Cameras to control what the player actually sees when they are playing the game.

3.1.6 Inspector

Projects in the Unity Editor are made up of multiple GameObjects that contain scripts, sounds, meshes, and other graphical elements such as lights. The Inspector window (sometimes referred to as "the Inspector") displays detailed information about the currently selected GameObject (see Figure 3.1 5.Insperctor), including all attached components and their properties, and allows the user to modify the functionality of GameObjects in the Scene. The user can use the Inspector to view and edit the properties and settings of almost everything in the Unity editor, including physical game items such as GameObjects, assets, and materials, as well as in-editor settings and preferences. When a GameObject is selected in either the Hierarchy or Scene view, the Inspector shows the properties of all components and materials of that GameObject. Actually, the Inspector can be used to edit the settings of these components and materials.

3.1.7 Project structure in Unity3D

Unity is defined by its component based architecture. Its workflow builds around the structure of components. Each component has its own specific job, and can generally accomplish its task or purpose without the help of any outside sources.

Each game or application created in Unity is called a project. Each project consists of one or more scenes. Scenes contain the objects of the game. They can be used to create a main menu, individual levels, and anything else. Every scene is considered as a unique level. In each scene, the user can position the 3D models, construct the environment and essentially design most of the functionality. Every object placed in a scene is considered a **GameObject**. GameObject sconsist of one or more **Components**. Components are Unity's fundamental elements, which are used to define properties, behavior and characteristics of a GameObject. The user can add a wide variety of components in a GameObject to achieve the desired functionality. Some of the most commonly used components in Unity are presented next:

- **Transform Component**. Every GameObject contains a Transform Component. When creating a GameObject a transform component is added automatically. It is impossible to create a GameObject without one or remove it. The Transform Component is one of the most important and most frequently accessed Component. It defines a GameObject's position, rotation, and scale in the game world based on the x,y,z coordinate system.These parameters are initialized by hand and/or can modified in runtime by script to make objects move, rotate and more. It is important to note that when scripting functionality such as movement, Unity considers the Z axis as forward/backwards, Y axis as up/down and X axis as left/right.

- **Mesh Component**. Physics components allow the user to give objects realistic motion and reaction to collisions by simulating physics laws. Unity has NVIDIA PhysX physics engine built-in. A physics engine is computer software that provides an approximate simulation of physical systems. This allows for unique realistic behavior and has many useful features. A rigidbody component makes the object that is attached to be affected by gravity or linear and angular forces and collide with other objects. There is also a variety of collider components (mesh, box, sphere, wheel collider) which surround the shape of an object for the purposes of detecting physical collisions. In this project, due to its simulation nature, a physics component was attached on the Aircraft GameObject.

- **Rendering Component**. These are the components that have to do with rendering in-game and user interface elements, as well as lighting and special effects. The camera component is essential as it is used to capture and display the world to the player. It can be customized and manipulated to fulfill the requirements of the users application. The GUI Texture and GUI Text components are made especially for user interface elements, buttons, or decorations as well as displaying text on screen. Another important rendering component is the light component. It brings a sense of realism. Lights can be used to illuminate the scenes and objects, to simulate the sun, flashlights, or explosions just to name a few.

- **Audio Component**. These components are used to implement sound. The most important component here, is the Audio Source component, which as the name suggests, plays a sound file at the location of the GameObject it is attached to. The developer can set parameters such as sound volume, pitch and change the sound file to be played at any time. These parameters can also be changed by script during run-time.

- **Script Component**. The script component is used to attach a script onto a GameObject. Scripts are often attached to objects, to define their behavior and trigger effects upon specified conditions. More about scripts in the following section.

- **Materials and Shaders**. Materials and shaders are crucial components that are categorized in the asset component group. There is a close relationship between materials and shaders. Materials are used in conjunction with mesh renderers and other rendering components used in Unity. They play an essential part in defining how the object is displayed. The properties that a materials inspector displays are determined by the shader that the material uses. A shader is a specialized kind of graphical program that determines how texture and lighting information are combined to generate the pixels of the rendered object onscreen. In other words, it tells the graphics hardware how to render surfaces.The user can select which shader each material will use. Specifically, a material defines which texture and color to use for rendering, whereas the shader defines the method to render an object.

3.1.8 Navigation and Pathfinding

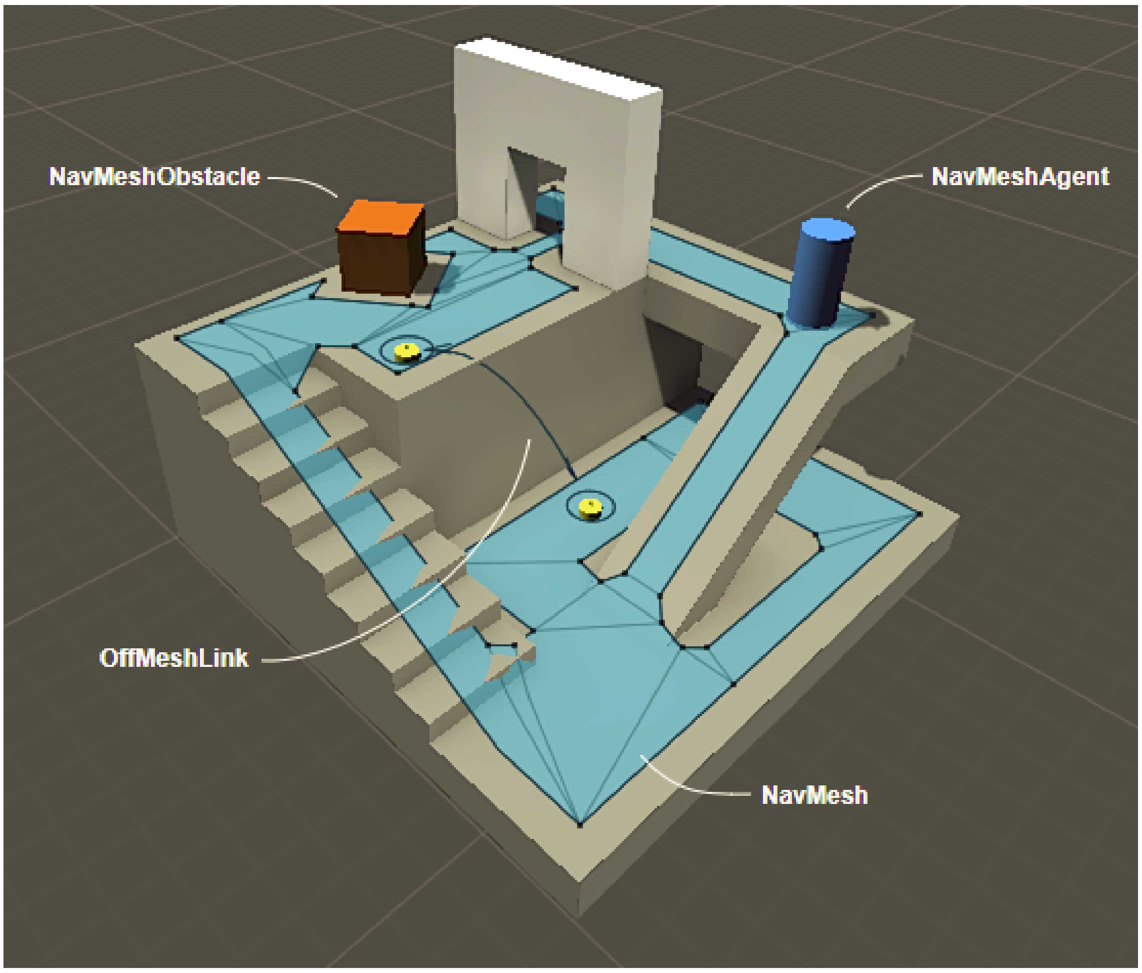


Figure 3.4: Navigation Mesh Overview

The navigation system allows you to create characters that can intelligently move around the game world, using navigation meshes that are created automatically from your Scene geometry. Dynamic obstacles allow you to alter the navigation of the characters at runtime, while off-mesh links let you build specific actions like opening doors or jumping down from a ledge.

The Navigation System allows you to create characters which can navigate the game world. It gives your characters the ability to understand that they need to take stairs to reach second floor, or to jump to get over a ditch. The Unity NavMesh system consists of the following pieces:

- **NavMesh** (short for Navigation Mesh) is a data structure which describes the walkable surfaces of the game world and allows to find path from one walkable location to another in the game world. The data structure is built, or baked, automatically from your level geometry.
- **NavMesh Agent component** help you to create characters which avoid each other while moving towards their goal. Agents reason about the game world using the NavMesh and they know how to avoid each other as well as moving obstacles.
- **Off-Mesh Link** component allows you to incorporate navigation shortcuts which cannot be represented using a walkable surface. For example, jumping over a ditch or a fence, or opening a door before walking through it, can be all described as Off-mesh links.
- **NavMesh Obstacle** component allows you to describe moving obstacles the agents should avoid while navigating the world. A barrel or a crate controlled by the physics system is a good example of an obstacle. While the obstacle is moving the agents do their best to avoid it, but once the obstacle becomes stationary it will carve a hole in the navmesh so that the agents can change their paths to steer around it, or if the stationary obstacle is blocking the path way, the agents can find a different route.

3.1.9 Scripting

Scripting is an essential part of Unity as it defines the entire behavior of the game or application. Even the simplest game needs a script to respond to user's input. Scripts can be used for several reasons such as: to create graphical effects, to control physical behavior of objects or characters, to trigger effects upon specified conditions or even implement a custom AI system for characters in the game.

The behavior of GameObjects is controlled by the Components that are attached to them. Although Unity's built-in Components can be very versatile, the programmer will soon find he needs to go beyond what they can provide to implement custom game play features. Unity allows the user to create custom Components using scripts. These trigger game events, modify Component properties over time and respond to user input in any way he could probably want to.

Unity supports two programming languages natively, the C#, an object oriented programming language similar to Java or C++ and the UnityScript, a language designed specifically for use with Unity and modelled after JavaScript. The scripts can be written and edited in MonoDevelop, which is an integrated development environment (IDE) within Unity or in any other IDE like Visual Studio. An IDE combines a text editor with additional features for debugging, auto-complete and other project management tasks.

A script makes its connection with the internal workings of Unity by implementing a class which derives from the built-in class called MonoBehavior. The reader can think of a class as a kind of blueprint for creating a new Component type that can be attached to GameObjects. Each time the programmer attaches a script component to a GameObject, it creates a new instance of the object defined by the blueprint. The name of the class is taken from the name that the programmer supplied when the file was created. The class name and file name must be the same to enable the script component to be attached to a GameObject.

Thus, components can be accessed or modified by script at any time to achieve desired behavior and functionality. When a script is created, there are two functions automatically declared in it, the ***Start()*** function and the ***Update()*** function. The Update function is the right place to write code that will handle the frame update for the GameObject. This might include movement, triggering actions and responding to user input, basically anything that needs to be handled over time during game play. To enable the Update function to do its work, it is often useful to be able to set up variables, read preferences and make connections with other GameObjects before any game action takes place. The Start function will be called by Unity when a script is enabled and will be called exactly once in its lifetime. The Start function is the ideal place where initialization occurs. It is used to initialize an object's position, state and properties or load other scripts and GameObjects for later use.

A script in Unity is not like the traditional idea of a program where the code runs continuously in a loop until it completes its task. Instead, Unity passes control to a script intermittently by calling certain functions that are declared within it. Once a function has finished executing, control is passed back to Unity. These functions are known as event functions since they are activated by Unity in response to events that occur during game play. Unity uses a naming scheme to identify which function to call for a particular event. For instance, we have already mentioned the Update function (called before a frame update occurs) and the Start function (called just before the object's first frame update). Many more event functions are available in Unity; the following are some of the most common and important event.

- **Regular Update Events:** These events can make changes to position, state and behavior of objects in the game just before each frame is rendered. The *Update* function is the main place for this kind of code in Unity. *Update* is called before the frame is rendered and also before animations are calculated. For physics update, like adding force to a GameObject, the best option is to place the code in the *FixedUpdate* function which updates more frequently than the *Update* function. Sometimes the best place to write code is the *LateUpdate* function in order to be able to make additional changes at a point after the *Update* and *FixedUpdate* functions have been called for all objects in the scene and after all animations have been calculated.

- **Initialization Events:** It is often useful to be able to call initialization code in advance of any updates that occur during game play. The Start function is called before the first frame or physics update on an object. The Awake function is called for each object in the scene at the time when the scene loads.

Note that although the various objects' Start and Awake functions are called in arbitrary order, all the Awakes will have finished before the first Start is called. This means that code in a Start function can make use of other initialization previously carried out in the awake phase.

- **GUI Events:** Unity has a system for rendering GUI controls over the main action in the scene and responding to clicks on these controls. This code is handled somewhat differently from the normal frame update and so it should be placed in the OnGUI function, which will be called periodically. For instance, a set of OnMouseXXX event functions (e.g., OnMouseOver, OnMouseDown) is available to allow a script to react to user actions with the mouse. For example, if the mouse button is pressed while the pointer is over a particular object then an OnMouseDown function in that object's script will be called if it exists.

- **Physic Events:** The physics engine will report collisions against an object by calling event functions on that object's script. The OnCollisionEnter, OnCollisionStay and OnCollisionExit functions will be called as contact is made, held and broken. The corresponding OnTriggerEnter, OnTriggerStay and OnTriggerExit functions will be called when the object's collider is configured as a Trigger (i.e., a collider that simply detects when something enters it rather than reacting physically). These functions may be called several times in succession if more than one contact is detected during the physics update and so a parameter is passed to the function giving details of the collision (position, identity of the incoming object, etc.).

Except for the functions that Unity provides, the developer can create his/her own functions in order to control or determine the behavior of a GameObject, change the properties of a component or altering the overall state of the application. In order for these custom functions to be executed, they have to be called inside a Unity event function, like the Update. The most commonly used functions were presented briefly above, as well as the concept of how they are used. The basic notion of the Unity scripting is that the scripts are components that can control the GameObject. Each component property corresponds to a script variable and the scripts can access not only the components of the GameObjects they are attached to, but also other GameObjects.

3.2 3D Modeling and Animating

For the purposes of this project, two 3D modeling tools have been used in order to create/modify or animate GameObjects and characters as well as for the Character Rigging. This tools are 3DSmax and MotionBuilder which offer free Academic/student licence.

Character Rigging is a very tedious process. It requires creativity, precision, and an eye for detail. As such, character riggers are responsible for using computer programs to form skeletons by creating a series of bones that deform and animate specific parts of the character.

- **3Dsmax :** 3D modeling and rendering software 3ds Max helps you create massive worlds in games, stunning scenes for design visualization, and engaging Virtual Reality (VR) experiences.

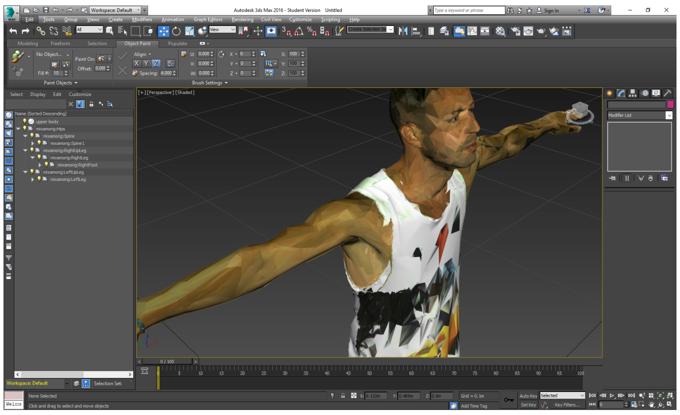


Figure 3.5: 3DsMax

- **MotionBuilder :** MotionBuilder is a professional 3D character animation software produced by Autodesk. It is used for virtual production, motion capture, and traditional keyframe animation.

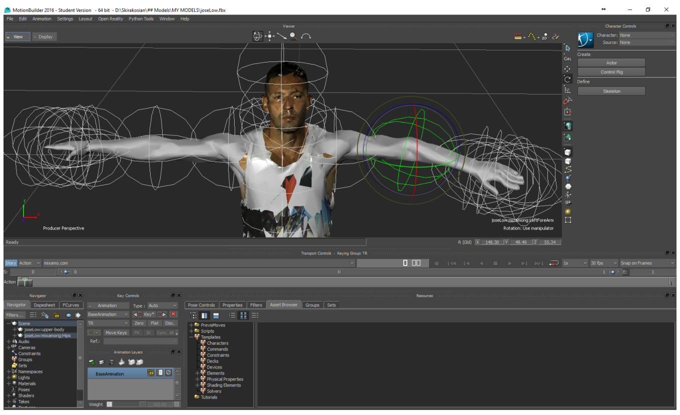


Figure 3.6: MotionBuilder

Chapter 4

User View

4.1 Game Play

VR Dance Lessons: The presented VR Training application will immerse the user in a Latin dance studio in VR. The Training application puts users in the heart of the basic knowledge of the Latin dance Salsa. The users guided to go through a series of tutorials provided by the current application and interact with the laser scanned 3D model representing the famous Spanish Latin dancer Alberto Rodriguez. After completing the interaction process, users can perform an interactive dance with this 3D model by leading the 3D character to execute the basic steps learned.

4.1.1 Main Scene

As users start the game, their profiles appear in the Main tutorial VR scene. Users see a UI dialog in front of them. This UI dialog prepares the users to take a brief pre-tutorial guide and to get familiar with the scene, gesture and UI interaction. This dialog offers them useful instructions about the game play. Users with more experience in VR games and generally in VR interaction, can skip this pre-tutorial procedure and go directly to Main Tutorial UI. See Figure (4.1).

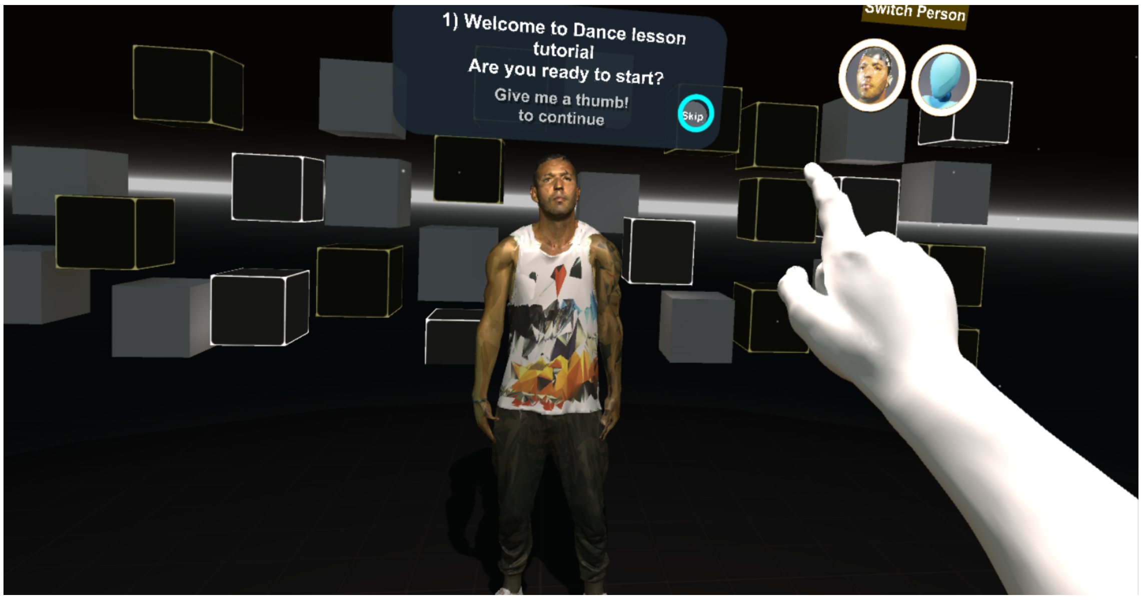


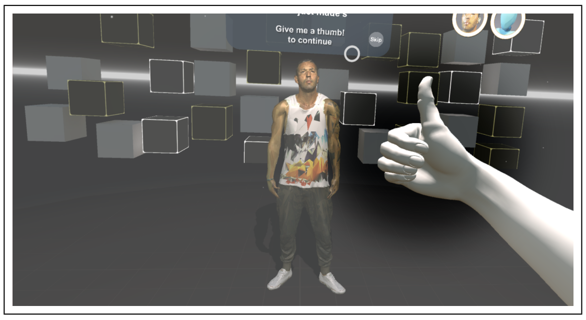
Figure 4.1: Main Scene

4.1.2 Gestures

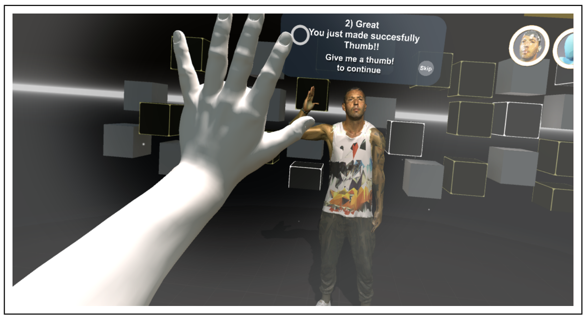
Gestures are very useful in our application and serve as an interaction and feedback mechanism between the users required to interact with the pre-tutorial process, and the dance training system. For this reason, feedback and interaction based on gestures, was implemented. **There are two types of Gestures in the Training Application:**

- **1:** Right Thumb Gesture (Figure 4.2a). Users form a thumb Gesture every time they are asked by the pre-tutorial guide panel to continue to the next dialog.

- **2:** "Wave Hand" Gesture (Figure 4.2b). Users form a wave hand Gesture in order to interact with the 3D character that is in front of them. When the "Wave Hand" gesture is successfully activated, the 3D humanoid character responds with the same "Wave Hand" movement.



(a) Thumb Gesture



(b) Shake Hand

Figure 4.2: Gestures

4.1.3 Tutorial UI

During the Main UI Tutorial process, users are presented with three types of tutorial stages that they can select, see (4.4). The first option is "Basic Steps". When the"Basic Steps" stage is selected, there is an extra UI panel that appears in the scene. While interacting with this UI panel, users can select and learn three types of basic steps ("Basic A", "Side Steps", "Right Turn") of Latin dance, see figure (4.5). Next option is "Interaction", which is located at the Main UI, see (4.4). During the "Interaction" stage, users interact with the 3D character by learning to lead the 3D character following the basic steps. In order to achieve the "Side Step" leading of the 3D character, users have to move their left hand to the left, or execute the basic "Side Step" to the left transferring their axis to the left side. The 3D character can recognize the users hand movement to the left and only then the 3D character executes the "Side Step" animation. For the execution of the "Turn" animation, users must rise their left hand up. The 3D character recognizes the elevation of the user's left hand and executes the basic right "Turn" animation. The last option is a "Dance" where the music is playing and users try to improvise and do some basic steps leading the 3D character to a series of basic step combinations. In order to complete all leading operations, the user must be in the "Interaction Zone", see Figure (5.21).

Figure 4.3: Return button

The users can select "Return" whenever they want to return from the "Basic Steps" to the "Main Tutorial" UI panel. The selection is achieved with the pinching action, see Figure (4.3).

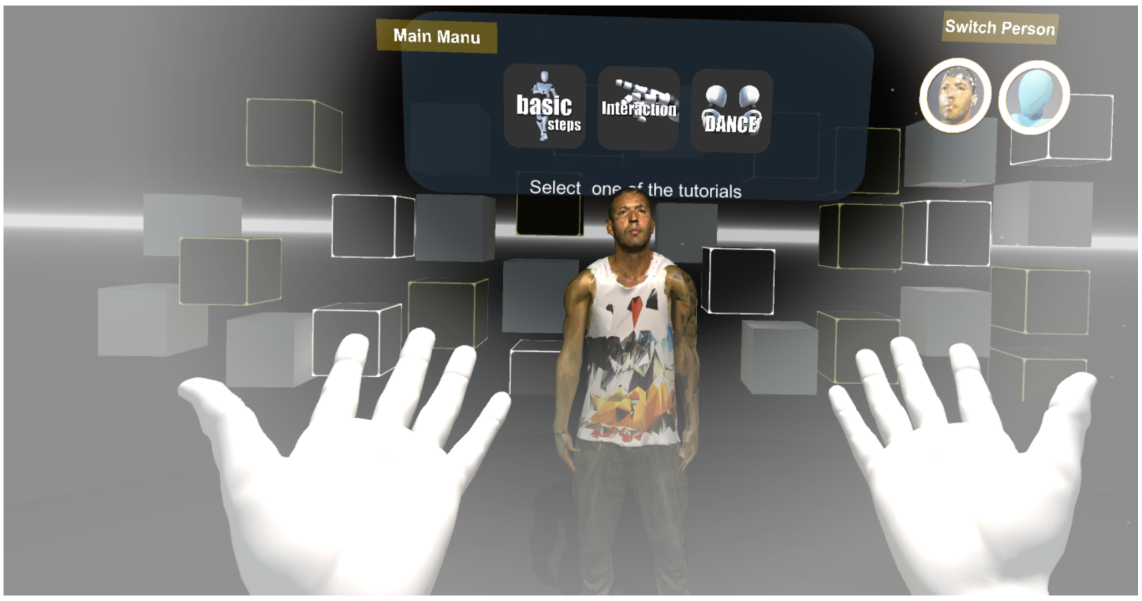


Figure 4.4: Tutorial UI

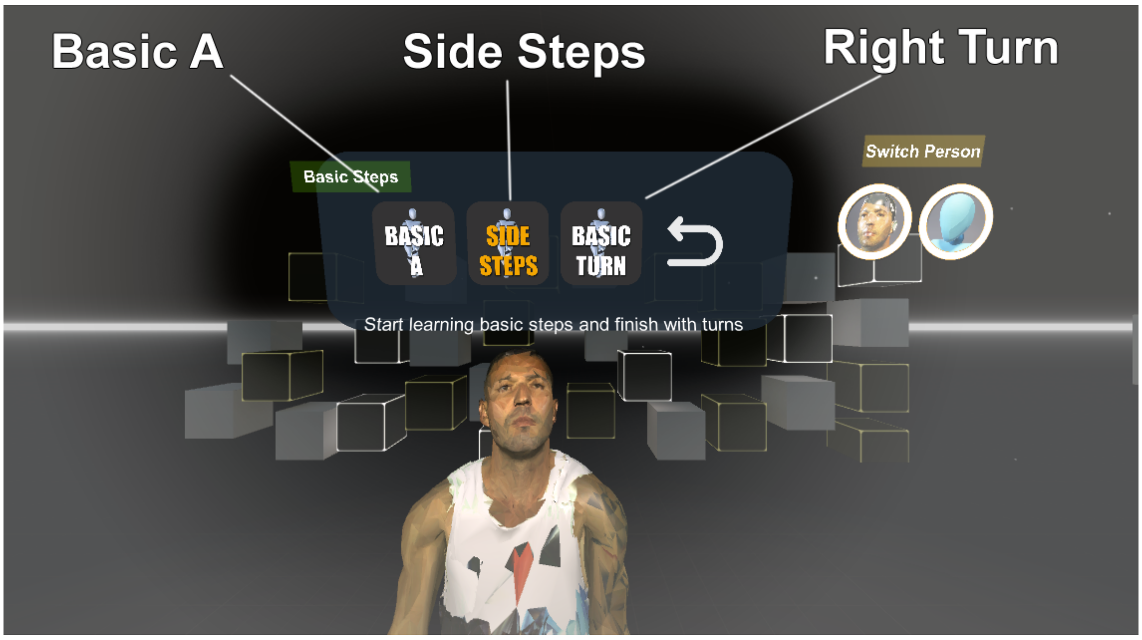


Figure 4.5: Basic Steps

4.1.4 Switch Character

When users execute the tutorial lessons, they can switch between a male human realistic 3D character and a female humanoid robot. In order to switch 3D characters, users must pinch the switch icons that are on the left side of the Main UI Tutorial Panel as shown in a Figure (4.6). Users can switch the character throughout the game play. After the switching action is registered, the selected 3D character is initialized to the default position under the UI Tutorial panel.

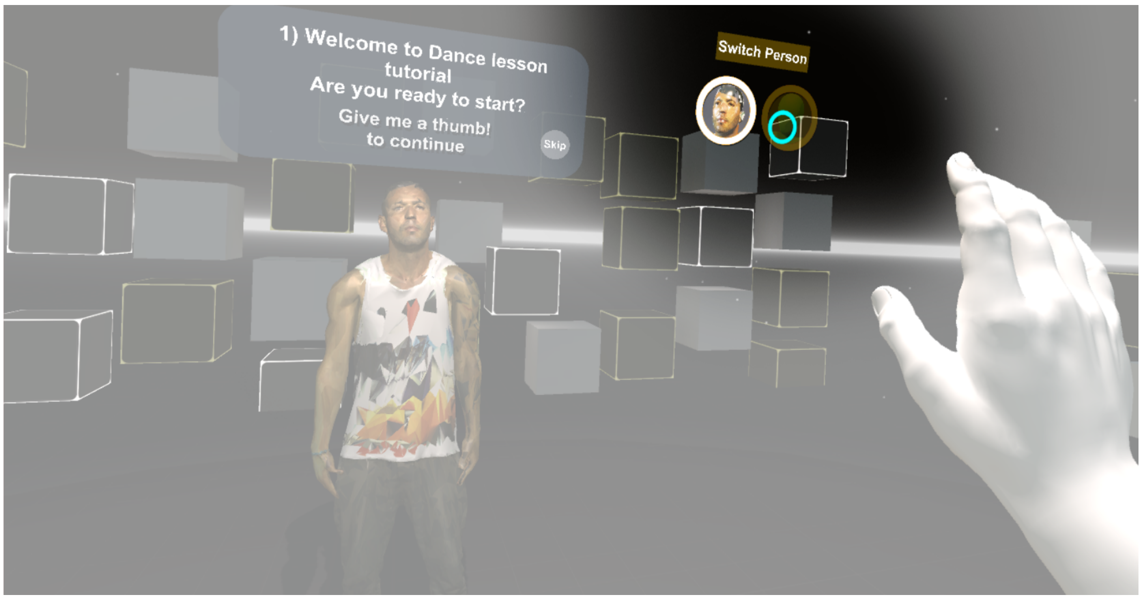


Figure 4.6: Character Switch UI

4.1.5 Settings UI

Main features on Setting UI are: (Figure 4.7)

- **1:** Adjust the volume of music: Users can adjust the volume of the current playing music through the UI setting. The adjustment can be conducted by sliding the green button on the slide bar of the UI Setting panel. The sliding can be achieved by touching the green button or by pinching it while the user is far away from the panel. Interacting with the UI setting panel, users can also Zoom in and Zoom out by touching the background of the panel.

- **2:** Mute the music: Users can also Mute the playing music by pressing the toggle button on the UI Setting panel. The selection of this event can materialize with the touch and pinch action.

- **3:** Restart the Game: Users can restart the game by pressing or pinching the "Restart button" on the UI Setting panel. The training application starts from the beginning, allowing Users to skip this pre-Tutorial procedure.

```
public void hardRestartGame()
    {
     SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
```

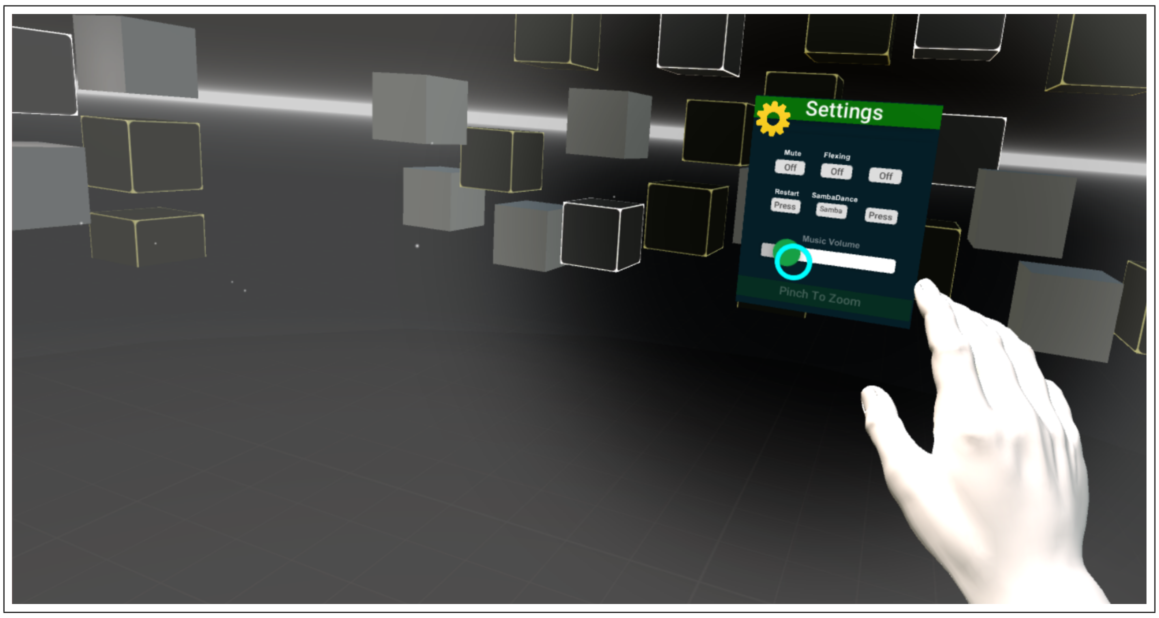


Figure 4.7: Settings UI

4.2 3D Virtual Scene

This section describes the implementation of the current VR Scene (User View) of the VR Interactive Dance Lesson application.

4.2.1 Character Modeling

Character modeling is developed with industry standard 3D modeling software called 3DsMax based on a student license (see Figure 3.5). The first attempt to create a 3d Character was a male humanoid 3d Model (see Figure 3.5). For the purpose of this project we tried to slice the character into separate objects (hands, body, legs, head...) and connect these objects with joints through Unity. This attempt was a complete failure because no realistic humanoid interaction could be applied to this sliced model while the complexity of controlling the realistic animation was very high.

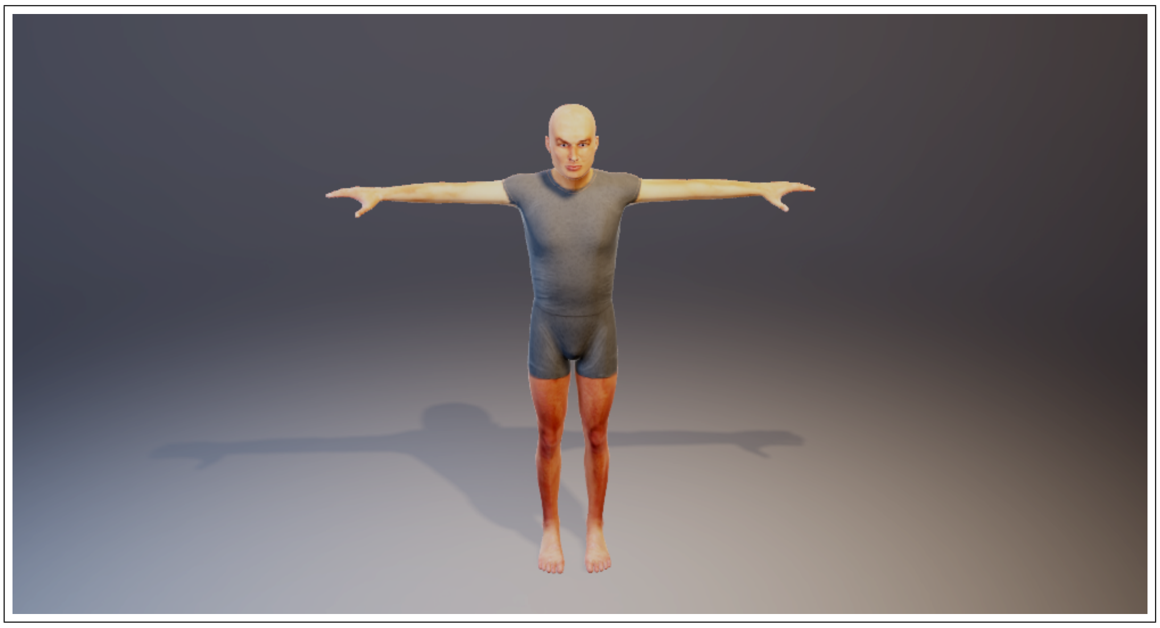


Figure 4.8: First Attempt

To achieve a humanoid realistic interaction for this project we used an existed robot model where we applied skeleton rigging using cross platform 3d modeling software 3DSMAX and MotionBuilder so that sliced hierarchy was created applying physics and interactive animations.

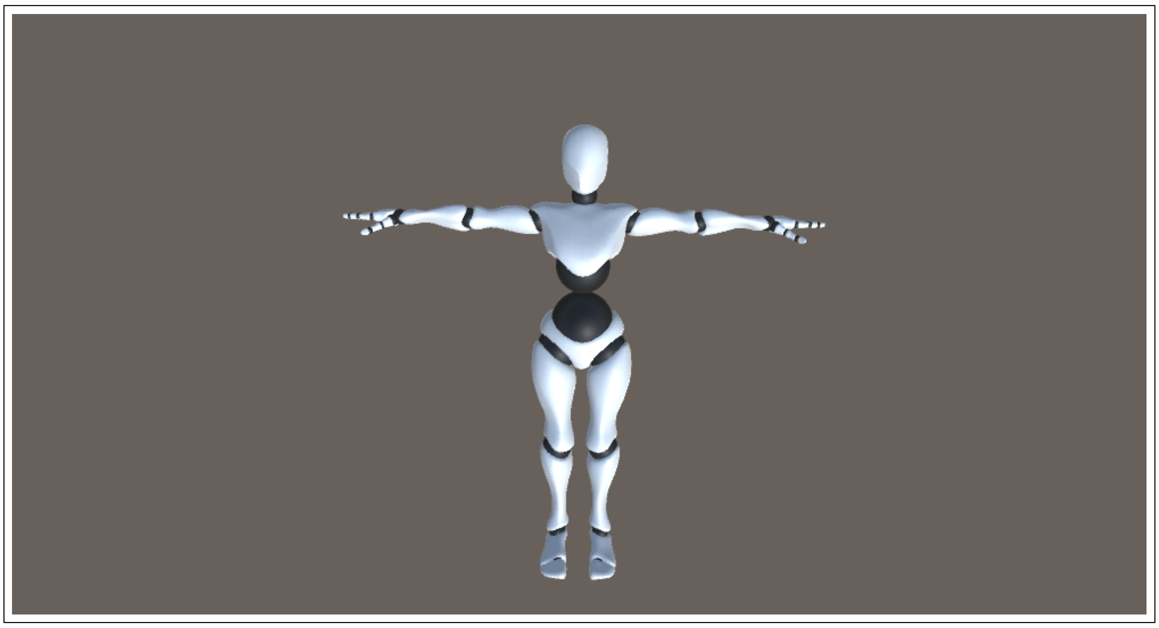


Figure 4.9: Final

4.2.1.1 White light Scanning

To create a more realistic humanoid 3d character in this project we collaborated with Technological Educational Institute of Crete - School of Applied Sciences with Manolis Maravelakis (see Figure 4.10a). In addition, we invited the professional Spanish Latin dancer Alberto Rodriguez(see Figure 4.10b) who was used as a 3d Model in our project.

(a) Manolis Maravelakis

(b) Alberto Rordriguez

Figure 4.10: Collaborators

For the scanning process, the latest lase scanning technology was used collaborating with the experienced team led by Prof. Manolis Maravelakis in their laboratory located at the Technological Educational Institute of Crete. The White Light scanning equipment that was used is manufactured by the company Artec3D product model Artec Eva (see Figure 4.12).

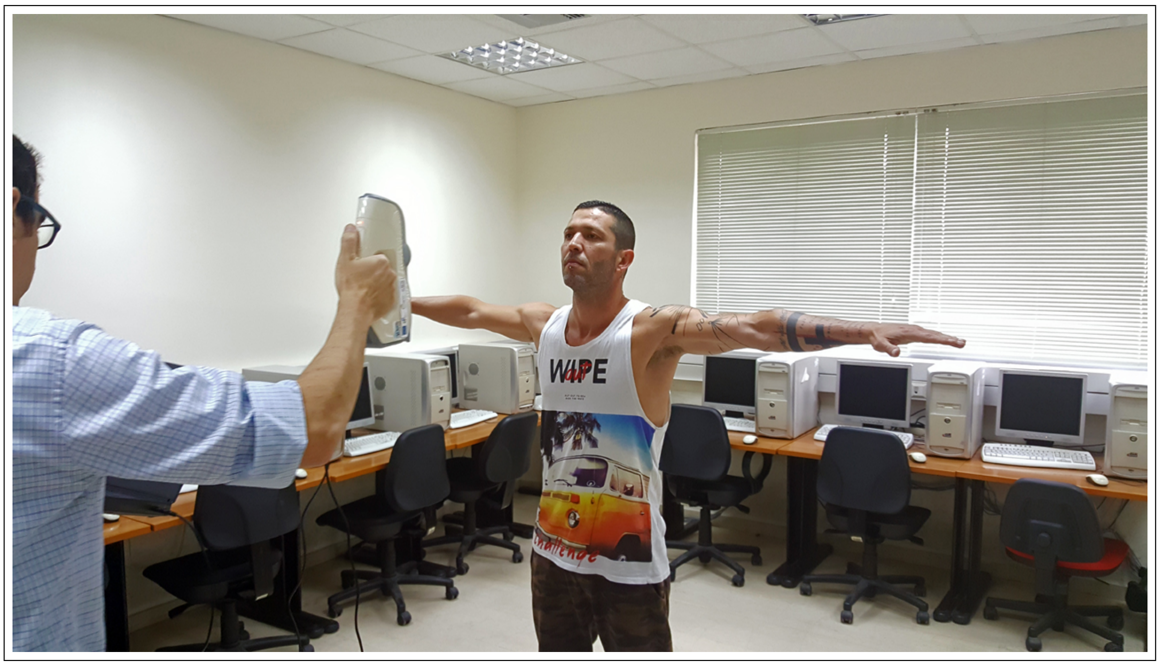


Figure 4.11: Process of white light scanning

Figure 4.12: Artec Eva white light scanning machine

The method that we followed for scanning a model so that the final 3d humanoid model was produced, consisted of four steps (Figure 4.13):

- **1: Scanning a physical person:** The physical model was standing on a T pose while the white light scanning machine operator Prof. Michalis Maravelakis was moving around the physical model scanning all parts of the body.
- **2: Repair 3D model using 3D modeling software:** Because of the faulty scan of some parts of a scanned body, a separate scanning of specific parts was

needed to repair the initially failed scanned parts. After the separate scanning was completed, a manual repair and connection of these parts was needed. The 3D modeling platform 3DSmax was used to complete this task.

- **3: Optimize using 3D modeling software:** After the white light scanning was completed and all parts of the body were connected, the 3D model included more than 1 million vertices. Through 3DSmax, we reduced the resolution (amount of vertices) by 90% lower than the original scanned model, trying to keep details of a face so that the user could intuitively perceive that the 3D character represents a realistic human model.

- **4: Biped Rigging using 3DSmax:** Finally, we added bones to the skin of our scanned 3D model so that animations could be applied. This procedure is described in the following chapters.

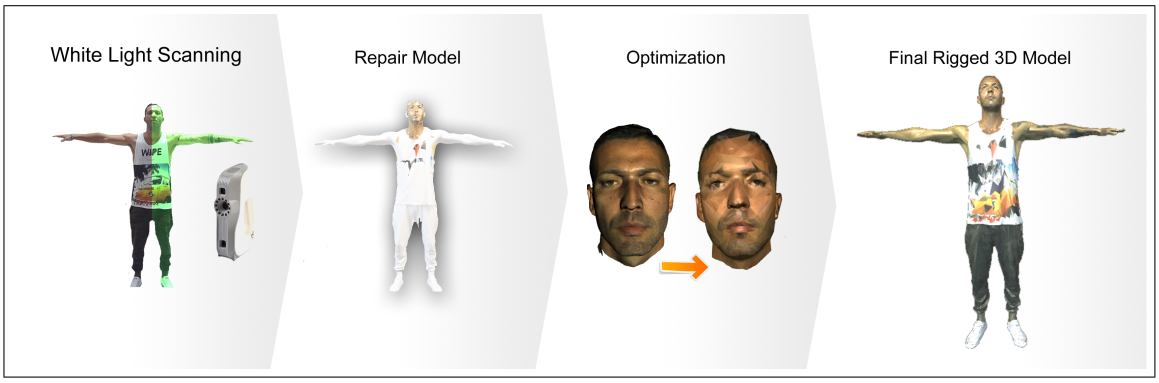


Figure 4.13: Procedure for white light scanning

4.2.2 Environment modeling

The 3D environment in this project adopted a minimal design style by simulating the default scene of the Occulus Rift. The main purpose of the minimal style was to have minimum geometrical complexity and computational cost of the 3D scene rendering, so that the power of the graphics card (GPU) could be applied while the 3D character is rendered.

Baked GI - Traditionally, video games and other real-time graphics applications have been limited to direct lighting, while the calculations required for indirect lighting were too slow so they could only be used in non-real-time situations such as CG animated films. A way for games to work around this limitation is to calculate indirect light only for objects and surfaces that are known ahead of time to not move around (that are static). That way the slow computation can be done ahead of time, but since the objects don't move, the indirect light that is pre-calculated this way will still be correct at run-time. Unity supports this technique, called Baked GI (also known as Baked Lightmaps), which is named after "the bake" - the process in which the indirect light is precalculated and stored (baked). In addition to indirect light, Baked GI also takes advantage of the greater computation time available to generate more realistic soft shadows from area lights and indirect light than what can normally be achieved with real-time techniques.

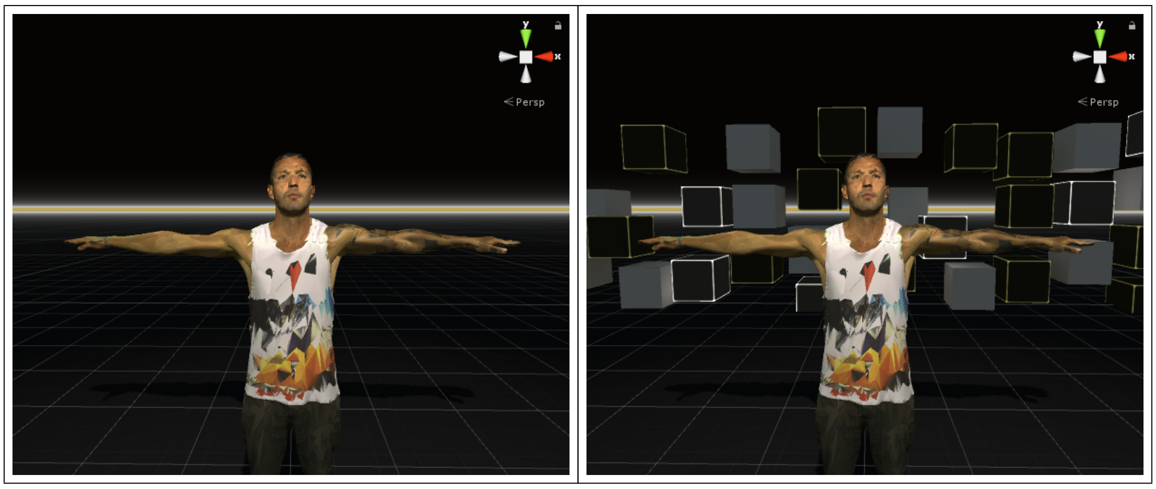
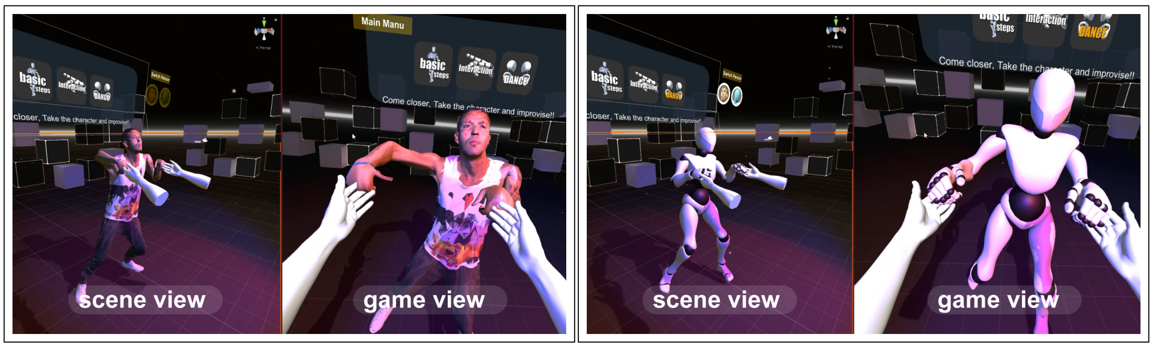

(a) No Decoration (b) with decoration

Figure 4.14: Minimal Environment

In relation to the environment where the dance action takes place, a set of lightened boxes was added to the scene. A special texture was created applied to the boxes with bright edges rendered on them. In order to optimize the shadow calculation from the main light source, we used baked light in order not to calculate every frame of the light path for the static objects. A real-time calculation of direction lighting was added to the scene only to affect the 3D character model and the user's 3D hands model (see Figure 4.14, 4.15).



(a) Jose dance view (b) Robot dance view

Figure 4.15: Game Play interaction scene

Chapter 5

Implementation

5.1 UI implementation

The dance tutorial will be conducted by inexperienced users. The development of a simple and intuitive User Interface (UI) was necessary. When developing the UI, several guidelines of HCI (Human Computer Interaction) such as the semantics of the colors and the ideal positioning of the interface elements, were taken into consideration. Each selection included in the UI canvas represents a touchable button that the user can touch through the 3d hand model visualized for interaction. For this project three different UI Panels were created:

- **1: UI Dialog:** While interacting with this UI, the user gets a brief introduction to the game. This procedure prepares and guides users through certain steps while user gestures and virtually touching buttons are used to interact with the UI canvas (see Figures 5.1,5.2). While the text is being displayed in the dialog UI, sound and speech connected with typing for each step is playing.

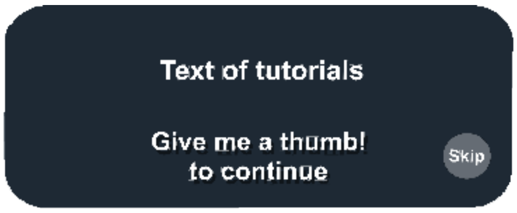


Figure 5.1: UI Dialog

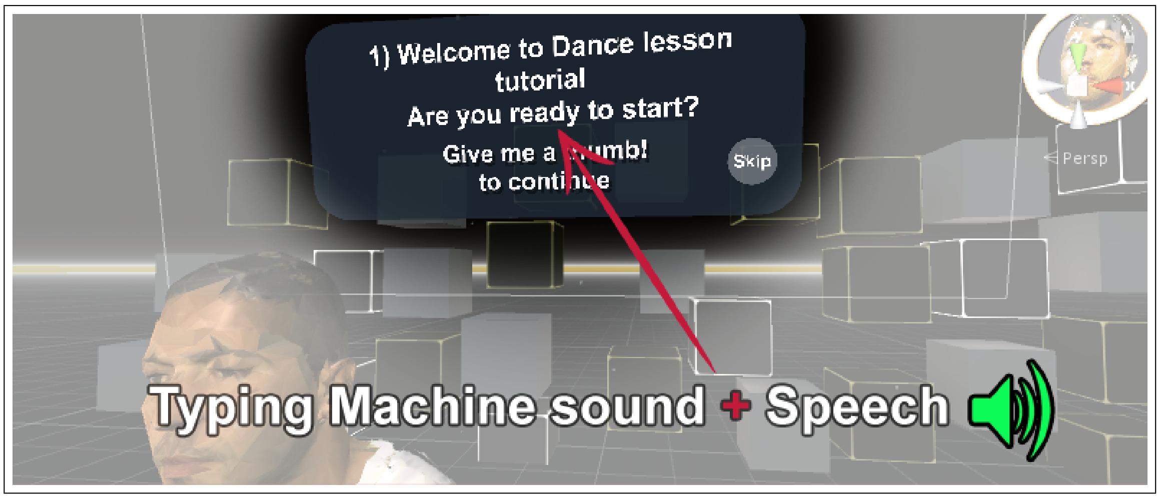
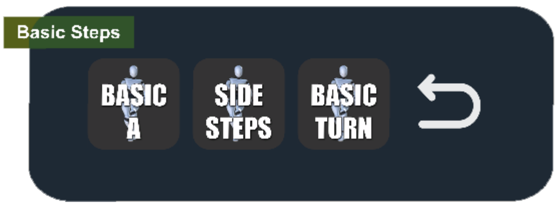
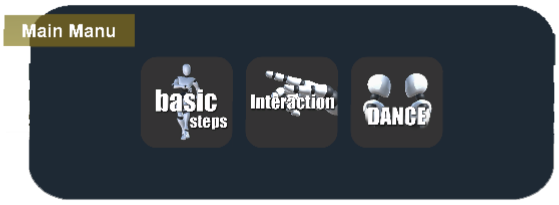

1) Welcome to Dance lesson tutorial
Are you ready to start?
Give me a thumb!
to continue
Skip
Persp
Typing Machine sound + Speech

Figure 5.2: UI Typing sound & Speech

- **2: UI Menu Tutorial:** This UI consist of 2 levels (Main Menu and Basic Steps). While interacting with the first level (see Figure 5.3a), users are able to select between three options (Basic Steps, Interaction, Dance). If the "Basic Steps" option is selected, then the second level of UI appears (see Figure 5.3b) and users could select between three additional options reflecting Salsa's basic steps ("Basic A", "Side Steps" and "Turn").



(a) Fist Level



(b) Second Level

Figure 5.3: Minimal Environment

- **3: UI Settings:** While interacting with this UI, the user controls certain basic settings such as "Game Restart" or adjust the Music volume that is located at the GameObject "_scripts_" (see Figure 5.4).

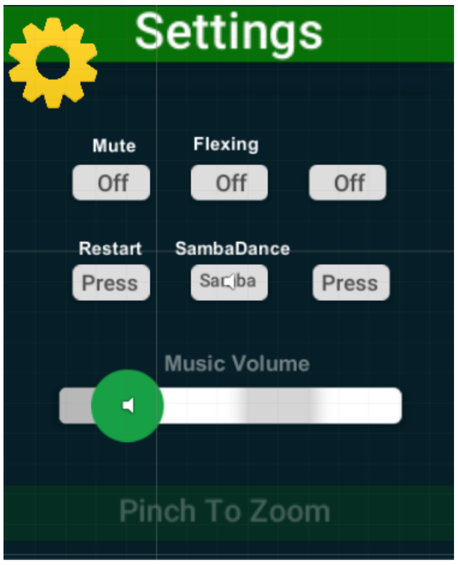


Figure 5.4: UI Dialog

In order to show and control globally the **UI Dialog** and **UI Menu Tutorial** a central UI Canvas GameObject was created, the position of which is initialized when the game starts. Every time the UI Dialog or the UI Menu Tutorial is enabled, this UI becomes a child of the Central UI Canvas, taking the parent's position.

In order to enhance the feeling of VR presence, a shadow and sound feedback on the UI buttons were added, to appear when a button is pressed utilizing the user's 3D hand model.

5.2 Animation

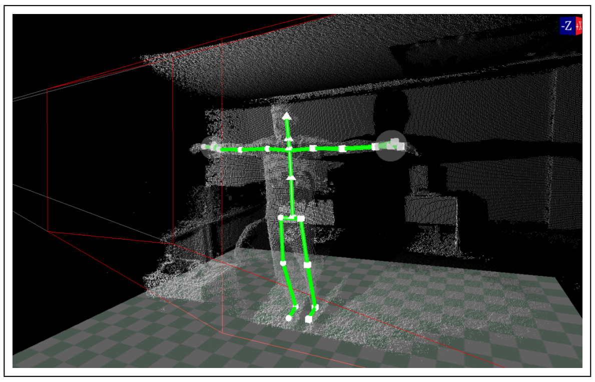
Moreover, the implementation of realistic animation movement was created. We will describe the process we followed to create certain basic realistic animations that were applied to the main 3D Character and UI GameObjects in the following chapters.

5.2.1 Character Animation

The main application scene is populated by two characters, the first is the user (users can see their hands which are represented by 3D hand models) and the second character is a 3D humanoid model that users interact with. Animations are applied to the 3D humanoid model. In order to animate a 3D humanoid character we needed a model that is rigged as we described in (Figure 4.13) step 4. For this project, we used two distinct ways to create animations for the character. The first was to take existed animations such as "Idle state" , "Flexing state" , "Walk state", "Turn state" where certain parameters of animations such as speed were edited employing Motion Builder platform and the Unity itself. The second way was to capture motions of a real human with a tracking device and then transfer this data onto the 3D humanoid model producing a realistic animation of movement.

5.2.1.1 Motion Capture

There are two ways to capture real human movement and transfer this data to the 3D humanoid rigged model. The first one is motion capture of real human movement. This data is not directly connected to the main 3D model but to the general skeleton while the record mode was enabled through the Motion Builder platform. After the recording is finished, we optimized the animations by smoothing the jitters. Finally, the animation was ready to be applied to the humanoid rigged 3D characters in Unity, see (Figure 5.5a).


(a) Motion Capture with Kinect v2

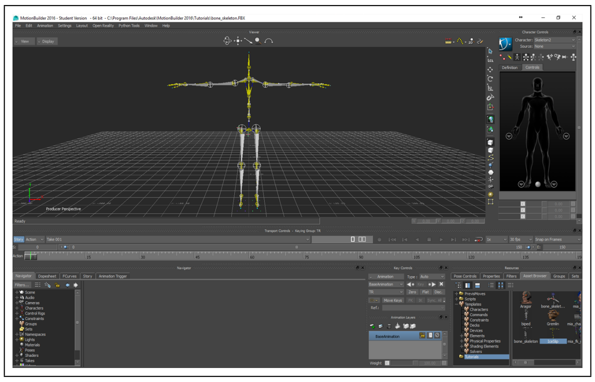

(b) Edit Animation with Motion Builder

Figure 5.5: Mocap

The second way was to capture the motion and transfer data to the main scene in real time. This technique was applied for the real-time tracking of the users hands. For this purpose, a motion tracking device named Leap Motion was needed, see (Figure 5.6). No tracking data is stored in the application. Any transformation that happens to the physical hands is reflected to the transformation of the user's 3D hand models. This type of motion capture was needed so that the user interacts with the UIs and characters while immersed in the main scene.

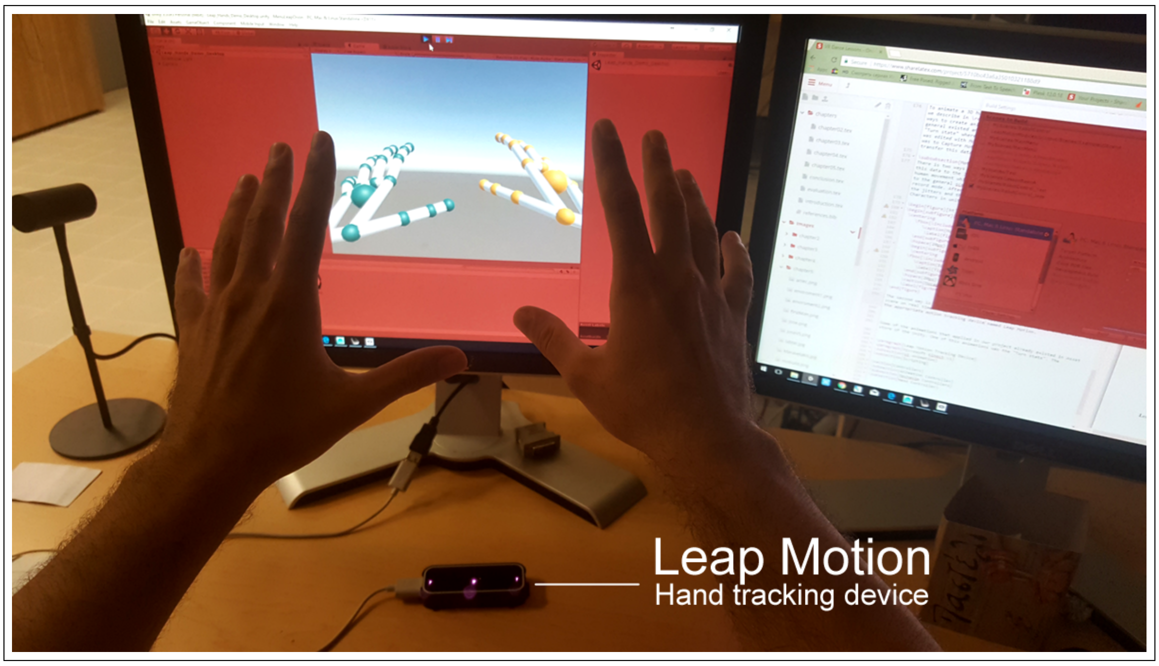


Figure 5.6: Leap Motion tracking device

Some of the animations that were applied in our project to the 3D humanoid character, already existed in Unity's Asset store. One of these animations was the "Turn state". The "Turn state" was impossible to be captured utilizing the available equipment in our lab, by only one Kinect sensor, see (Figure 5.8a). The problem was that the Kinect device captures only a projection of the view, see (Figure 5.7).

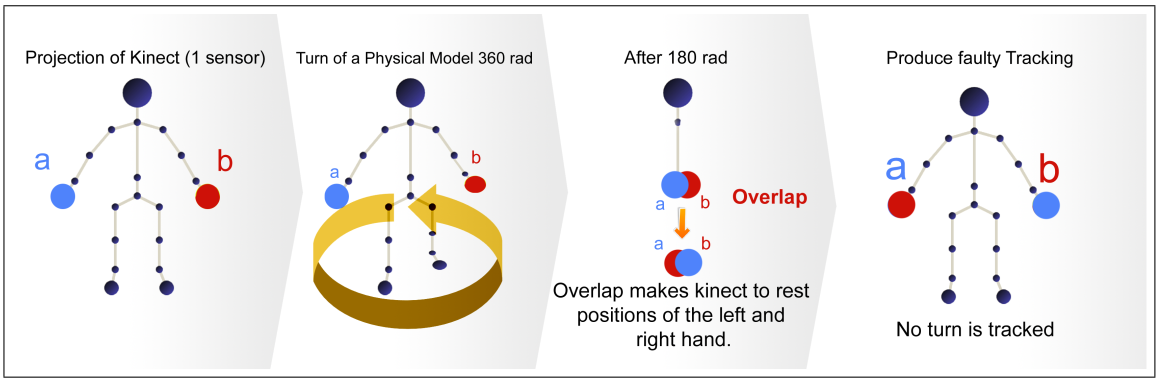
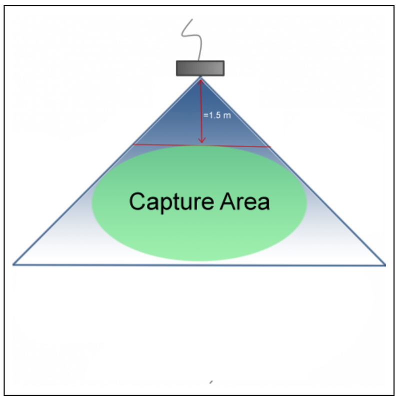
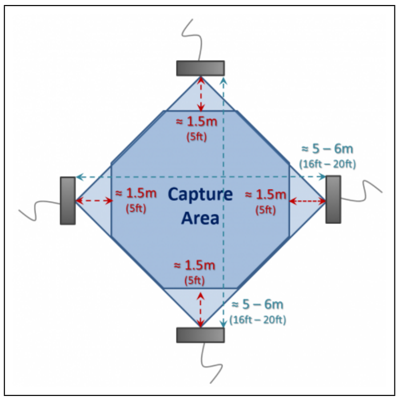

Projection of Kinect (1 sensor)
a
b
Turn of a Physical Model 360 rad
a
b
After 180 rad
a
b
Overlap
a
b
Overlap makes kinect to rest positions of the left and right hand.
Produce faulty Tracking
a
b
No turn is tracked

Figure 5.7: Error on 360 turn with Kinect with 1 sensor

Therefore, in order to capture the 360 rad motion, we must use more than one sensor. To achieve the best motion capture, a four sensor system setup that is shown in the Figure (5.8b), is suggested. In the current project, only one sensor was used.



(a) Single sensor Setup



(b) Multiple sensor setup

Figure 5.8: Kinect Topology

5.2.2 Ui Animation

In order to create additional visual effects visualized onto the Settings Menu UI, (see Figure 4.7) we added an animation to canvas. Whenever the user wants to make a selection or view buttons from a close position, the UI menu is moving towards the user on demand by zoom-in and zoom-out, see (Figure 5.10).

In order create the UI animation, we used a different technique compared to the 3D humanoid character animation we described above. The method we used was the creation of animation through Unity itself, using Animation View windows. The **Animation view** is linked to the Hierarchy view, the Project window, the Scene View, and the Inspector window. Like the Inspector, the Animation View shows the time-line and key frames of the Animation for the currently selected GameObject or Animation Clip Asset.

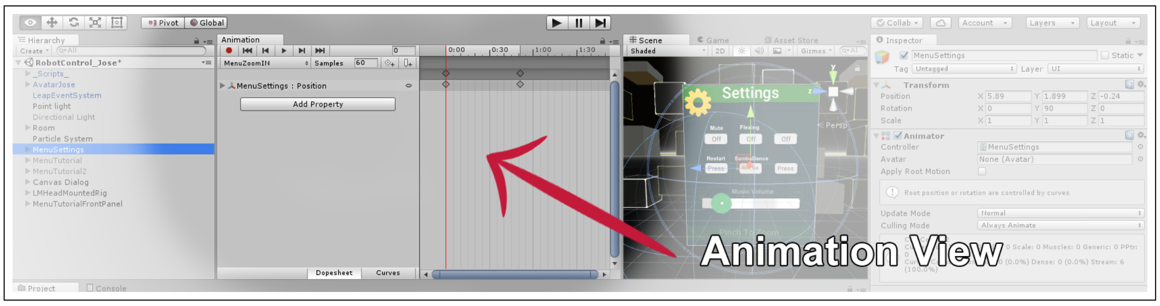


Figure 5.9: Animation View

The implementation controlling the UI Animations and 3D character animations will be described in the next section.

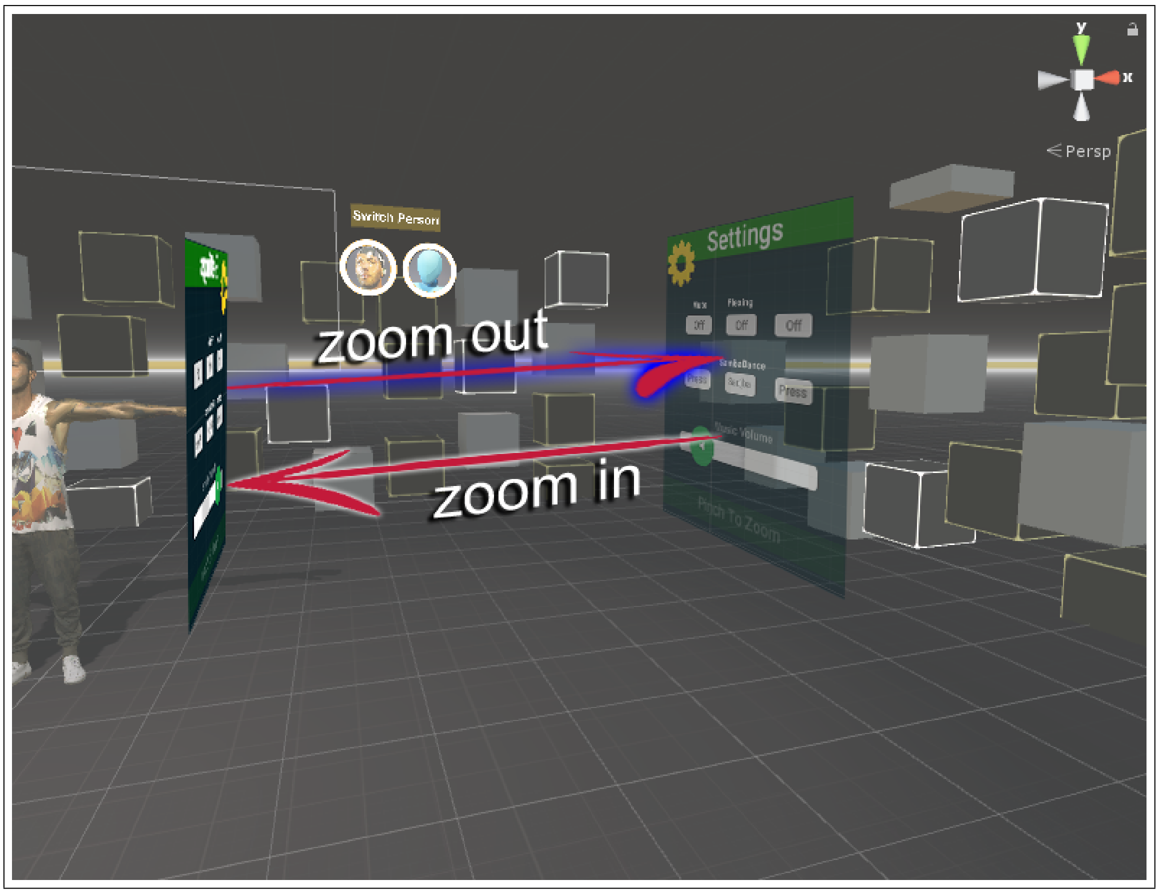


Figure 5.10: UI Setting Menu Animation

5.3 Controllers

In order to control the multiple animations included in this project, we used Animator controllers combined with scripting in C#. **Animator Controller** allows you to arrange and maintain a set of animations for a character or other animated GameObjects.

The controller has references to the animation clips used within it, and manages the various animation states and the transitions between them using a so-called **State Machine**, which could be thought of as a kind of flow-chart, or a simple program written in a visual programming language within Unity.

Animation State Machines. It is common for a character or other animated GameObject to have several different animations that correspond to different actions it can perform in the game. For example, a character may breathe or sway slightly while idle, walk when commanded to and raise its arms in panic as it falls from a platform. A door may have animations for opening, closing, getting jammed, and being broken open. Mecanim uses a visual layout system similar to a flow-chart, to represent a state machine to enable you to control and sequence the animation clips that you want to use on your character or object. This section gives further details about state machines we used for the animations in the current application.

5.3.1 3D Character Animation Controller

For this project we created a dedicated control of animations for a 3D character using the Animator Controller with a State Machine that can apply to any Humanoid character in the scene.

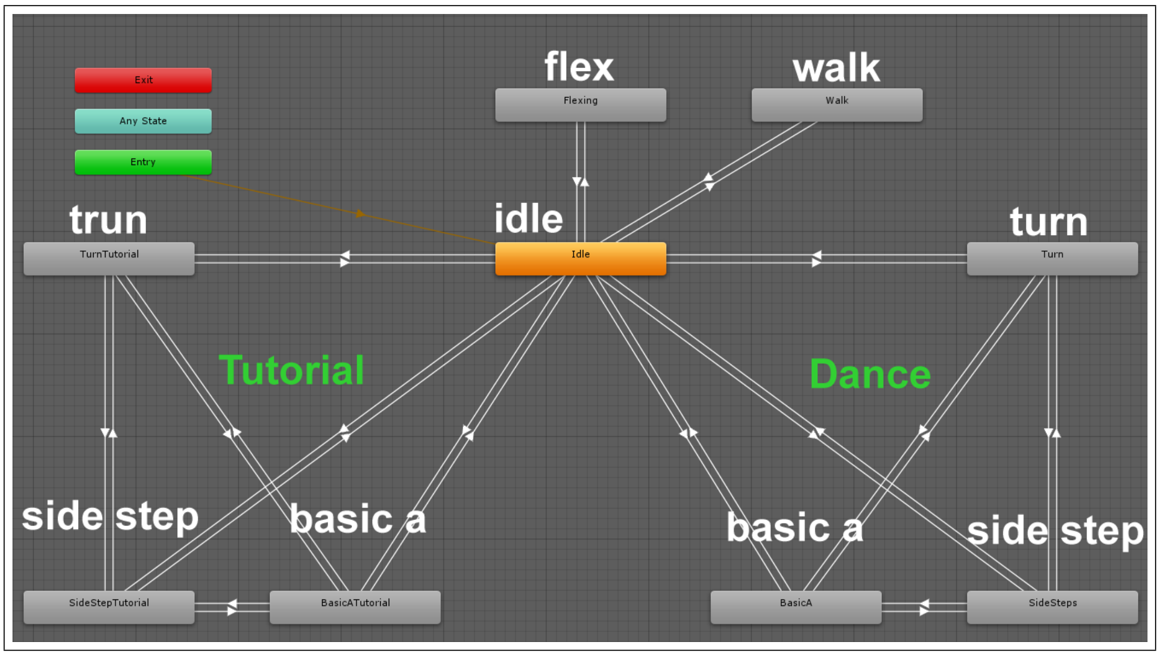
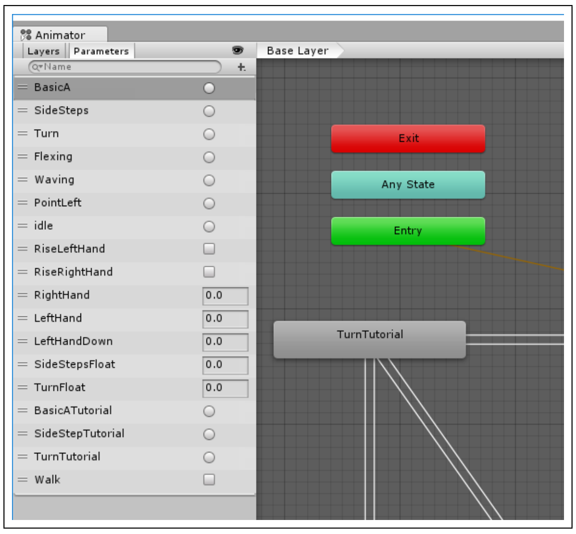


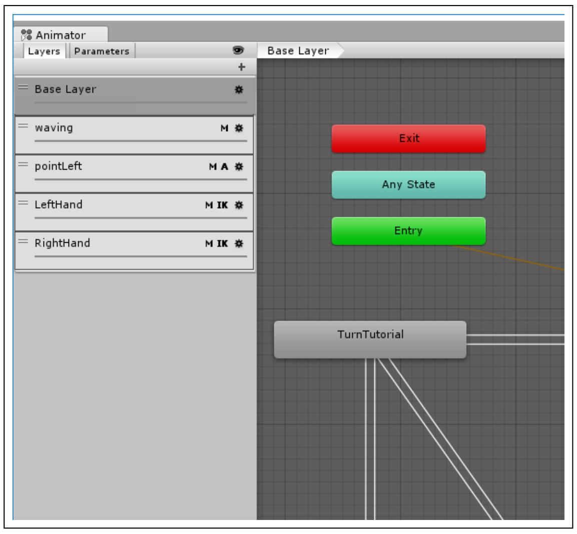
Figure 5.11: 3D animation State Machine Base Layer

Animation Parameters are variables that are defined within an Animator Controller that we access and assign values from scripts (see Figure 5.12a). Some of the functions that access those parameters will be described in the Scripting section.

Animation Layers. Unity uses Animation Layers for managing complex state machines for different body parts of a 3D character. For example, in this project, we have a lower-body layer for idle state, and an upper-body layer for the waving hand state. This technique allows the 3D character to wave a hand while in the idle mode. We can manage animation layers by interacting with the Layers Widget in the top-left corner of the Animator Controller see Figure 5.12b.



(a) Parameters

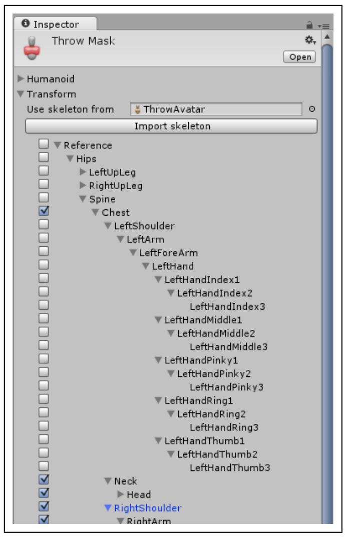


(b) Layers

Figure 5.12: Animator Controller

In order to create realistic animation transitions we used Avatar Mask to separate animations for the different parts of 3D character body.

Avatar Mask allows you to discard some of the animation data within a clip, allowing the clip to animate only parts of the object or character, rather than the entire thing. For example, in this project, the 3D character may have a standard idle animation that includes both arm and leg motion, but if a character is waving a hand then we would't want the arms to use an idle animation movement. However, we could still use the standard idle animation while waving hand by using a mask to only animate the upper body part (only right hand) of the "waving" animation over the top of the "idle" animation.



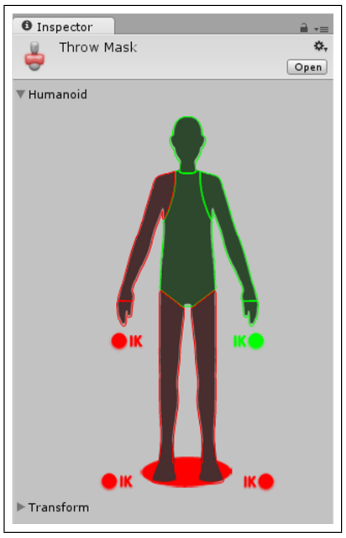
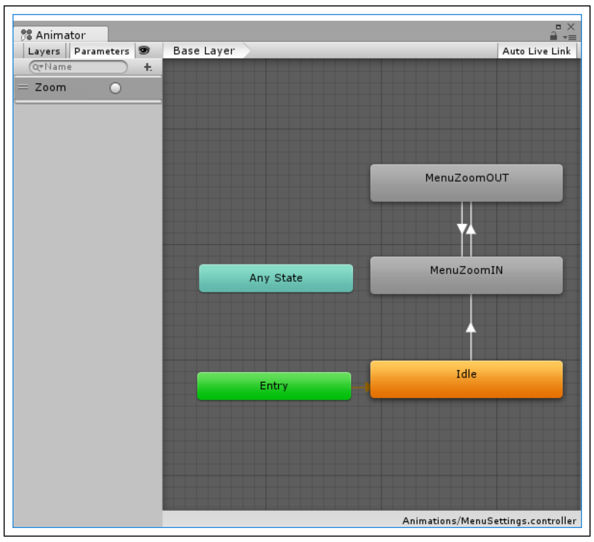


Figure 5.13: Mask Parameters

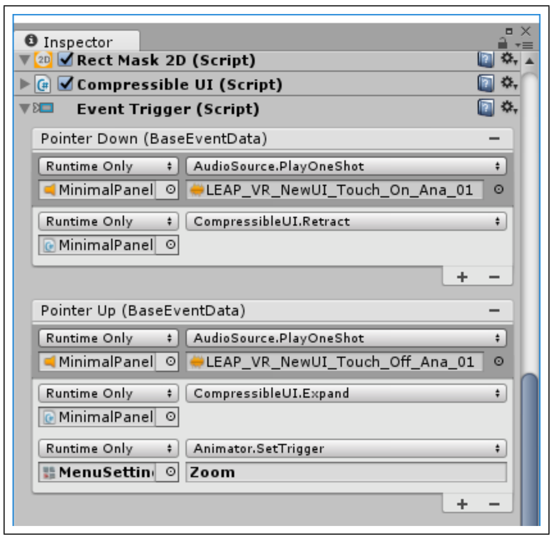
5.3.2 UI Animation Controller

For the UI animation we used a simplified State Machine, controlled in the Animator Controller. We used only three states with one parameter ”zoom” type trigger (see Figure 5.14).

Every time the user touches the UI Settings menu Canvas, a trigger parameter ”Zoom” is being activated and a state is changing. The UI Setting Canvas starts to animate every time the ”Zoom” trigger parameter is activated. The animation translate the UI forward and backward on the X axis of the scene.



(a) Parameters



Inspector
Rect Mask 2D (Script)
Compressible UI (Script)
Event Trigger (Script)

Pointer Down (BaseEventData)

Runtime Only	AudioSource.PlayOneShot
MinimalPanel	LEAP_VR_NewUI_Touch_On_Ana_01
Runtime Only	CompressibleUI.Retract
MinimalPanel	

Pointer Up (BaseEventData)

Runtime Only	AudioSource.PlayOneShot
MinimalPanel	LEAP_VR_NewUI_Touch_Off_Ana_01
Runtime Only	CompressibleUI.Expand
MinimalPanel	
Runtime Only	Animator.SetTrigger
MenuSettin	Zoom

(b) UI Canvas Button Event

Figure 5.14: UI Setting Mecanim

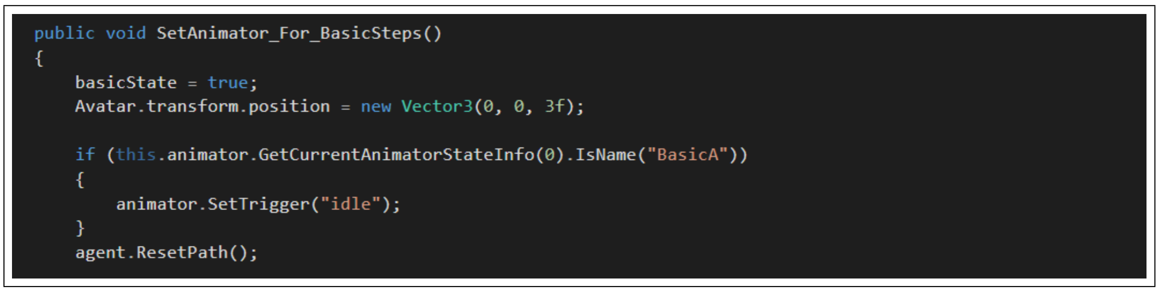
5.3.3 Script Handling

In our project we created a GameObject named _**Scripts**_ that controls some of the main GameObjects in our scene. There are five main scripts included as a components in this _Scripts_ GameObject which controls the game play of the VR Dance:

- **Tutorial Stage Script**: This Script controls the Dialog panels at the beginning of the VR Game. It is in charge of show/hide the main UI of the game. Also we added a function that allows to simulate a machine typing sound when characters (text) appear in the Dialog UI.

- **Game Control Script**: This script is used for loading game stages and reload game on user demand.

- **Character Mechanics Script**: This script controls the parameters of the Animator Controller of the humanoid 3D Character we want to animate, see example (5.15). We also use a simple AI in this script that we will describe on our AI Section.

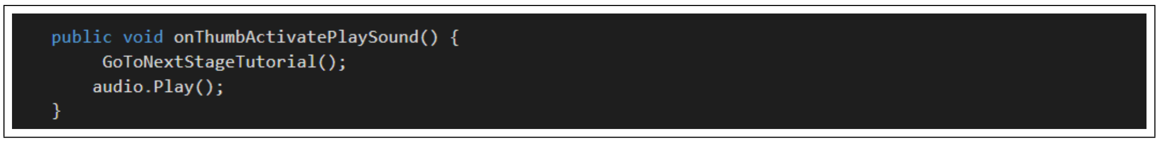


```
public void SetAnimator_For_BasicSteps()
{
    basicState = true;
    Avatar.transform.position = new Vector3(0, 0, 3f);

    if (this.animator.GetCurrentAnimatorStateInfo(0).IsName("BasicA"))
    {
        animator.SetTrigger("idle");
    }
    agent.ResetPath();
```

Figure 5.15: Accessing the idle trigger parameter in Animator controller

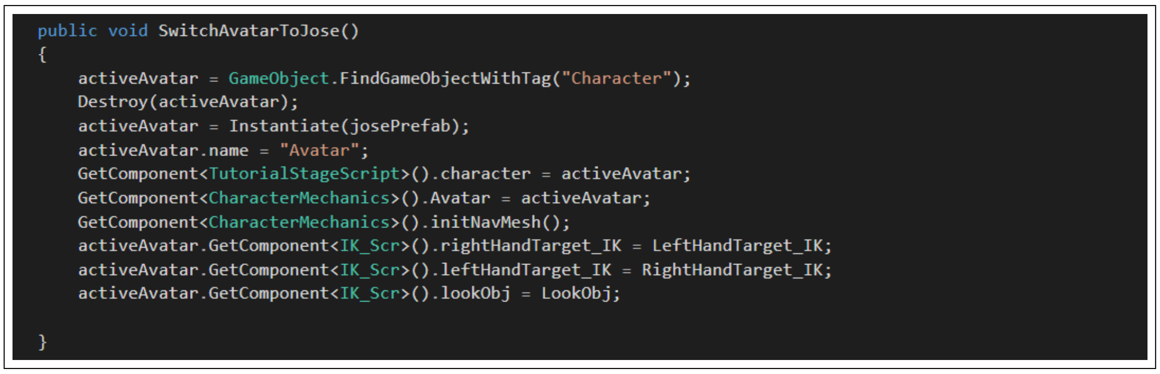
- **My Gesture Detection Script**: This script controls the available gestures. After a gesture activation, the script produces a sound and goes to the next stage, see example (5.16).



```
public void onThumbActivatePlaySound() {
     GoToNextStageTutorial();
    audio.Play();
}
```

Figure 5.16: Play Sound when Gesture "Thumb up" is activated

- **Character Switch Controller** This is the script that we use for switching 3D characters between "Jose" and "Beat" while we are in the training application on play mode. Here is a sample of a code that switches (initialize the main Avatar) to "Jose", see example (5.17).



```
public void SwitchAvatarToJose()
{
    activeAvatar = GameObject.FindGameObjectWithTag("Character");
    Destroy(activeAvatar);
    activeAvatar = Instantiate(josePrefab);
    activeAvatar.name = "Avatar";
    GetComponent<TutorialStageScript>().character = activeAvatar;
    GetComponent<CharacterMechanics>().Avatar = activeAvatar;
    GetComponent<CharacterMechanics>().initNavMesh();
    activeAvatar.GetComponent<IK_Scr>().rightHandTarget_IK = LeftHandTarget_IK;
    activeAvatar.GetComponent<IK_Scr>().leftHandTarget_IK = RightHandTarget_IK;
    activeAvatar.GetComponent<IK_Scr>().lookObj = LookObj;

}
```

Figure 5.17: Switch Avatar to "Jose"

5.4 Interaction

The Interaction Engine is a layer that exists between the Unity game engine and real-world hand physics. In order to make object interactions work in a way that satisfies human expectations, it implements an alternate set of physics rules that take over when hands are embedded inside a virtual object.

The Interaction Engine endeavours to make the integration quick and easy. However, it also allows a high degree of customization across a wide range of features. The developer can modify the properties of an object interaction, including desired position when grasped, moving the object to the desired position, determining what happens when tracking is momentarily lost, throwing velocity, and layer transitions to handle how collisions work.

In our project, we use touches and gestures to interact with the UI GameObjects as described in the "UI Implementation" section.

5.4.1 Gestures

Interaction occurs through gestures. The Leap Motion software recognizes certain movement patterns as gestures which could indicate a user intent or command. The Leap Motion software reports gestures observed in a frame in the same way that it reports other motion tracking data such as fingers and hands.

5.4.2 Hand Model

We used a pair of humanoid realistic shape hands coloured grey color so that they will be neutral for all users. In order to enhance immersion, it was decided that a universal colored hand is preferable rather than a specific color because the real color of their hands may be different from the VR. For example, if we have a real person with hairy hands in physical space, but in VR, we may not be able to represent that.

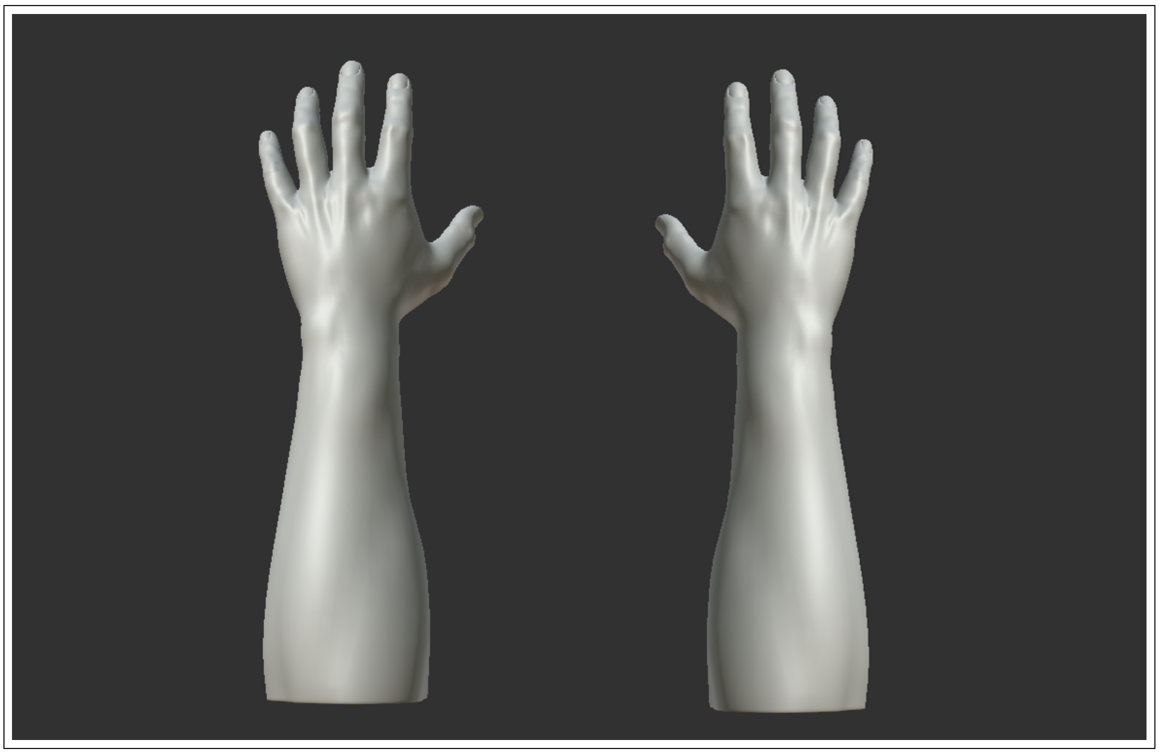


Figure 5.18: Grey color Hand of a User

5.4.3 Gesture Logic

In order to activate gesture,s we created a GameObject named "Thumb Up" and "Waving Hands" and we added as a component 3 Leap Motion Detection Utilities Scripts, see example (5.19). :

- **Finger Detection Detector** Here, we declare the fingers direction of the user's hand model we want to detect. In our project, we look for a detection event for 1 finger of a right hand of the user, specifically for the Thumb Gesture. So if the user extends his right thumb finger we have an Activation event of this script.

- **Extended Finger Detector** Here, we declare which fingers of the user's hand model we want to detect. In our project, we look at a detection event for the thumb finger of the right hand of the user. So if a user extends the thumb finger of the right hand upwards relative to the camera, then an event activation is registered.

- **Detector Logic Gate** In this Component, we assign in the Detector parameters the Finger Detection Detector Script and Extended Finger Detector Script with an AND Gate. So if we have the right hand thumb extended and the direction is UP, we have activation of the ThumbsUP Gesture. The same logic we use for the WavingHands Gesture.

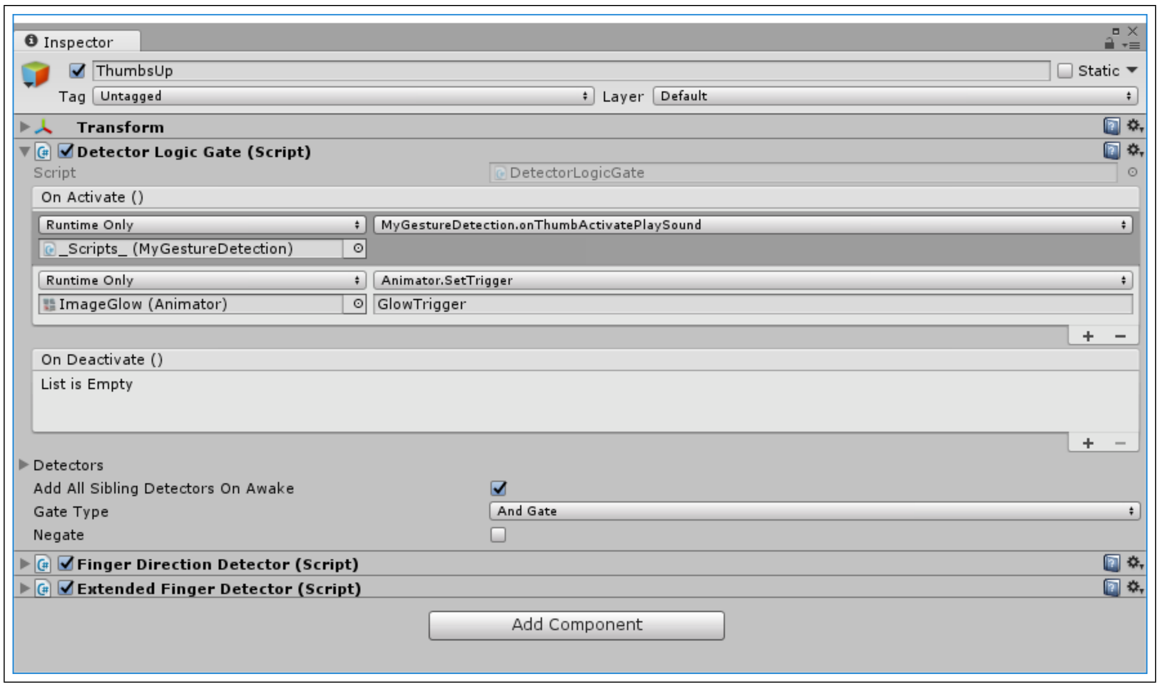

Inspector
ThumbsUp   Static
Tag Untagged   Layer Default
Transform
Detector Logic Gate (Script)
Script   DetectorLogicGate
On Activate ()
Runtime Only   MyGestureDetection.onThumbActivatePlaySound
Scripts (MyGestureDetection)
Runtime Only   Animator.SetTrigger
ImageGlow (Animator)   GlowTrigger
On Deactivate ()
List is Empty
Detectors
Add All Sibling Detectors On Awake   ✓
Gate Type   And Gate
Negate   ☐
Finger Direction Detector (Script)
Extended Finger Detector (Script)
Add Component

Figure 5.19: Logic of Activating Thumb Gesture

5.5 Inverse Kinematics (IK)

In order to achieve the interaction with a humanoid 3D character in our project, we used the Inversed Kinematics technique. Before we explain this, we need to mention the differences between Forward and Inverse Kinematics, e.g. being similar to a function and its inverse. In robotics for example, this normally refers to calculating the relations between end-effectors and joint angles. So for Forward Kinematics, the joint angles are the inputs, the outputs would be the coordinates of the end-effectors. On the other hand, for Inverse Kinematics, the given inputs are the coordinates of the end-effectors, however, the outputs to calculate are the joint angles. For a multiple DOF robot, the Forward Kinematics is quite straightforward. But inverse kinematics could be tricky in relation to the same end-effectors coordinates, lacking a unique configuration, especially when the system is redundant.

For this project, we created Inversed Kinematics for the 3D humanoid characters to control mainly hands motions interactions. In order to enhance the feeling of presence and real interaction, we added Inversed Kinematics on the 3D character so that the 3D character looks always at the user.

5.5.1 Implementation

In order to implement the Inversed Kinematics (IK), we created a script called **IK_Scr** where we control all IK options for the parts of the body we need:

- Right Hand IK
- Left Hand IK
- Upper Body IK

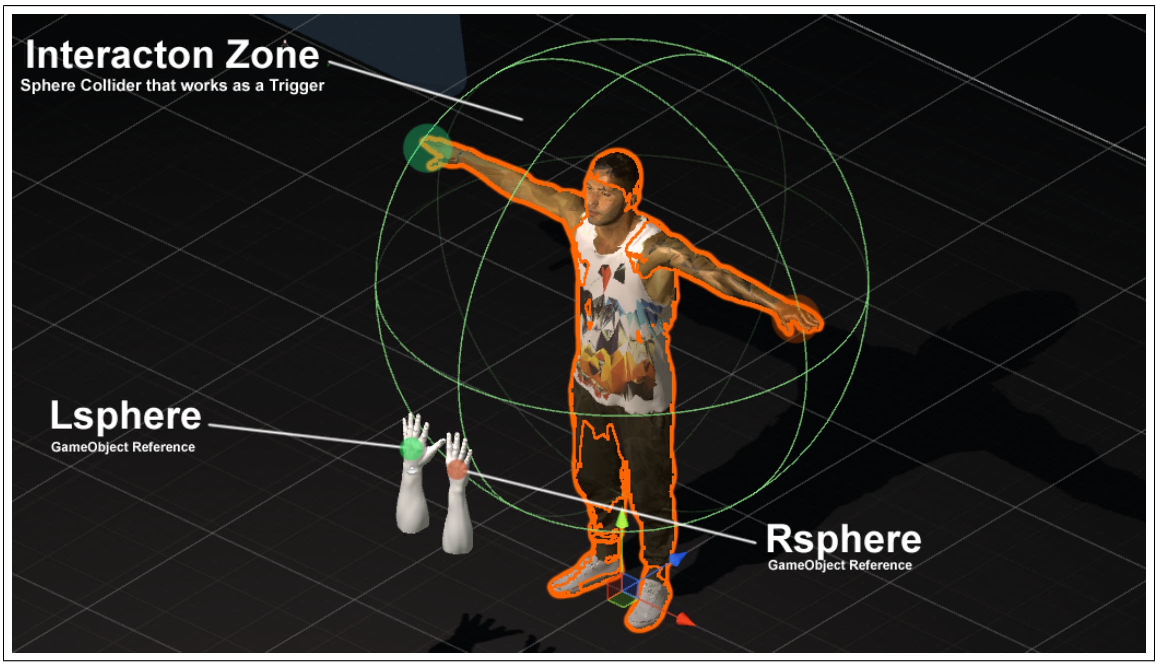
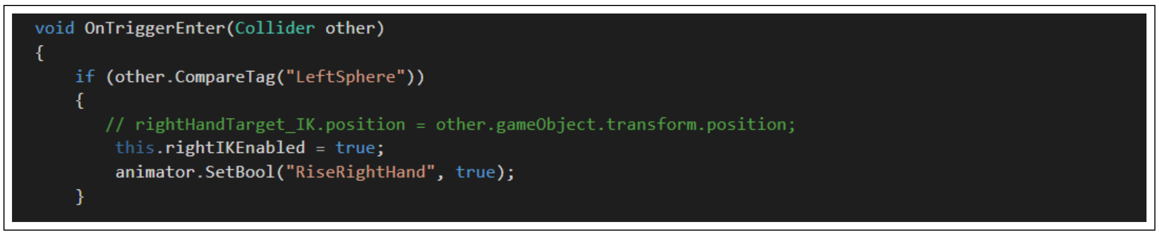


Figure 5.20: Interaction Zone

Right Hand IK In order to implement the interaction between the 3D humanoid character and the user's 3D hand model, we created a GameObject named **LeftIKControl** and added this object to the Left 3D Hand model of the User. This LeftIKControl contains a child GameObject named **Lsphere**. Lsphere has a collider and Rigid-body component inside. We will use this Lsphere GameObject as a reference in IK of the Right Hand of 3D Humanoid Character.

To import Lsphere as a reference to the IK_Scr script we added a Tag named "LeftSphere" to Lsphere GameObject. When the GameObject with Tag ""LeftSphere" enters the 3D Humanoid characters zone of interaction (Sphere collider that we use as a trigger see example 5.20), a function, OnTriggerEnter activates and the Right IK is set to Enable.



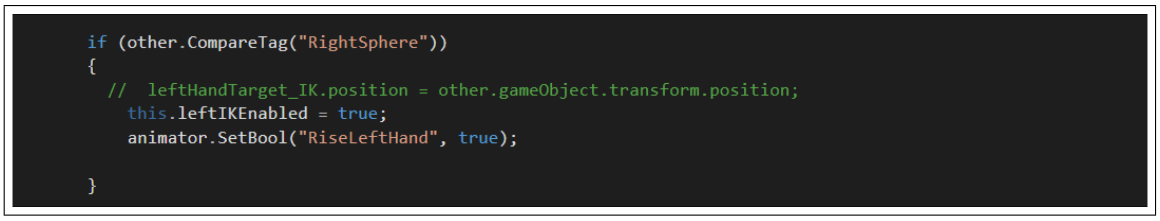
```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("LeftSphere"))
    {
       // rightHandTarget_IK.position = other.gameObject.transform.position;
        this.rightIKEnabled = true;
        animator.SetBool("RiseRightHand", true);
    }
```

Figure 5.21: Activation of RightIK

As long as the the reference of the users hand is in the Interaction zone

Left Hand IK In order to implement the interaction between the 3D humanoid character and the user's 3D hand model we created a GameObject named **RightIKControl** and added this object to the Right 3D Hand model of the User. The LeftIKControl contains a child GameObject named **Rsphere**. Lsphere includes a collider and Rigid-body component inside. We will use the Rsphere GameObject as a reference in IK kinematics of the Left Hand of 3D Humanoid Character.

In order to import Rsphere as a reference to the IK_Scr script we added a Tag named ”RightSphere” to Rsphere GameObject. When the GameObject with the tag ”RightSphere” enters the 3D Humanoid characters zone of interaction (Sphere collider that we use as a trigger see example 5.20), the function OnTriggerEnter is activated and the Left IK is set to Enable.

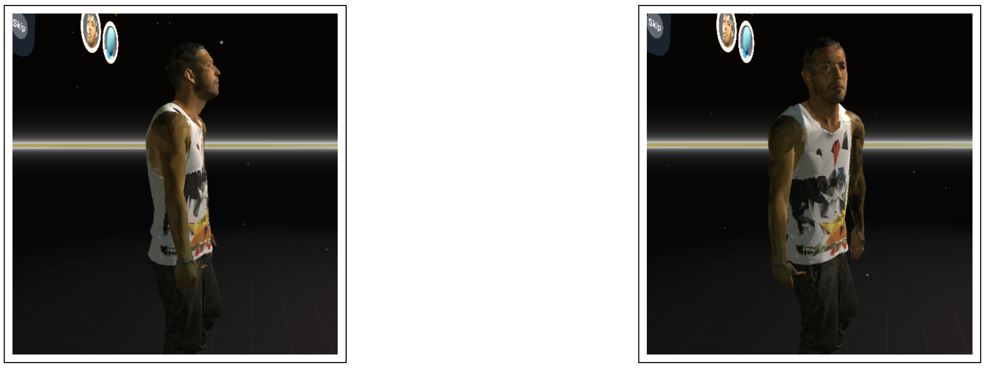


```
        if (other.CompareTag("RightSphere"))
        {
          //  leftHandTarget_IK.position = other.gameObject.transform.position;
            this.leftIKEnabled = true;
            animator.SetBool("RiseLeftHand", true);

        }
```

Figure 5.22: Activation of LeftIK

Upper Body IK In this part, we created IK control for the upper body of the 3D humanoid character to force the humanoid character always look at the user. These are complex IK movements that were applied to all upper bones of the character so that the 3D character rotates the upper body realistically staring at the user. Depending on the part of the body, we apply different weight of the rotation, see example 5.23.



(a) IK disabled (b) IK enabled

Figure 5.23: Upper body IK for ”Look At User” option

5.6 AI Navigation

The navigation system allows us to create characters that can intelligently move around the game world, using navigation meshes that are created automatically from the scene geometry. Dynamic obstacles allow altering the navigation of the characters at run-time. In our project, we do not have any obstacles and the reason is that we needed to minimize the distance the user needs to reach the 3D humanoid character, by guiding the 3D character move towards the user. We assigned the stop distance to 4 units in order give to the user a chance to walk with a minimum amount of steps to the interaction zone (Figure 5.20 we described before. The navigation mesh region of this project is shown in the Figure 5.24 visualized by the blue color.

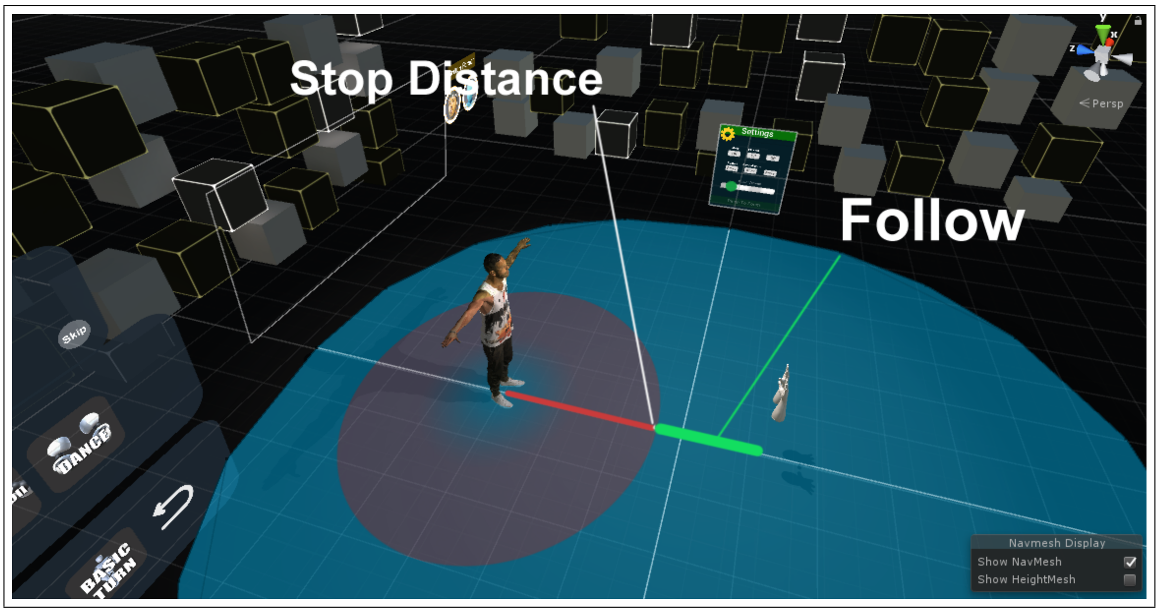


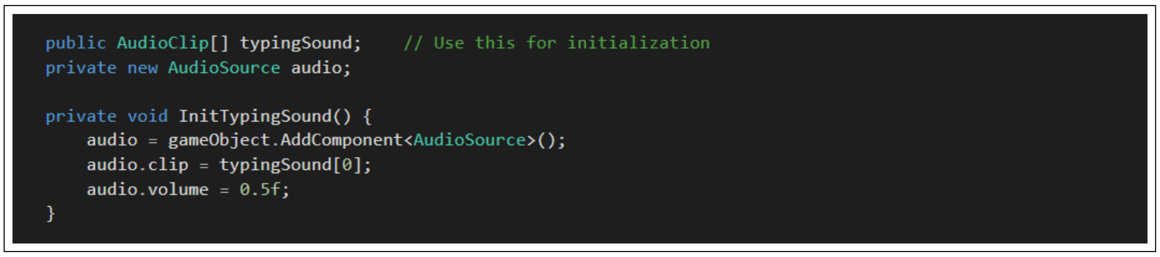
Figure 5.24: Navigation Mesh

5.7 Audio

A game would be incomplete without some kind of audio, be it background music or sound effects. Unity's audio system is flexible and powerful. It can import most standard audio file formats and has sophisticated features for playing sounds in 3D space, with optional effects like echo and filtering applied. Unity can also record audio from any available microphone on a users machine for use during game play or for storage and transmission. In this project, it is common that many distinct sound effects are played at the same time. For example , the user can hear the typing sound of the UI dialog canvas and Salsa music at the same time. Therefore, many audio sources are required. Even each GameObject has its own audio source. All main AudioSources are generated in _Scripts_ GameObject. There is one Audio Listener located in CenterEyeAnchor (Camera) and 1 Main Audio Sources:

- **Salsa Rythm** Located in _Scripts_ and start to play in the begging of the tutorial. The properties of volume is controlled with the Slide bar of the UI Settings.

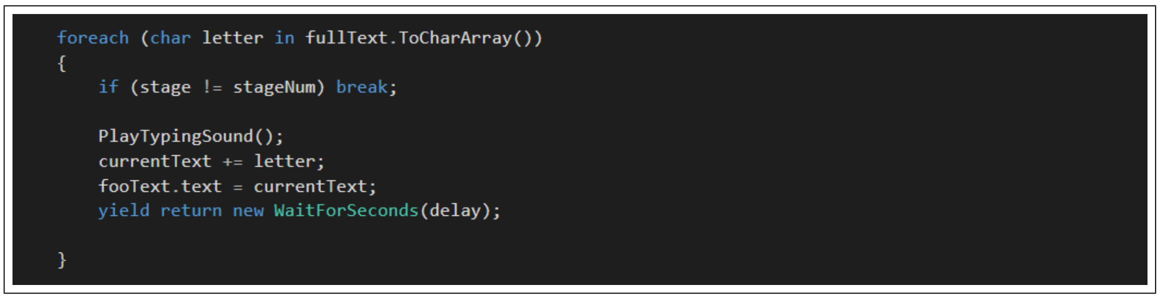
- **Typing Sound** Is generated with the script component with name Tutorial-StageScript that is located in _Scripts_. We initialize the Sound Source (see Figure 5.25) and add the Audio Clip to it when the game starts. Every time we want to show the message of the UI Dialog, we will play the clip for every letter of the Dialog text with a specific delay (see Figure 5.26).



```
public AudioClip[] typingSound;    // Use this for initialization
private new AudioSource audio;

private void InitTypingSound() {
    audio = gameObject.AddComponent<AudioSource>();
    audio.clip = typingSound[0];
    audio.volume = 0.5f;
}
```

Figure 5.25: Sound Initialization



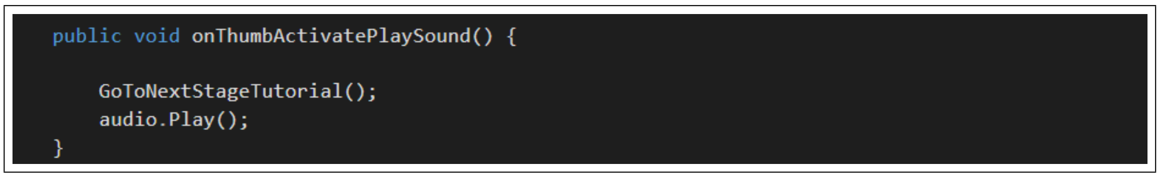
```
foreach (char letter in fullText.ToCharArray())
{
    if (stage != stageNum) break;

    PlayTypingSound();
    currentText += letter;
    fooText.text = currentText;
    yield return new WaitForSeconds(delay);

}
```

Figure 5.26: Sound Play for every char

- **Achievement Sound** Is generated based on the script component named MyGestureDetection that is located in _Scripts_. We initialize the Sound Source once the game starts and we add the Audio Clip. The sounds play every time a Thumb Up gesture is activated (see Figure 5.27).



```
public void onThumbActivatePlaySound() {

    GoToNextStageTutorial();
    audio.Play();
}
```

Figure 5.27: Sound Play for Thumb Up Gesture

5.8 Optimization

In this project, optimization plays an important role, especially when it comes to Virtual Reality which puts a big strain on the GPU compared to simple games that are displayed on a desktop monitor. Furthermore, a high frame rate is required. Developers have found several tricks to reduce the load on the GPU and this is basically conducted by rendering as few objects as possible. This is one of the main reasons low poly games have become so widespread in the VR gaming communities.

Lighting Baking Lightmapping calculates the lighting for static geometry and saves it in textures. The benefits are, that it has awesome lighting effects, such as light bouncing, Ambient Occlusion and Area lights. This means that lighting will look better without having to do much work. We can select between single and dual lightmaps. Single means it replaces any dynamic shadows and lighting, only using the baked maps. This increases performance quite a bit, but reduces what you can do with lighting dynamically. With dual lighmaps, Unity bakes both lightmaps with shadows and without. It then uses the shadow distance to determine where to fade between lightmap shadows and real-time shadows. We used the dual method in this thesis.

Chapter 6

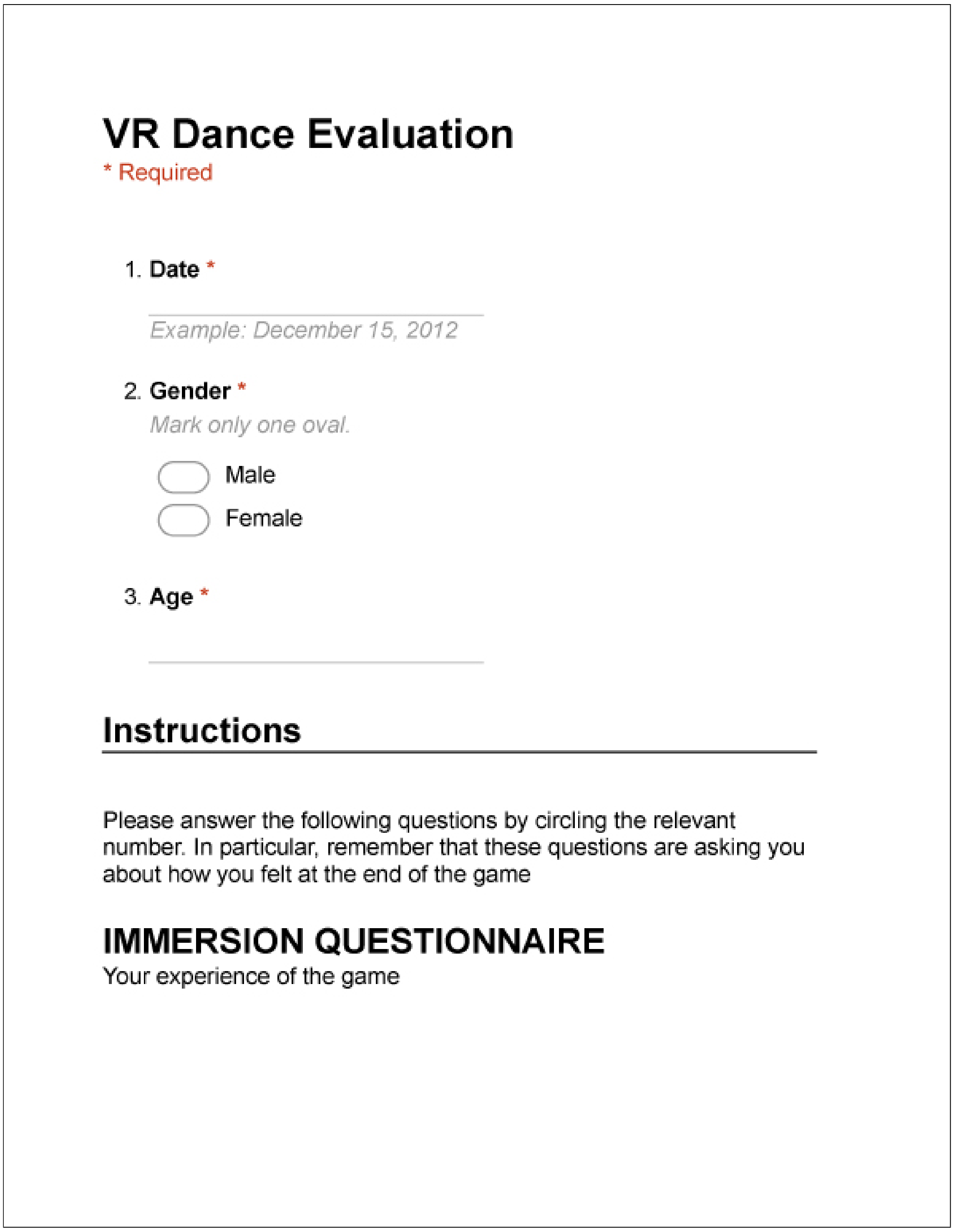
Evaluation

This chapter shows the evaluation of the current VR training application by various users. In particular, users evaluated this VR application investigating their perceived level of immersion and presence as well as any motion sickness symptoms they experienced.

6.1 Evaluation method

For the system evaluation, we used the *Experience of Immersion* in Games and the *Simulator Sickness* Questionnaire to determine the degree of immersion perceived and possible symptoms of dizziness and discomfort respectively.

The evaluation process consisted of 20 users (12 men and 8 women) aged 21-30 years old. The majority of the users were students (undergraduates, graduates) of the Technical University of Crete.



VR Dance Evaluation

* Required

1. **Date ***

   ____
   Example: December 15, 2012

2. **Gender ***
   Mark only one oval.

   ◯ Male
   ◯ Female

3. **Age ***

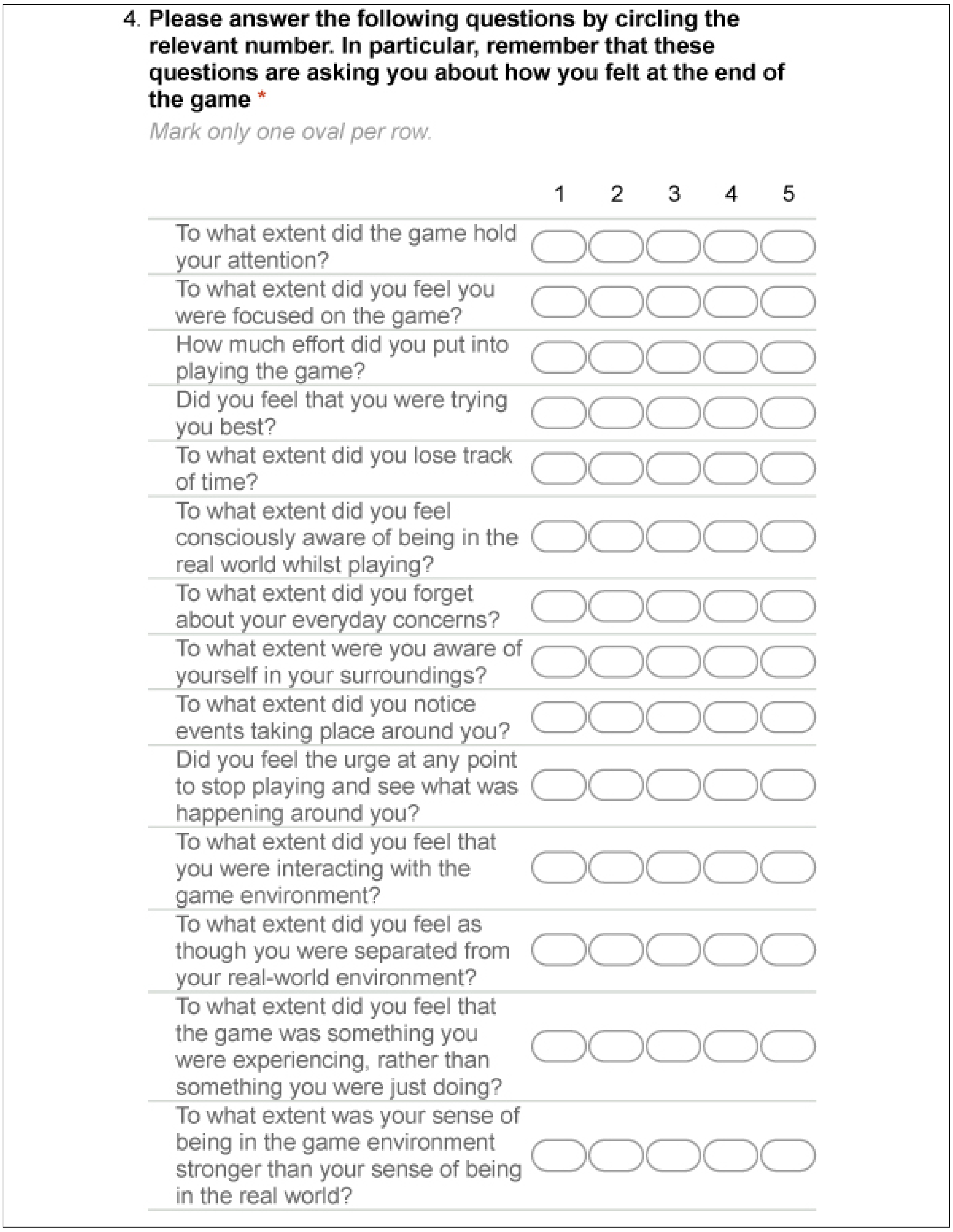
   ____

Instructions

Please answer the following questions by circling the relevant number. In particular, remember that these questions are asking you about how you felt at the end of the game

IMMERSION QUESTIONNAIRE

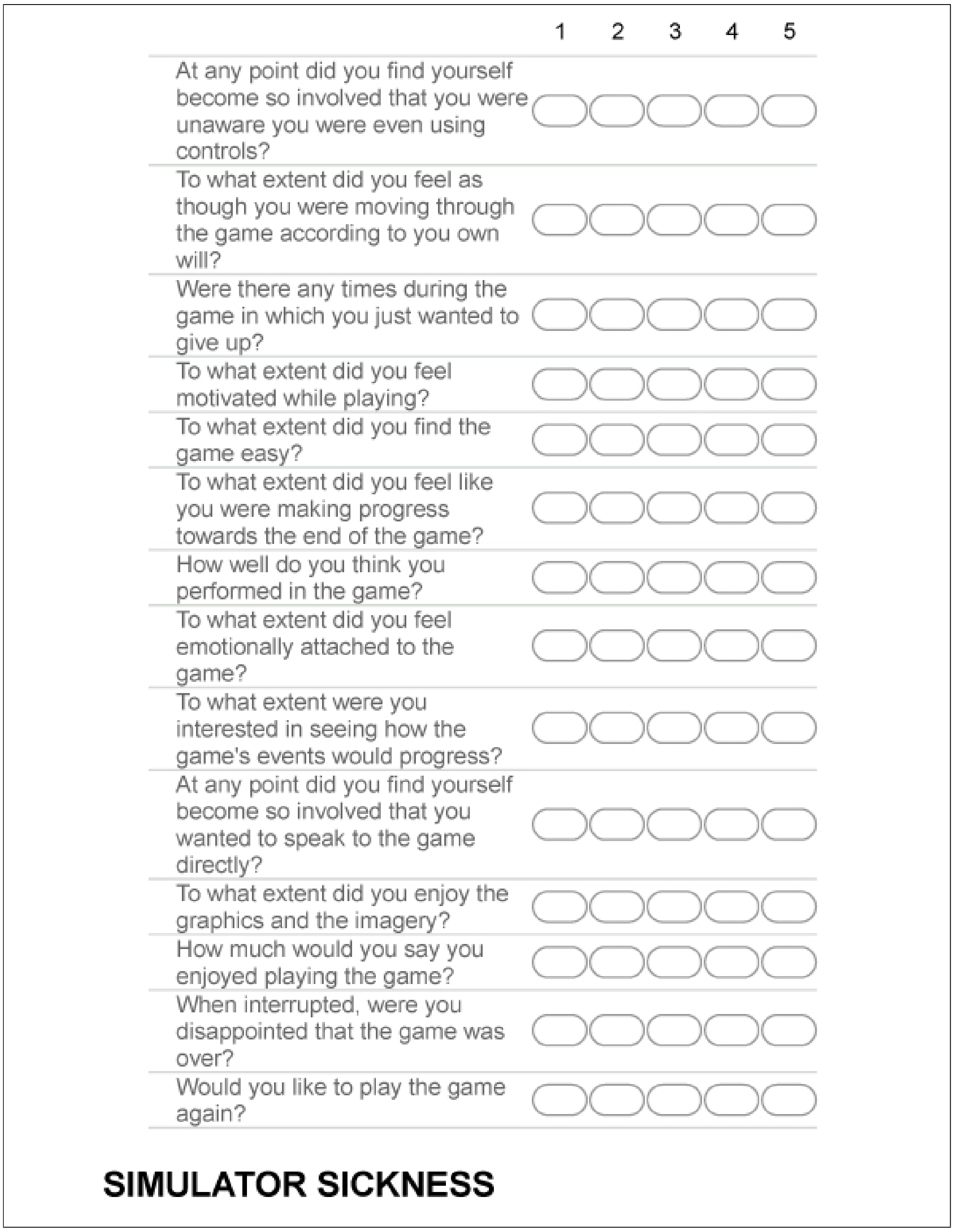
Your experience of the game



4. **Please answer the following questions by circling the relevant number. In particular, remember that these questions are asking you about how you felt at the end of the game ***

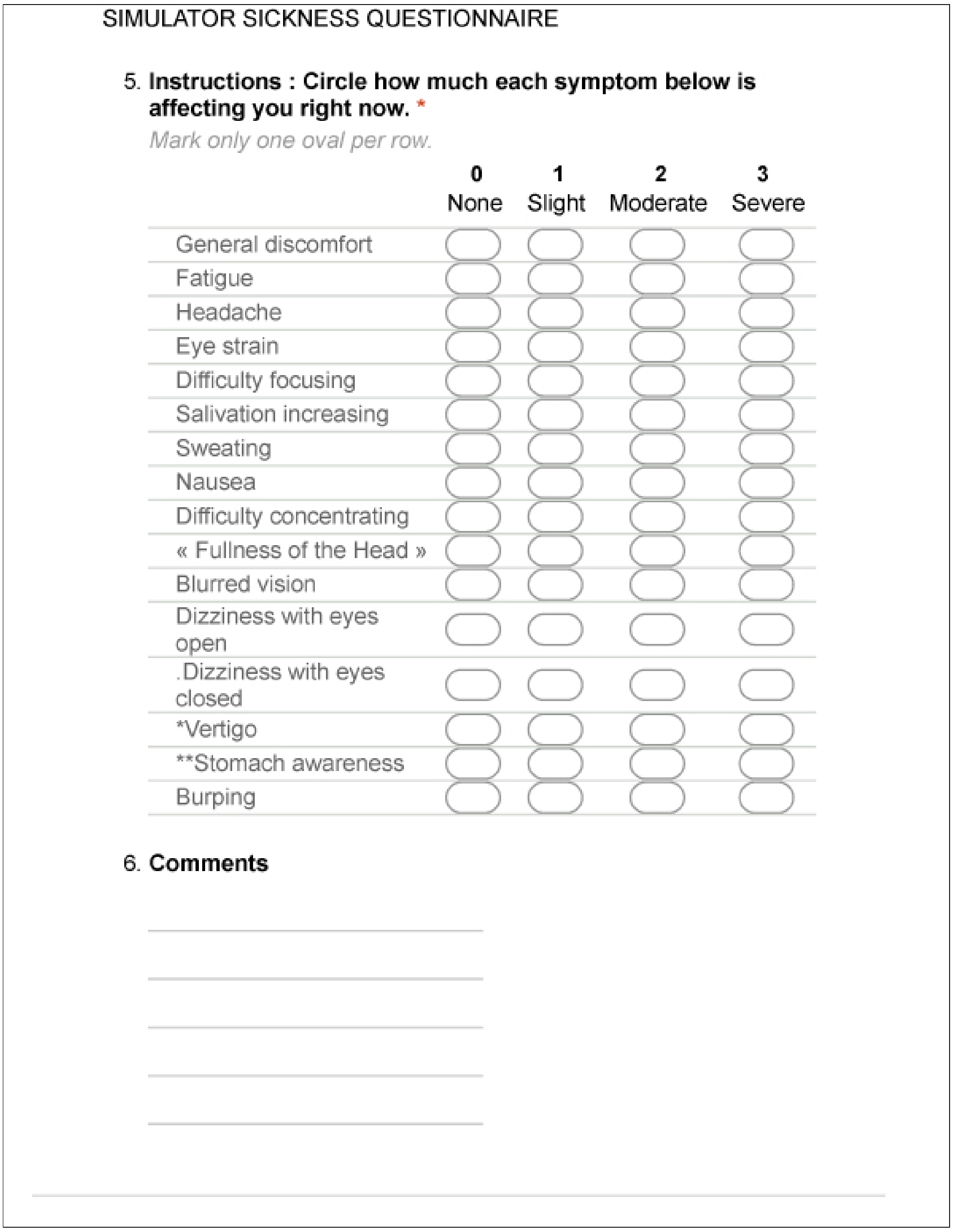
Mark only one oval per row.

	1	2	3	4	5
To what extent did the game hold your attention?	○	○	○	○	○
To what extent did you feel you were focused on the game?	○	○	○	○	○
How much effort did you put into playing the game?	○	○	○	○	○
Did you feel that you were trying you best?	○	○	○	○	○
To what extent did you lose track of time?	○	○	○	○	○
To what extent did you feel consciously aware of being in the real world whilst playing?	○	○	○	○	○
To what extent did you forget about your everyday concerns?	○	○	○	○	○
To what extent were you aware of yourself in your surroundings?	○	○	○	○	○
To what extent did you notice events taking place around you?	○	○	○	○	○
Did you feel the urge at any point to stop playing and see what was happening around you?	○	○	○	○	○
To what extent did you feel that you were interacting with the game environment?	○	○	○	○	○
To what extent did you feel as though you were separated from your real-world environment?	○	○	○	○	○
To what extent did you feel that the game was something you were experiencing, rather than something you were just doing?	○	○	○	○	○
To what extent was your sense of being in the game environment stronger than your sense of being in the real world?	○	○	○	○	○



	1	2	3	4	5
At any point did you find yourself become so involved that you were unaware you were even using controls?					
To what extent did you feel as though you were moving through the game according to you own will?					
Were there any times during the game in which you just wanted to give up?					
To what extent did you feel motivated while playing?					
To what extent did you find the game easy?					
To what extent did you feel like you were making progress towards the end of the game?					
How well do you think you performed in the game?					
To what extent did you feel emotionally attached to the game?					
To what extent were you interested in seeing how the game's events would progress?					
At any point did you find yourself become so involved that you wanted to speak to the game directly?					
To what extent did you enjoy the graphics and the imagery?					
How much would you say you enjoyed playing the game?					
When interrupted, were you disappointed that the game was over?					
Would you like to play the game again?					

SIMULATOR SICKNESS



SIMULATOR SICKNESS QUESTIONNAIRE

5. **Instructions : Circle how much each symptom below is affecting you right now.** *

Mark only one oval per row.

	0 None	1 Slight	2 Moderate	3 Severe
General discomfort	◯	◯	◯	◯
Fatigue	◯	◯	◯	◯
Headache	◯	◯	◯	◯
Eye strain	◯	◯	◯	◯
Difficulty focusing	◯	◯	◯	◯
Salivation increasing	◯	◯	◯	◯
Sweating	◯	◯	◯	◯
Nausea	◯	◯	◯	◯
Difficulty concentrating	◯	◯	◯	◯
« Fullness of the Head »	◯	◯	◯	◯
Blurred vision	◯	◯	◯	◯
Dizziness with eyes open	◯	◯	◯	◯
.Dizziness with eyes closed	◯	◯	◯	◯
*Vertigo	◯	◯	◯	◯
**Stomach awareness	◯	◯	◯	◯
Burping	◯	◯	◯	◯

6. **Comments**

6.1.1 Procedure

The survey was performed as an online form that consisted of 3 pages; the first was an introduction containing the requirements for participation in the study, followed with the *Defining an Experience of Immersion* Questionnaire and *Simulator Sickness* Questionnaire.

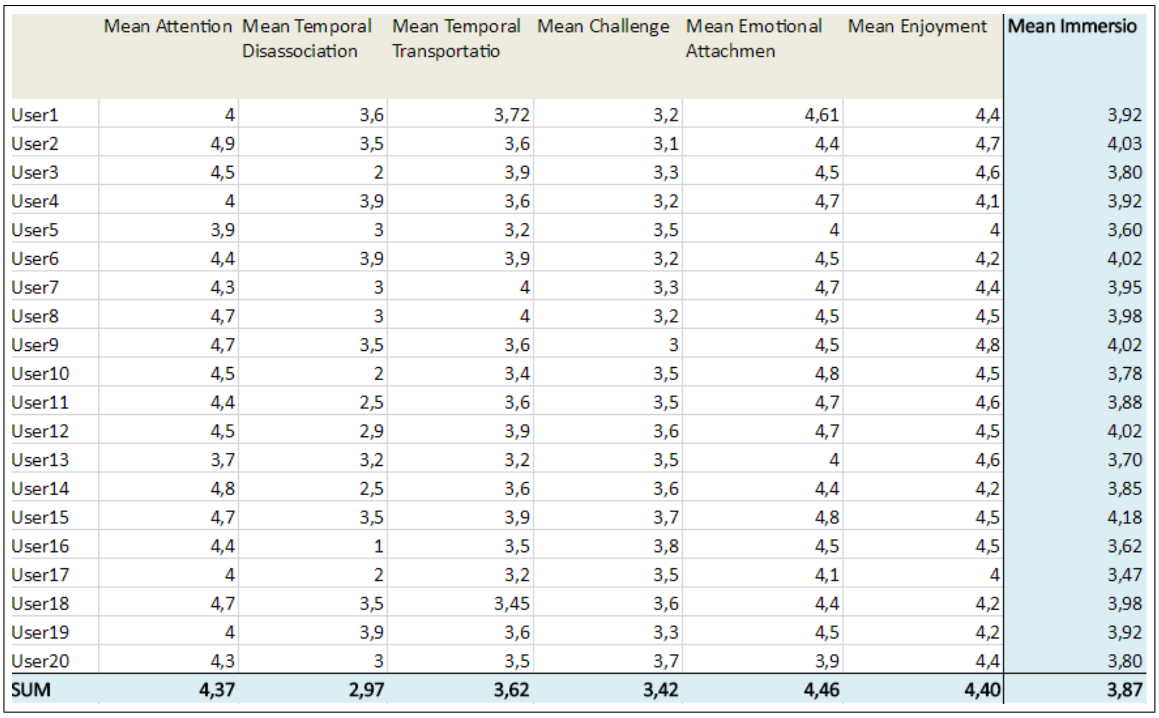
6.2 Evaluation Goals & Result

The above assessment consisted of two phases. First, the aim was to examine the degree of immersion in the Virtual Reality environment and secondly, to ascertain whether and to what extent symptoms of discomfort have been caused to users after exposure.

Defining an Experience of Immersion (IEQ) consists of 6 different types of questions, which are described by the indicators below. The questions were rated from [min = 1 to Max =5], with 1 signifying the minimum grade and 5 the maximum. [9]

- **Attention:** Relative to how much the user was concentrated (1, 2, 3, 4).
- **Temporal Disassociation:** Lack of sense of time during simulation (5, 6, 7, 8, 9, 10).
- **Temporal Transportation:** The extent to which the user felt he was more part of the game than the real environment (11, 12, 13, 14, 15, 16).
- **Challenge:** Relates to the game play of the game (fl ow) (17, 18, 19, 20, 21).
- **Emotional Attachment:** Emotional Attack, due to the feeling of being part of the virtual environment (22,23,24).
- **Enjoyment:** The extent to which the user was happy (25, 26, 27,28).

After users self-report answering the 6 questions above, we calculated the response averages and the exported results are shown below:



	Mean Attention	Mean Temporal Disassociation	Mean Temporal Transportatio	Mean Challenge	Mean Emotional Attachmen	Mean Enjoyment	**Mean Immersio**
User1	4	3,6	3,72	3,2	4,61	4,4	3,92
User2	4,9	3,5	3,6	3,1	4,4	4,7	4,03
User3	4,5	2	3,9	3,3	4,5	4,6	3,80
User4	4	3,9	3,6	3,2	4,7	4,1	3,92
User5	3,9	3	3,2	3,5	4	4	3,60
User6	4,4	3,9	3,9	3,2	4,5	4,2	4,02
User7	4,3	3	4	3,3	4,7	4,4	3,95
User8	4,7	3	4	3,2	4,5	4,5	3,98
User9	4,7	3,5	3,6	3	4,5	4,8	4,02
User10	4,5	2	3,4	3,5	4,8	4,5	3,78
User11	4,4	2,5	3,6	3,5	4,7	4,6	3,88
User12	4,5	2,9	3,9	3,6	4,7	4,5	4,02
User13	3,7	3,2	3,2	3,5	4	4,6	3,70
User14	4,8	2,5	3,6	3,6	4,4	4,2	3,85
User15	4,7	3,5	3,9	3,7	4,8	4,5	4,18
User16	4,4	1	3,5	3,8	4,5	4,5	3,62
User17	4	2	3,2	3,5	4,1	4	3,47
User18	4,7	3,5	3,45	3,6	4,4	4,2	3,98
User19	4	3,9	3,6	3,3	4,5	4,2	3,92
User20	4,3	3	3,5	3,7	3,9	4,4	3,80
SUM	**4,37**	**2,97**	**3,62**	**3,42**	**4,46**	**4,40**	**3,87**

Figure 6.1: IEQ Mean results

Result of the Evaluation:

- Total Mean Attention = **4.37**
- Total Mean Temporal Disassociation = **2,97**
- Total Mean Temporal Transportation = **3,62**
- Total Mean Challenge = **3,42**
- Total Mean Emotional Attachment = **4,46**
- Total Mean Enjoyment = **4,40**
- Total Mean Immersion = **3,87**

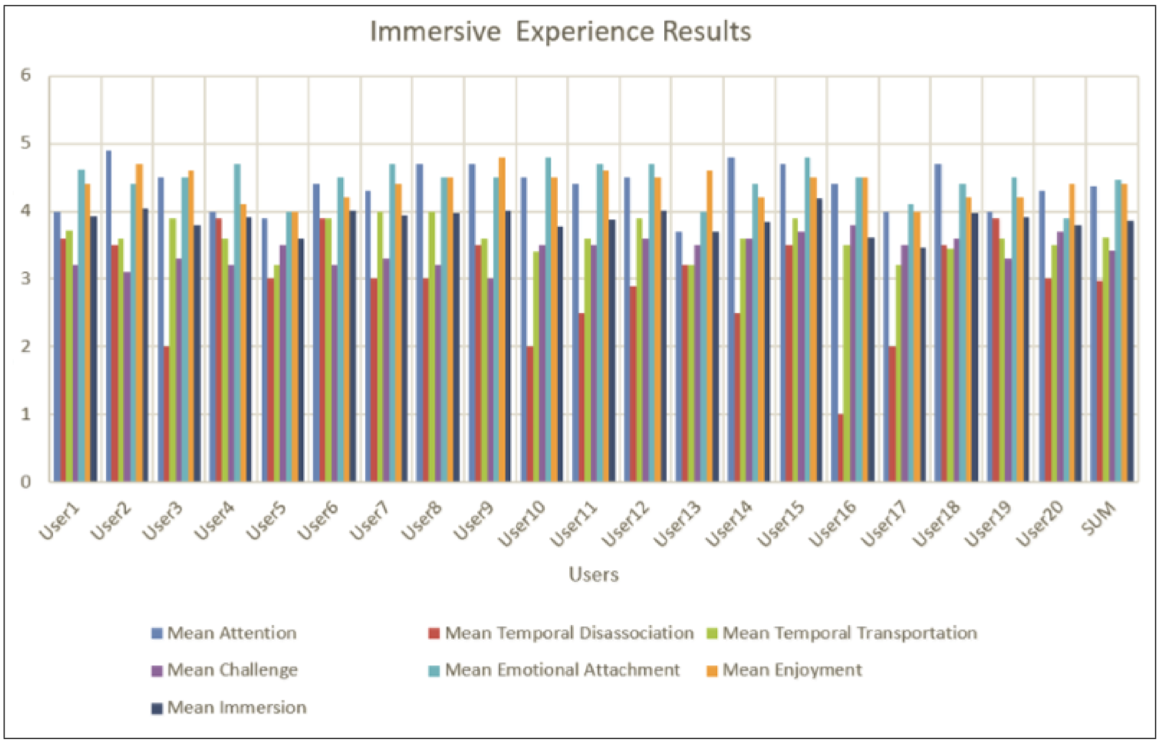


Figure 6.2: IEQ Results

Simulator Sickness Questionnaire The Simulator Sickness Questionnaire (SSQ) has so far been the standard for measuring motion sickness risks that a simulation can cause in a Virtual Reality environment. The SSQ includes a list of 16 symptoms that can be caused by exposure to a simulator. For each of the 16 symptoms, there are four possible options depending on the level to which users have experienced them [None=0, Slight = 1, Moderate=2, Severe=3]. A final simulation symptom score is produced.

In any case, zero represents the absence of stimulant disease symptoms, and the higher the score, the more severe the symptoms are. After completing the questionnaires, the Total mean value was calculated for each disease symptom for all users:

The sickness symptoms of the simulation is generally the result of the inconsistency between the simulation of movement within the virtual environment and the

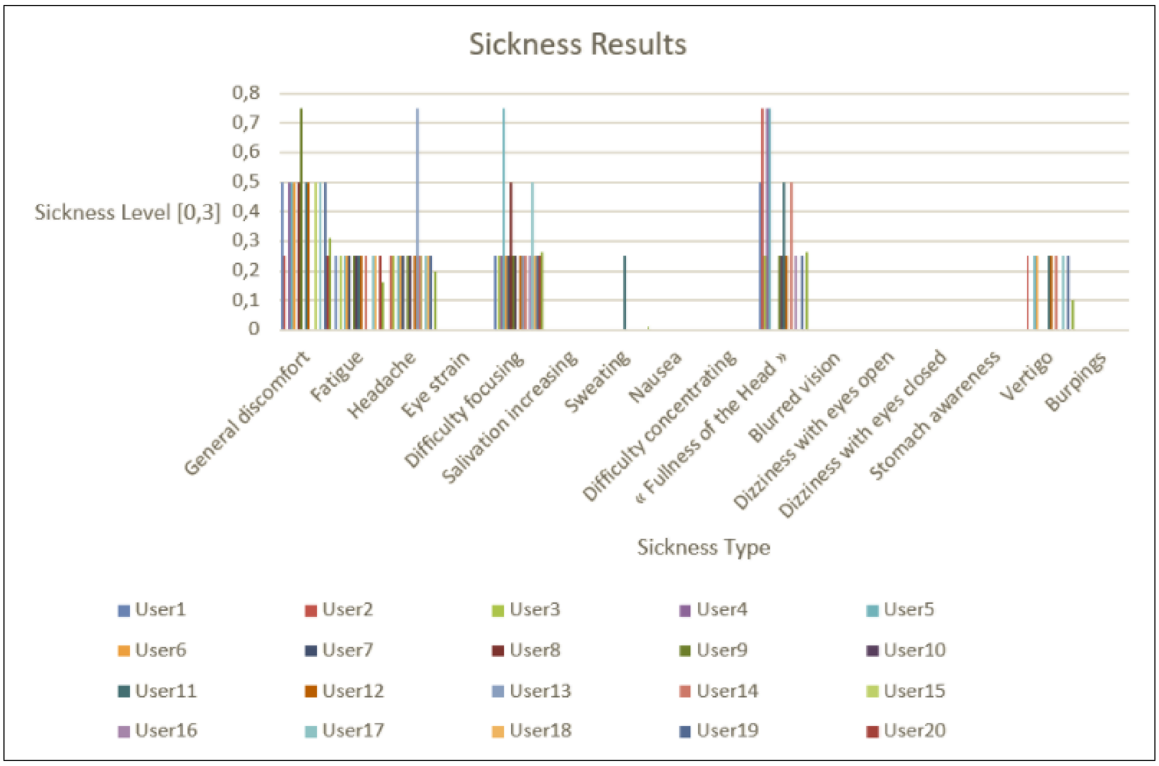


Figure 6.3: SSQ Results

sensation resulting from the vestibular system. In many simulations, the visual system receives information related to movement, but the vestibular system interprets a static state that can not be synchronized with the visual perception of motion. This difference is causing symptoms of Simulator Sickness to many people, at varied levels.

6.3 Conclusion of Evaluation

After the evaluation of the IEQ and the SSQ, in addition to users' comments after the evaluation process was ended, we conclude:

- **Immersive Experience Questionnaire** The results obtained from this questionnaire are quite satisfying. Users were generally very satisfied with the application, which is confirmed by the above indicators. The **Mean Immersion** is 3,87 with the lowest value 3,6 and the highest value 4,18. Most of the users enjoyed playing this game while their Mean Attention was very high (4,37).

- **Simulator Sickness Questionnaire** The SSQ results show that all participants had even a minor amount of motion sickness symptoms. However, certain of potential symptoms presented in the Questionnaire, were not reported as perceived during and after the evaluation.

Comments *Users were generally very impressed and excited by the dance training experience, especially the interaction that they had with the 3D humanoid character. Some of them said that they would consider buying the dance training application in Steam VR. The gameplay learning curve was also easy. Additional possibilities* *of interaction were desired, such as assigning to the 3D humanoid character more complex movements.*

The most interesting observation we acquired, is that after completing the evaluation, users reported feeling more confident asking a stranger for a dance at a social event.

Chapter 7

Conclusion

7.1 Main Contributions

The effectiveness of a Virtual Environments for training has often been linked to the sense of presence reported by the users. Presence is defined as the subjective experience of being in one place or environment, even when one is physically situated in another. Virtual Reality is on a good track in getting into our lives, not only for entertainment, but also for other purposes, such as education, medicine and simulations. As the technology grows people get less socialized and a person to person contact gets more difficult for them. Dancing makes people more happy and social. In this project we took the first step in aiding a user feel more comfortable participating in social dances.

7.2 Future Work

The implementation of this project as well as the experiments described in detail in Chapter 6, are formally designed. However, certain improvements could be accomplished if the following actions materliaze:

- Realistic scene & Characters. Because of the performance of our PC, we did not aim to produce a high quality of graphics, so that the VR game has no lags during the game-play. The alternative was to create a realistic game with real time shadows and light for all objects in the scene, if computational power was available.
- It would be a great idea if we analyze every music clip we add in the game, to count the tempo and the beats of the soundtrack. If we do that, we can synchronize the animations with the tempo of the music so that a 3D humanoid character would execute animations based on the tempo of the music selected by the user and not just offer pre-installed music clips.
- Animations could be more complex, if motion capture systems available to us could record motion in higher resolution. There are complex combinations of motion which can be imported in the tutorial allowing a developer to create animations without editing them with another program. Simply, the range of actions would be to execute the motion combination, capture it and apply it to the tutorial.

7.2.1 Recommendations for Developers

With the widespread availability and affordability of VR headsets, the number of people developing VR experiences has grown dramatically in recent years. Most VR developers to date have significant experience in the video game industry, where their skills and experience in developing games and game engines are "ported over" to VR. In some cases, simple adaptations are sufficient, but game developers have been repeatedly surprised at how a highly successful and popular game experience does not translate directly to a comfortable, or even fun, VR experience. The reason for this would be a lack of understanding of human physiology and perception. As the field progresses, developers are coming from an increasing variety of backgrounds, including cinema, broadcasting, communications, social networking, visualization, and engineering. Artists and hobbyists have also joined in to produce some of the most innovative experiences.This section provides certain useful recommendations based on the principles covered in this thesis. The vast majority of VR experiences to date are based on successful 3D video games, which is evident in the kinds of recommendations being put forward by developers today. Most of the recommendations below link to prior parts of this thesis providing scientific motivation.

Virtual worlds

- Set units in the virtual world to match the real world so that scales can be easily matched. For example, one unit equals one meter in the virtual world. This helps with depth and scale perception.
- Make sure that objects are fully modeled so that missing parts are not noticeable when the user looks at them from varied viewpoints which would not been possible when graphics are displayed on a screen.
- Design the environment so that less locomotion is required.
- Consider visual and auditory rendering performance issues and simplify the geometric models as needed to maintain the desirable frame rates on targeted hardware.

Visual rendering

- Never allow words, objects, or images to be fixed to parts of the screen; all content should appear to be embedded in the virtual world.
- Be careful when adjusting the field of view for rendering or any parameters that affect lens distortion that so the result does not cause further mismatch.
- The rendering system should be optimized so that the desired virtual world can be updated at a frame rate that is at least as high as the hardware. requirements (for example, 90 FPS for Oculus Rift and HTC Vive); otherwise, the frame rate may decrease and vary, which causes discomfort.

- Avoid movements of objects that cause most of the visual field to change in the same way; otherwise, the user might feel as if moving.
- Determine how to cull away geometry that is too close to the face of the user; otherwise, substantial vergence-accommodation mismatch will occur.
- Unlike in games and cinematography, the viewpoint should not change in a way that is not matched to head tracking, unless the intention is for the user to feel as if moving in the virtual world, which itself can be uncomfortable.
- For proper depth and scale perception, the interpupillary distance of the user in the real world should match the corresponding viewpoint distance between eyes in the virtual world.
- In comparison to graphics on a screen, reduce the brightness and contrast of the models to increase VR comfort.

Tracking and the matched zone

- Never allow head tracking to be frozen or delayed; otherwise, the user might immediately perceive self-motion
- Make sure that the eye viewpoints are accurately located.
- Beware of obstacles in the real world that do not exist in the virtual world; a warning system may be necessary as the user approaches an obstacle.
- Likewise, beware of obstacles in the virtual world that do not exist in the real world. For example, it may have unwanted consequences if a user decides to poke his head through a wall.
- As the edge of the tracking region is reached, it is more comfortable to gradually reduce contrast and brightness than to simply hold the position fixed.

Interaction

- Consider interaction mechanisms that are better than reality by giving people superhuman powers, rather than applying the universal simulation principle.
- For manipulation in the virtual world, try to require the user to move as little as possible in the physical world; avoid giving the user a case of gorilla arms.

User interfaces

- If a floating menu, web browser, or other kind of virtual display appears, then it should be rendered at least two meters away from the user's viewpoint to minimize vergence-accommodation mismatch.
- Such a virtual display should be centered and have a relatively narrow field of view, approximately one-third of the total viewing area, to minimize eye and head movement.
- Embedding menus, options, game status, and other information may be most comfortable if it appears to be written into the virtual world in ways that are familiar.

Audio

- Be aware of the difference between a user listening over fixed, external speakers versus attached headphones; sound source localization will not function correctly over headphones without tracking.
- The Doppler effect provides a strong motion cue.

Self appearance

- The feeling of being present in the virtual world and the ability to judge scale in it are enhanced if the user is able to see her corresponding body in VR.
- A simple virtual body is much better than having none at all; the presence of user's hands enhances the feeling of immersion.
- Unexpected differences between the virtual body and real body may be alarming. They could have a different gender, body type, or species. This could lead to a powerful experience, or could be an accidental distraction.
- If only head tracking is performed, then the virtual body should satisfy basic kinematic constraints, rather than decapitating the user in the virtual world.
- Users' self-appearance will affect their social behavior, as well as the way people around them react to them.

Bibliography

[1] Khullani M. Abdullahi. The reality-virtuality continuum.

[2] H. Takemura A. Utsumi F. Kishino Milgram, Paul. Augmented reality: A class of displays on the reality-virtuality continuum. *Proceedings of Telemanipulator and Telepresence Technologies*, pages 2351–34, 1994.

[3] Steuer Jonathan. Defining virtual reality: Dimensions determining telepresence. *Journal of Communications*, 42(4):73–93, 1992.

[4] SIGGRAPH 94. Character motion systems course 9.

[5] Wiki. Methos and systems of mocap.

[6] Wiki. Microsoft kinect.

[7] Unity Technologies. Inverse kinematic.

[8] Wiki. Salsa (dance).

[9] P Cairns S Dhoparee A Epps T Tijs A Walton C Jennett, AL Cox. Quantifying the experience of immersion in games. *International journal of human-computer studies*, 66(9):641–661, 2008.