Universidad Carlos III de Madrid

Departamento de Ingeniería Telemática

TRABAJO FIN DE GRADO

Development of a virtual reality environment to study psychomotor skills in children

Autor: Raúl Araújo Álvarez

Tutora: Mª del Carmen Fernández Panadero

Madrid, Junio 2015

Acknowledgments

To my family, specially my parents for their unconditional support in any situation.

To my friends for being always there and giving me a hand whenever I needed it.

To my tutor M^{a} Carmen for her invaluable guidance and effort to make this project possible.

And finally to Rafa for adding magic to this technical project.

Abstract

Even though Virtual Reality is not a new technology, it is experiencing a renaissance and expansion to different fields with a great potential due to the development of new devices. This fact in addition to the interest that videogames awake in children, motivated the development of this project: a system that could take advantage of the current rise of Virtual Reality for its use as an educational tool without letting the fun factor out of the equation.

More specifically we designed and developed a set of three Virtual Reality games for children to exercise their psycho-motor skills: The first one allows the player to become a Pegasus, the second lets him take the reins of this Pegasus to go for a ride and in the third one the player becomes an elf who has to face a sword training session. These games are based on a fantasy novel which served a double purpose: making the system more appealing and, more importantly, promoting reading habits in children.

Since one of the requirements that we set was not to use traditional input methods, we carried out an analysis of the current technologies used for motion tracking. The conclusion was that we should partially develop a custom solution to keep simplicity and budget low.

The finally developed system was successfully tested in a primary school with a group of children of 8 to 9 years of age. After embodying different characters from the story, they showed an increase of interest for them, their adventures and thus for reading the book.

Keywords: Virtual Reality, psycho-motor, embodiment, skills, game, simulation, Oculus Rift, Kinect, Android, bluetooth, Windows, Unity, Blender, 3D, .NET, C#, Java, fantasy, children, book, reading

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since its early beginnings, Virtual Reality (VR) has been a technology with an enormous potential due to its inherent three-dimensional structure, both in terms of display and interaction. According to Bryson [1], the vision of a three-dimensional environment applied to three-dimensional tasks and of highly intuitive interfaces which make the computer hardware "invisible" seems to be an entirely reasonable and desirable vision.

However, the adoption of Virtual Reality (VR) has been neither as fast nor as extensive as expected at the beginning. There could be a variety of reasons for this failure, but one of the most important was that the available interface hardware (Head-mounted Display (HMD), trackers, etc.) during the first decades failed to deliver the effects of immersion or presence required for many tasks mainly due to performance issues.

Recent advances in stereoscopic displays such as VR glasses (e.g: Oculus Rift, Samsung Gear) and other consumer devices available on the mass market such as hand-held terminals (smartphones, tablets), and game platforms (Wii, PlayStation) makes the technology of sensors and trackers much more affordable for the users. This fact has allowed virtual reality to emerge from the simulation field to a wider range of different applications such as manufacturing [2], neuropsychology [3] ,rehabilitation [4], [5] and for leisure (games, films) [6].

Additionally, the relationships between body, mind and emotions have been widely exploited in high level sports and dance performance, but its penetration in academic environments is still very slow [7]. The theory of multiple intelligences by Gardner [8], the growing interest in embodied cognition [9] and studies supporting the connection between motor activity and brain neuroplasticity [4], [10] have once again drawn attention to the potential of body-mind-emotions connections in learning environments.

Taking advantage of these two tendencies and the fact that several authors have highlighted the potential of VR to provide a multi-sensory learning feedback, the main objective of this project is to assess the integration possibilities of the current commercial devices and develop a prototype that consists of a set of simple games, which involve physical activity and explore different ways of interaction with the virtual world. Finally it would be necessary to tests these games in a primary school with children in order to explore their reactions and impressions.

Presenting the idea of this activity in a school before starting the actual development allowed to redefine the initial objectives for a better integration of the games within the activities carried out there as well as a more practical application.

In this chapter we describe how this project started and how its initial conception evolved to the final idea. This evolution had an effect in the project objectives and motivation as we will explain.

A brief overview of this document is presented as well.

1.1 Project History

The starting point for this project was the idea of researching and implementing new ways for user interaction inside Virtual Reality (VR) environments. Then creating some simple demos, with a videogame format, to test our work with final users so we could study their reactions and feedback.

For our testing scenario we contacted a primary school which happened to be carrying out activities to develop multiple intelligence in children. At this point we realized our project could fit as a part of this set of activities if we modified our initial objectives and requirements (as explained in the next section) an so we did.

This way we could integrate our project with the curricular activities programmed for the children's education as established by "Real Decreto 89" which regulates the educative contents in Comunidad de Madrid. Specially, article 10 makes reference to "playing the role of different characters from tales and stories" [11].

Figure 1.1: "El Secreto de Marcos" book cover.

These activities were based on a fantasy novel for children titled "El Secreto de Marcos"[12]. So it seemed a good idea to adapt our project visual and artistic style to the one described in the book. We contacted the author, Rafael Nieto, to learn more about this fantasy world and discuss some of the artistic aspects. We thought that the fact of basing the game on a book could help promoting children's reading as well.

So the relatively simple initial idea of a demo, developed into creating a VR experience set in a fantasy world and then testing it with children as a part of their school activities.

1.2 Project Motivation

When children finish their second year of primary school education (between 8 and 9 years of age) they should have acquired the habit of reading since they just learned to do so. Unfortunately, kids at that age do not find books attractive since there are plenty of ways of entertainment that are more dynamic and immediate such as television or videogames.

In the last decades, several attempts have been made to transport literature into virtual reality [13], [14], in these cases the idea is to recreate traditional literary narratives inside the virtual world. The usual method is to recreate the atmosphere and deconstruct the text so that the reader can be immerse in the environment and play a more active and closer to the game role, enabling interaction with different elements and giving some degree of freedom to make plot choices.

In our case the approach taken is different, we do not want to develop a new way to present the book as a whole. We want to develop an activity that encourages the children to read the book in its traditional format. Our approach is based on the fact that, by embodying some of the characters, the children will be much more interested in knowing about them and their adventures and so they will be more curious about the book.

1.3 Project Objectives

Once we had contacted the school and the author of the book, we had all the necessary information to define the following objectives:

1.3.1 Functional Objectives

- **FO-01: Create a VR experience, in the form of a game, which produces a sense of embodiment being intense, immersive and fun.** Since the main users will be kids we want to create the feeling of really being transported to a magical world and embodying the different characters with the final goal of waking their curiosity about the book.

- **FO-02: The game atmosphere, locations and characters have to be based on the novel "El Secreto de Marcos".** As we mentioned before, our project will be integrated into the set of activities that the school was carrying out so it has to fit within the global theme of these activities.

- **FO-03: Testing the game with final users and gathering their feedback, reactions and opinions.** For us, one of the most important objectives is being able of actually testing the results of our work and check that it is useful in some way. Testing with final users is also the most effective way to improve our work.

1.3.2 Technical Objectives

- **TO-01: Develop one experience where the user controls the avatar using his body.** The player must take control of a non human character, but his body movements must be closely mapped to this character.
- **TO-02: Develop one experience where a physical object is used to interact with the virtual world.** Using a physical, real world object (but not a traditional game-pad or controller), the player will control some element inside the virtual world.
- **TO-03: Develop a system that is easy to setup.** The project is intended to be tested with a high number of children in a relatively small time lapse, so we want short setup and configuration times between users. This implies to use as few cables as possible and quick calibration phase.
- **TO-04: Develop a system that is affordable.** Since we will be working with children we want to create a fun experience that requires only affordable consumer grade technology so that we don't have to worry too much about the equipment in case of it breaking down. We also wanted to keep the budget as low as possible.

1.4 Document Structure

In order to make the reading of this document easier a short description of each chapter is presented.

- **Chapter 1:** An introduction to the whole project, a description of its objectives and legal considerations involved.
- **Chapter 2:** An analysis of the current state of the art in VR control systems and the possible alternatives to use in the project.
- **Chapter 3:** The actual software and hardware architecture, design and implementation details using Unity3D and C# are described.
- **Chapter 4:** A description of all the tests performed as well as the results obtained.
- **Chapter 5:** How the project was planned in time as well as budget considerations.
- **Chapter 6:** Final conclusion words.
- **Chapter 7:** Description of ideas for future development using the acquired experience.

Chapter 2

Problem Analysis and Alternatives

With the introduction of VR we face new challenges for providing user interaction. While traditional input methods can work they are not ideal for a really immersive and comfortable experience. Some of the problems that might arise are:

- Since the user's vision is completely immersed in the virtual world, he or she is unable to see the actual input device so devices with a high number of buttons or keys can be difficult to get used to.See [15] on page 6.

- If not done carefully, the use of traditional input methods to control the virtual avatar can induce *simulator sickness* due to conflicts between the visual and bodily senses.

Although the user will develop resistance to simulator sickness just by experience, this disparity between real and virtual world can diminish the sense of immersion. See Appendix G: Simulator Sickness of [15].

However most of current technologies, with some exceptions that will be discussed later in this chapter, are focused purely on vision immersion. Although vision is the main pillar to create immersion, we should look for new control devices that will provide the user with ways to interact with the virtual environment without breaking the immersion and at the same time are intuitive and natural to use.

Since in our case the choice of the device used for vision (Oculus Rift) was constrained by the available resources, this state of the art analysis will focus on other devices that can be used for interaction.

2.1 State of the Art

In this section we focus on the analysis of tracking systems that require the use of psycho-motor skills or physical activity.

Current alternative input methods and control devices can be classified according the type of sensor technologies used. Some solutions use a mix of different sensors as it will be described see in this chapter. We will first take a look at these technologies and then review some of the most relevant devices.

2.1.1 Sensor Technologies

The main groups of sensor technologies widely used for tracking are:

Inertial Sensors

- **Accelerometer** An accelerometer measures accelerations. This is useful to measure changes in velocity (directly, as the acceleration is the first time derivative of the velocity) and changes in position (by integrating the signal). They are usually used for measuring small movements. Also note that gravity acts like a continuous acceleration upward (via Einstein's equivalence principle), so a multiple-axis accelerometer can also be used as an absolute orientation sensor in the UP-DOWN plane.

- **Gyroscope** A gyroscope measures either changes in orientation (called regular or integrating rate gyroscope) or changes in rotational velocity (rate gyroscope).

A common setup consisting of a combination of these two types of sensors (usually including a magnetometer as well) is called **Inertial Measurement Unit (IMU)** which is able to measure velocity, orientation and gravitational forces as vectors in a single device.

Magnetic Sensors

- **Magnetometer:** It is a kind of sensor that measures magnetic fields. Since the earth has a significant magnetic field magnetometers can be used as a compass to provide absolute orientation on the earth's surface. Another possible use is to have a magnetic reference different from earth's north such as an external coil that is placed in a known location [16].

Optical Sensors

Optical sensors for motion tracking typically make use of one or more RGB cameras, infrared emitters and sensors or a combination of all and rely on software algorithms to extract the relevant tracking information from image data. Some optical solutions based on cameras require external markers in the objects being tracked. These markers can be passive (reflective surfaces) or active (special light sources).

2.1.2 Overview of Commercial Devices

After researching what kind of sensors the commercial devices usually incorporate we proceeded to analyze the most popular ones.

Leap Motion

Released in 2014, the Leap Motion hardware features three infrared LEDs and two infrared cameras that provide raw sensor data to the computer. This data is then processed by the software to recreate a 3D representation of what the device sees and extract data about tracked hands or tools from it.

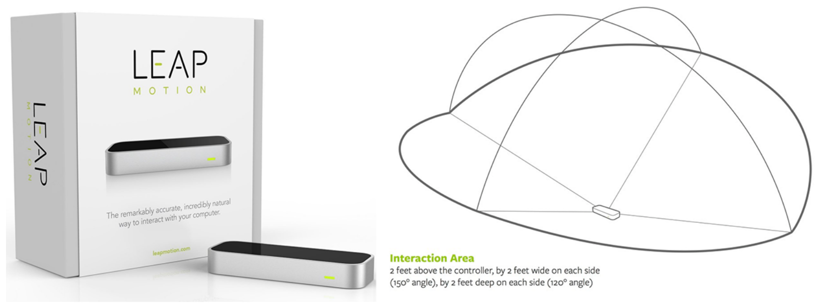

LEAP
MOTION
Interaction Area
2 feet above the controller, by 2 feet wide on each side (150° angle), by 2 feet deep on each side (120° angle)

Figure 2.1: Leap Motion and a diagram of the user interaction area.
Source: `http://amazon.com` and `http://blog.leapmotion.com`

Microsoft Kinect / Asus Xtion

The first version of Kinect was launched in 2010. It features and infrared projector which projects a grid dot pattern in the scene. The dot pattern is unique in each grid cell allowing the software to individually identify them and using a triangulation algorithm determining the depth of each one. A depth map is constructed from this information so that tracking data can be extracted from it. With this configuration Kinect allows for full body tracking of up to two users. However the heavy calculations that it has to perform and the refresh rate of the optical sensors introduces a high latency, ranging between 100ms and 500ms in the worst cases [17]. Precision is low too since certain lighting conditions have to be met like no sun light shining directly onto the scene.

The differences between Kinect and Xtion are mainly in software and drivers compatibility being Microsof's device more popular and better supported.

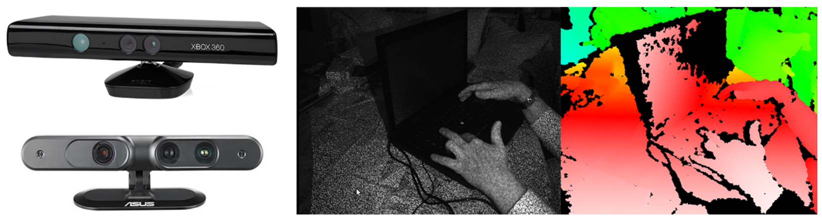


Figure 2.2: From right to left: Kinect and Xtion devices. Projected IR dot patter. Depth map constructed from the dot pattern.
Source: `http://asus.com` and `http://en.wikipedia.org/`

Wii Remote

Released in 2006, the Wii Remote features a combination of two sensor technologies. It uses a three axis accelerometer to detect forces applied to it as well as rotations in the X and Z axes. Additionally it is equipped with an infrared low resolution camera that keeps track of the position of two external infrared LEDs used as a reference. This setup allows to determine where exactly the WiiMote is pointing at with the limitation imposed by the fact that a line of sight must exist between the device and the external LEDs. In 2009 Wii MotionPlus was launched as an accessory for the Wii Remote which added two axis gyroscope to increase the accuracy and provide Y axis rotation data.

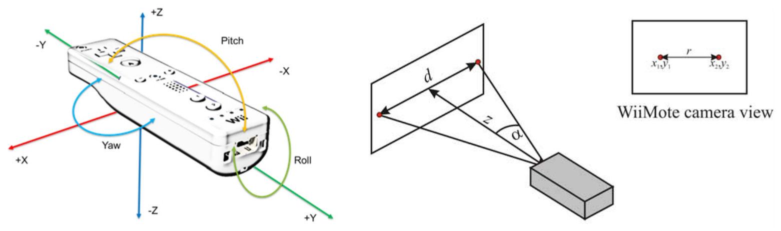


Figure 2.3: The Wii Remote coordinate system and a triangulation diagram using the external IR LEDs.
Source: `http://www.embedded.com/` and `http://wiiphysics.site88.net/`

PlayStation Move

PlatStation Move was launched in 2010 and like the Wii Remote, this device combines optical and inertial sensor technologies. First it features a glowing orb that is tracked by an external camera (PlayStation Eye) providing position data (including depth). For orientation tracking it relies on a three axis accelerometer, a two axis gyroscope and a magnetometer to help with calibration by using the earth's magnetic field as a reference.

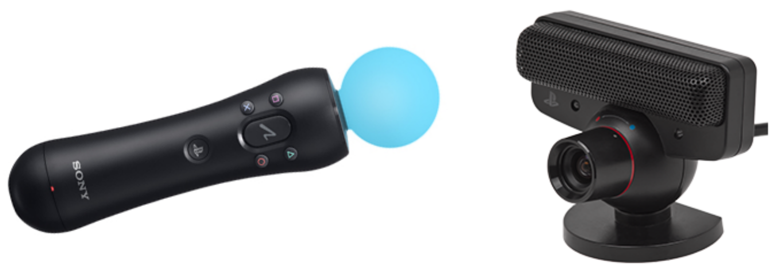


Figure 2.4: PlayStation Move on the left and Playstation Eye on the right.
Source: `http://us.playstation.com/` and `http://en.wikipedia.org/`

Razer Hydra

The Razer Hydra was released to the market in 2011 and features two controllers (one for each hand) and an external base station. This external hardware has three coils oriented along the three different axis that are powered in an alternating fashion, thus producing magnetic fields oriented differently that are then sensed by the two controllers thanks to a three axis magnetometer[18]. With this setup it is possible to determine both orientation and distance to the external base for each controller.

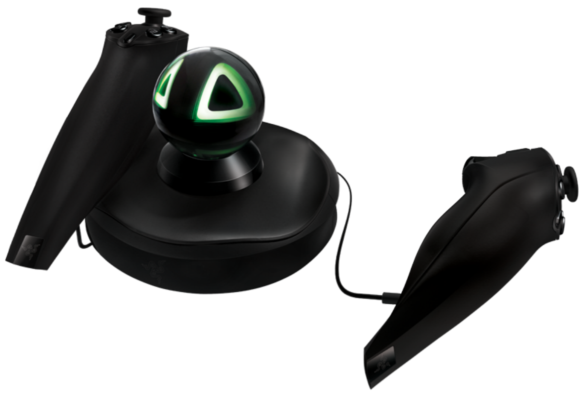


Figure 2.5: Razer Hydra controllers and external base station.
Source: `http://www.razerone.com/`

Sixense STEM System

Sixense is the company behind the magnetic technology used for Razer Hydra and they developed the STEM System as an improved version which is expected for July 2015. They both work under the same principle but the STEM reduces latency by activating all the coils at once instead of sequentially which allows for a faster sampling rate in the sensor side. Another improvement is that the coils in the base station are bigger so the range and precision of the device is extended. The mentioned latency reduction allowed Sixense to make the whole system wireless[18]. Optionally, additional sensor modules can be attached to different parts of the body.

PiroVR

Supposedly to be launched in spring 2015, PiroVR is a full body tracking solution based purely on inertial sensors (accelerometers and gyroscopes) placed in different key parts of the body. It also includes two joysticks for additional input. The system is connected via wireless to the computer, however wiring is required around the user's body.

ControlVR

Shipping to costumers of this device is expected for the second quarter of 2015 and, similarly to PiroVR, it relies on several inertial sensor placed in key body parts. However ControlVR does not provide full body tracking, instead it aims

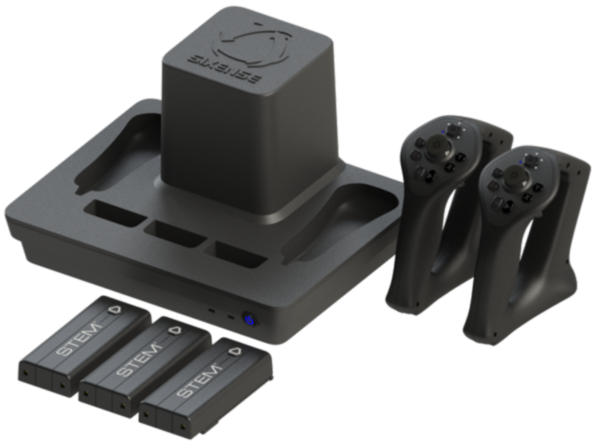


Figure 2.6: STEM controllers, external base station and three sensor modules.
Source: `http://store.sixense.com/`

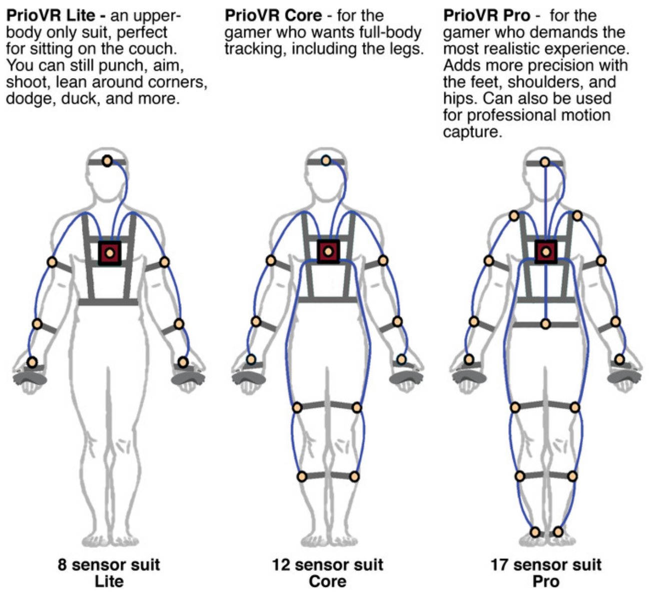

PrioVR Lite - an upper-body only suit, perfect for sitting on the couch. You can still punch, aim, shoot, lean around corners, dodge, duck, and more.
PrioVR Core - for the gamer who wants full-body tracking, including the legs.
PrioVR Pro - for the gamer who demands the most realistic experience. Adds more precision with the feet, shoulders, and hips. Can also be used for professional motion capture.
8 sensor suit Lite
12 sensor suit Core
17 sensor suit Pro

Figure 2.7: An overview of the different versions of PiroVR.
Source: `http://www.priovr.com/`

to track the user's hands and torso as well as both hands with individual fingers by using gloves equipped with additional inertial sensors.

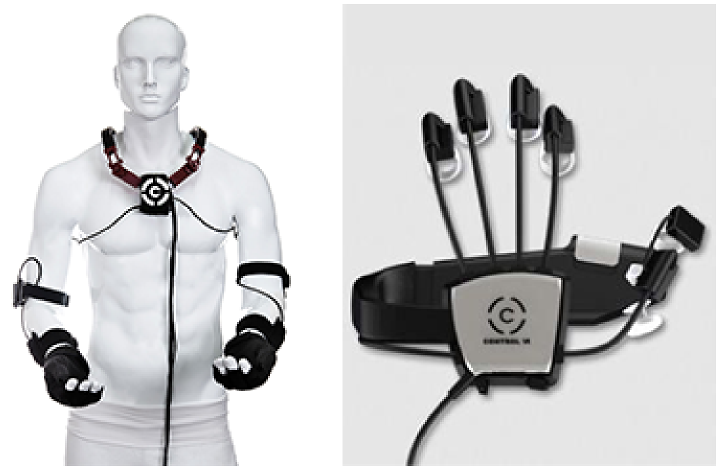


Figure 2.8: ControlVR system and a closeup of one of the gloves featuring inertial sensors.
Source: `http://controlvr.com/` and `http://vrlife.de/`

2.2 State of the Art Conclusions

In order to make the device choice easier, a summary of the state of the art conclusions drawn from the analysis is shown in the form of a comparison table in 2.1. It summarizes the features that we considered more relevant for our project needs.

After reviewing all the different technologies and devices, some of them such as the Sixense STEM, ControlVR and PiroVR had to be immediately discarded due to the high cost, low availability and potentially long setup time caused by the amount of wiring and straps they require. This lets us with Kinect as the only real option for full body tracking which was our final choice.

For our requirement of manipulating a physical object to control a virtual world element, we had now to choose between PS Move, Wii Remote or Razer Hydra. The latter has a price too high for our budget so we ruled it out. Since we already have to use an external camera (Kinect) the PS Move option, requiring an additional external camera as well, involved too many peripherals and potential malfunctions caused by interference between both of them so we left this device out of the equation as well.

At this point we started performing some tests with the Wii Remote. During these tests, and despite not being included in our previous analysis since we did not consider it an intended motion tracking device, we had the idea of trying a smartphone as the control device getting advantage of its capabilities as an . These testing and conclusion processes are explained in more detail in the testing section 4.1.

So after all this analysis and testing process, we concluded that our interaction system would consist of Kinect for full body tracking and an Android smartphone for our physical control element.

Name	**Tracking**	**Sensor Tech.**	**Availability and price**	**Pros / Cons**
Kinect / Xtion	Full body	RGB and IR cameras	In the market. 69.99€	+Low price +Easy to use −Low precision −High latency
Leap Motion	Hands and fingers	IR cameras	In the market. 89.99€	+Low price +Easy to obtain +Easy to use −Limited range
Wii Remote	One hand orientation	IR cameras	In the market. 54.19€	+Affordable +Easy to use +Low precision −External LEDs
PS Move	One hand position and orientation	RGB camera and IMU	In the market. 49.95€	+Affordable +Easy to use +Latency −External camera
Razer Hydra	Both hands position and rotation	Magnetic	In the market. From 470€	+Low latency +High precision −High price −Low range
Sixense STEM	Hands, feet and torso position and rotation	Magnetic	Q3 2015. From 266€	+Low latency +High precision +Price −Low range
PiroVR	Full body	Multiple IMU and joysticks	Spring 2015. From 260€	+Low latency +High precision +Price −Complex setup
ControlVR	Torso, arms, hands and fingers	Multiple IMUs	Q2 2015. From 535€	+Low latency +High precision +Price −Complex setup

Table 2.1: Summary of tracking devices.

2.3 System Requirements

The purpose of this section is to enumerate and describe the different functional and non-functional requirements as well as the relationship between them an the previously defined project objectives.

2.3.1 Functional Requirements

Identifier	FR-01
Name	Kinect to avatar mapping
Description	The system shall reproduce, closely and with low latency, the user movements captured by Kinect in the virtual world character.
Type	Mandatory
Related Objectives	FO-01, TO-01

Table 2.2: Functional Requirement FR-01.

Identifier	FR-02
Name	Smartphone to object mapping
Description	The system shall reproduce, closely and with low latency, the smartphone orientation changes captured in a virtual world object.
Type	Mandatory
Related Objectives	FO-02, TO-02

Table 2.3: Functional Requirement FR-02.

Identifier	FR-03
Name	Smartphone to computer link
Description	The system shall provide a way for the smartphone to send sensor data to the computer in real time and without using cables.
Type	Mandatory
Related Objectives	TO-02, TO-03, TO-04

Table 2.4: Functional Requirement FR-03.

Identifier	FR-04
Name	Oculus Rift visualization
Description	The system shall integrate the Oculus Rift device for 3D visualization.
Type	Mandatory
Related Objectives	FO-01

Table 2.5: Functional Requirement FR-04.

Identifier	FR-05
Name	Aural feedback
Description	When performing important actions or an important event occurs, the system shall play a characteristic sound to provide additional feedback on the actions of the user.
Type	Mandatory
Related Objectives	FO-01

Table 2.6: Functional Requirement FR-05.

Identifier	FR-06
Name	Choice of hand
Description	When manipulating an object is required, the system shall allow the user to choice which hand to use for this task so that the system supports left-handed and right-handed users.
Type	Mandatory
Related Objectives	TO-02

Table 2.7: Functional Requirement FR-06.

Identifier	FR-07
Name	Calibration phase
Description	The system shall provide a calibration phase for the user to calibrate the devices and adjust them to his features such as height.
Type	Mandatory
Related Objectives	TO-01, TO-02, TO-03

Table 2.8: Functional Requirement FR-07.

Identifier	FR-08
Name	Performance information
Description	At the end of a game, the system shall inform the user about his performance.
Type	Mandatory
Related Objectives	FO-01, FO-03

Table 2.9: Functional Requirement FR-08.

2.3.2 Non-functional Requirements

Identifier	NFR-01
Name	Intuitive interaction
Description	The system shall provide an intuitive and natural way of interaction for the user.
Type	Mandatory
Related Objectives	FO-01, TO-03

Table 2.10: Non-functional Requirement NFR-01.

Identifier	NFR-02
Name	Game atmosphere
Description	The game theme shall be set in a fantastic medieval environment.
Type	Mandatory
Related Objectives	FO-01, FO-02, FO-03

Table 2.11: Non-functional Requirement NFR-02.

Identifier	NFR-03
Name	Attractive visuals
Description	The system shall provide attractive visuals and a coherent art style to catch the children's attention.
Type	Mandatory
Related Objectives	FO-01, FO-02

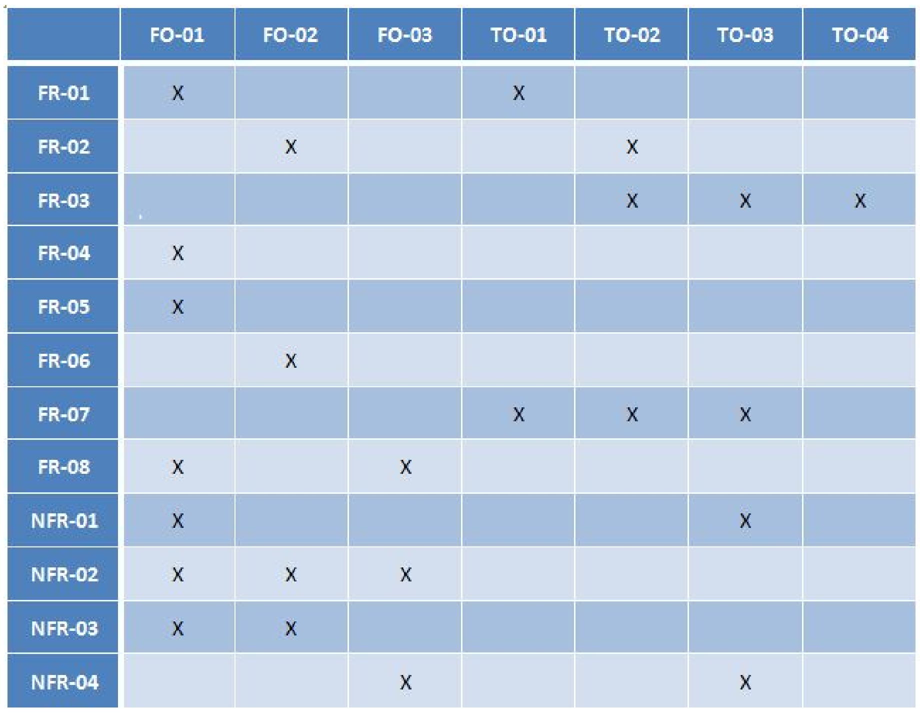
Table 2.12: Non-functional Requirement NFR-03.

Identifier	NFR-04
Name	Calibration and setup time
Description	The system setup and calibration times shall be lower than one minute per user.
Type	Mandatory
Related Objectives	FO-03, TO-03

Table 2.13: Non-functional Requirement NFR-04.

2.3.3 Objectives vs. Requirements

The following table (figure 2.9) shows the relationship between the previous requirements and the objectives set in the first chapter.



	FO-01	FO-02	FO-03	TO-01	TO-02	TO-03	TO-04
FR-01	X			X			
FR-02		X			X		
FR-03					X	X	X
FR-04	X						
FR-05	X						
FR-06		X					
FR-07				X	X	X	
FR-08	X		X				
NFR-01	X					X	
NFR-02	X	X	X				
NFR-03	X	X					
NFR-04			X			X	

Figure 2.9: Objectives vs Requirements table.

2.4 Legal and Regulation Aspects

Once the functional and technical objectives for our project have been set, it is also necessary to analyze the restrictions that apply to our system from the legal perspective.

There are two major areas in the project that require especial attention about legislation and regulation. The main one has to do with the fact that we were working with children and during the testing phase we had to record video and audio footage of them using our system. Since the image rights of children are specially protected by law, we asked the parents to sign an authorization document to avoid possible conflicts. A template of this agreement document can be found in annex D.

The second aspect concerning legal matters is about the licenses of the used artistic assets (listed in section 3.4). The used assets come from either the Unity Asset Store [19] or Open Game Art [20] website.

The Unity Asset Store provides an End User License Agreement (EULA) that grants the right to use, modify and redistribute the assets if they are bundled inside an application (section 2.2) :

> Licensor grants to the END-USER a non-exclusive, worldwide, and perpetual license to the Asset to integrate Assets only as incorporated and embedded components of electronic games and interactive media and distribute such electronic game and interactive media. Except for game services software development kits (Services SDKs), END-USERS may modify Assets. END-USER may otherwise not reproduce, distribute, sublicense, rent, lease or lend the Assets. It is emphasized that the END-USERS shall not be entitled to distribute or transfer in any way (including, without, limitation by way of sublicense) the Assets in any other way than as integrated components of electronic games and interactive media. Without limitation of the foregoing it is emphasized that END-USER shall not be entitled to share the costs related to purchasing an Asset and then let any third party that has contributed to such purchase use such Asset (forum pooling).

As for Open Game Art website, there is not a unified license, instead each individual asset can have a different license set by the author. In our case all the used assets feature a Creative Commons Zero (CC0) license which, as specified in the Creative Commons website grants all the rights to the end user [21]:

> You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission.

2.5 Tools and Technologies Used

Once we had performed the state of the art analysis and tests to choose the most appropriate technologies for interaction and the system requirements were defined, we chose our development tools. In this section we describe these different technologies (hardware and software) and tools used to develop our project.

2.5.1 Software

The choice of these specific tools is motivated mainly by the familiarity with them and relatively low (or null) cost.

Unity 3D

Unity 3D is a game engine created by Unity Technologies, it provides great multiplatform capabilities allowing to develop videogames for all major computer operative systems (Windows, OS X and Linux), videogame consoles (Xbox One and PlayStation 4) and mobile devices (Android, iOS and Windows Phone) as well as web browser thanks to the WebGL technology included in Unity 5.

All the engine features (rendering, audio, animation, physics...) can be accessed by custom scripts thanks to the Mono runtime implementation. These scripts can be written in C#, JavaScript or Boo programming languages. There is extensive and in-depth documentation for this languages and Application Programming Interface (API) features available for free in the Unity website.

The free version of the engine implements almost the same features than the Pro version with the exception of (among others) post-processing effects, deferred rendering or the ability to use native plugins.

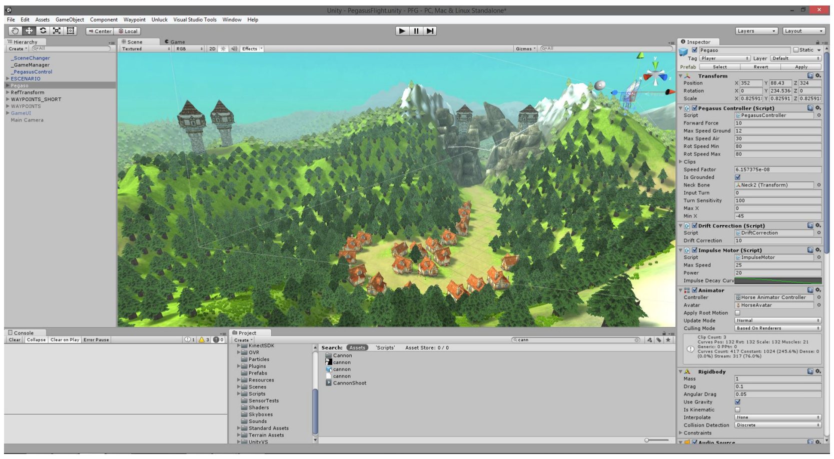

File Edit Assets GameObject Component Waypoint Unlock Visual Studio Tools Window Help
Hierarchy
Scene
Game
Inspector
Transform
Pegasus Controller (Script)
Drift Correction (Script)
Impulse Motor (Script)
Animator
Rigidbody
Console
Project

Figure 2.10: A screenshot of the Unity Editor window.

The Unity engine is in constant and quick development. In fact during the final stages of this project the version 5.1.1 was released implementing full native support for virtual reality devices such as Oculus Rift and GearVR without requiring external plugins.

For this project we used Unity Free version 4.6.4f1 as the core of our system. This game engine is responsible for integrating the visuals (3D models, animations, effects...), audio, game logic and collecting the user input.

Blender 3D

Blender 3D is an open-source and free 3D modeling and animation software. It is currently maintained by the non-profit organization Blender Foundation and it is widely used for creating animated films, visual effects and video games among others.

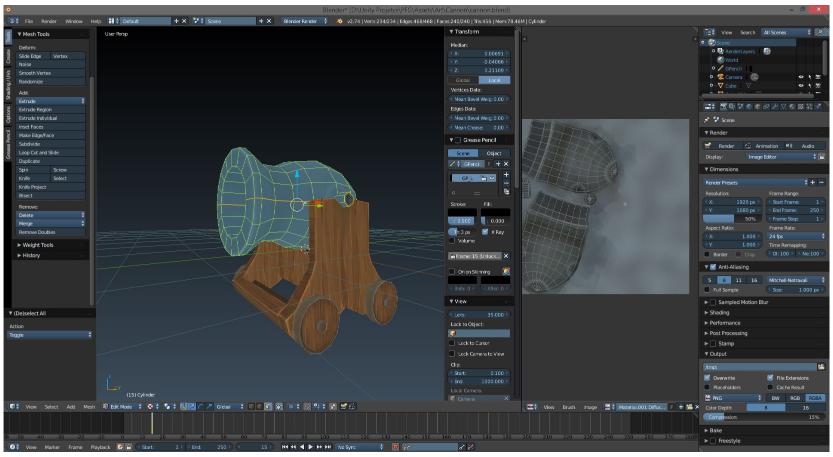


Figure 2.11: A screenshot of Blender 3D workspace.

We used Blender 2.74 throughout this project to adapt some existing 3D models, modeling and texturing some of them from scratch (such as the cannon in figure 2.11) and animating.

Photoshop

Photoshop is an image editing software developed by Adobe Systems Incorporated. Although it is mainly used for edition it can also be a great tool for creating textures, 2D animations, sprites, graphical interface elements or particles. For this project we used Adobe Photoshop CS5 mainly to create some of the game textures.

Visual Studio

Visual Studio is an Integrated Development Environment (IDE) developed by Microsoft for Windows operative system and supports a variety of languages such as C, C++, C#, Visual Basic and F#. It is a non-free software, however in 2014 Microsoft announced the Community Edition which features almost the same functionality of the paid version but for free since it is aimed for small or independent development teams. It is important to remark that Visual Studio is compatible with the Unity Engine for writing and debugging scripts.

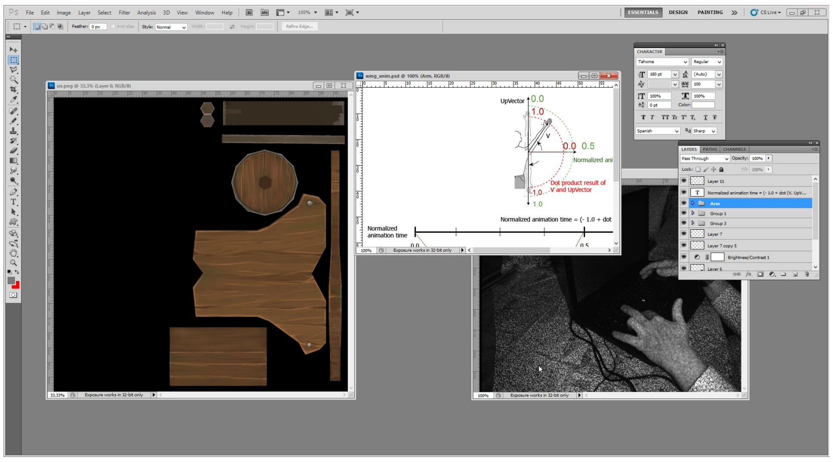
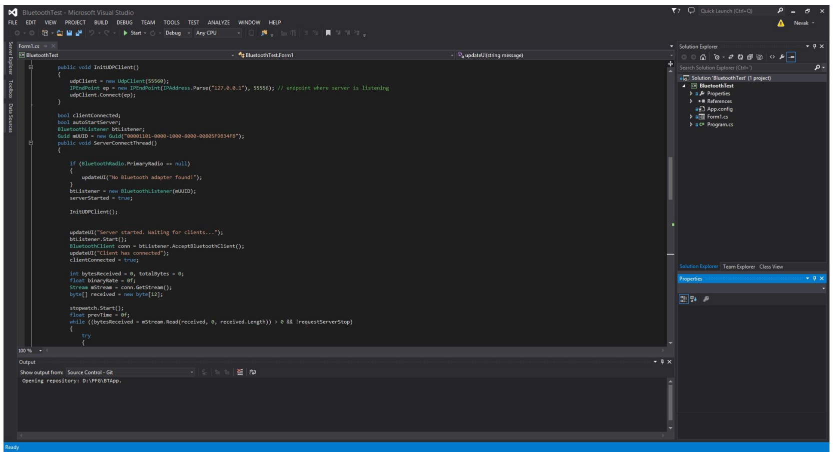


Figure 2.12: A screenshot of the Photoshop workspace.



```
public void InitUDPClient()
{
    udpClient = new UdpClient(55560);
    IPEndPoint ep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 55556); // endpoint where server is listening
    udpClient.Connect(ep);
}

bool clientConnected;
bool autoStartServer;
BluetoothListener btListener;
Guid mUUID = new Guid("00001101-0000-1000-8000-00805F9B34FB");
public void ServerConnectThread()
{

    if (BluetoothRadio.PrimaryRadio == null)
    {
        updateUI("No Bluetooth adapter found!");
    }
    btListener = new BluetoothListener(mUUID);
    serverStarted = true;

    InitUDPClient();


    updateUI("Server started. Waiting for clients...");
    btListener.Start();
    BluetoothClient conn = btListener.AcceptBluetoothClient();
    updateUI("Client has connected");
    clientConnected = true;

    int bytesReceived = 0, totalBytes = 0;
    float binaryRate = 0f;
    Stream mStream = conn.GetStream();
    byte[] received = new byte[12];

    stopwatch.Start();
    float prevTime = 0f;
    while ((bytesReceived = mStream.Read(received, 0, received.Length)) > 0 && !requestServerStop)
    {
        try
        {
```

```
Opening repository: D:\PFG\BTApp.
```

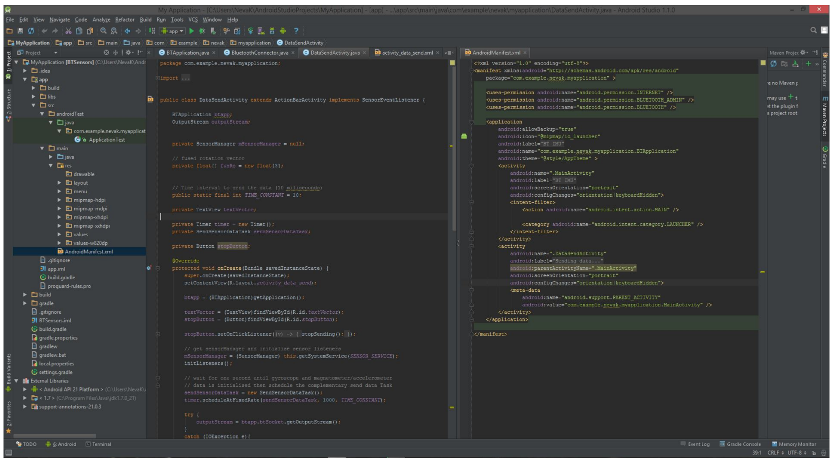
Figure 2.13: A screenshot of Visual Studio Community Edition.

The version we used for this project is Visual Studio Community Edition 2013 and was used to program all the C# scripts for the game logic as well as to develop a part of the bluetooth communication subsystem.

Android (Software) and Android Studio IDE

Developed mainly for mobile devices, Android is an operative system built on top of the Linux Kernel. It was initially developed by the company Android Inc. which was bought by Google in 2005 and finally presented to the public in 2007 by the Open Handset Alliance (a group of mobile and technology companies) as an open industry standard. Developing applications for Android is possible on any major desktop operative system (Windows, OS X or Linux). Google an provides a useful IDE called Android Studio and extensive API documentation for free.

The target platform for this project was Android 4.3 and we used Android



```java
package com.example.nevak.myapplication;

import ...


public class DataSendActivity extends ActionBarActivity implements SensorEventListener {

    BTApplication btapp;
    OutputStream outputStream;


    private SensorManager mSensorManager = null;

    // fused rotation vector
    private float[] fusRo = new float[3];


    // Time interval to send the data (10 miliseconds)
    public static final int TIME_CONSTANT = 10;

    private TextView textVector;

    private Timer timer = new Timer();
    private SendSensorDataTask sendSensorDataTask;

    private Button stopButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_data_send);

        btapp = (BTApplication)getApplication();

        textVector = (TextView)findViewById(R.id.textVector);
        stopButton = (Button)findViewById(R.id.stopButton);

        stopButton.setOnClickListener((v) -> { stopSending(); });

        // get sensorManager and initialise sensor listeners
        mSensorManager = (SensorManager) this.getSystemService(SENSOR_SERVICE);
        initListeners();

        // wait for one second until gyroscope and magnetometer/accelerometer
        // data is initialised then schedule the complementary send data Task
        sendSensorDataTask = new SendSensorDataTask();
        timer.scheduleAtFixedRate(sendSensorDataTask, 1000, TIME_CONSTANT);

        try {
            outputStream = btapp.btSocket.getOutputStream();
        }
        catch (IOException e){
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.nevak.myapplication" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BLUETOOTH" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="BT IMU"
        android:name="com.example.nevak.myapplication.BTApplication"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="BT IMU"
            android:screenOrientation="portrait"
            android:configChanges="orientation|keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".DataSendActivity"
            android:label="Sending data..."
            android:parentActivityName=".MainActivity"
            android:screenOrientation="portrait"
            android:configChanges="orientation|keyboardHidden">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.nevak.myapplication.MainActivity" />
        </activity>
    </application>

</manifest>
```

Figure 2.14: A screenshot of Android Studio IDE.

Studio version 1.1.0 which was used to develop the Android application involved in the bluetooth subsystem.

2.5.2 Hardware

The choice of these devices is a result of the state of the art conclusions, with the exception of the Oculus Rift which was used for being the best VR helmet we had available.

Oculus Rift

The Oculus Rift is a Virtual Reality (VR) helmet currently being designed by Oculus VR. A consumer version has not been released to the market yet, instead two development kits are available (DK1 and DK2) intended to be used for prototyping. The final version for consumers is expected to be released in 2016. It works by sending slightly different images to each user's eye creating a 3D effect and also tracks the head orientation by using an Inertial Measurement Unit (IMU) (with accelerometer, magnetometer and gyroscope) to move the virtual camera accordingly. These two features can be easily integrated inside Unity using the package provided by Oculus which includes a plugin to interact with the Software Development Kit (SDK) as well as prefabs with a dual camera setup ready to use.

Some technical details of the software and device used are shown next:

- Unity 4 Integration package version: 0.4.4-beta [22]
- Oculus SDK version: 0.4.4-beta
- Oculus Rift DK1 (fig. 2.15)

We used the Oculus Rift as our display and head orientation tracking device.

Figure 2.15: The Oculus Rift DK1

Kinect Sensor

The Kinect sensor captures color and depth information and extracts the user skeletal data from it so that it can be retrieved trough the Microsoft Kinect SDK in the application side. This SDK can be used directly, however to simplify the process an external Unity package was used that, among other features, provides a framework to add custom gesture recognition and ready-to-use scripts that map skeleton data to virtual character models.

Here are some technical details of the software and device used:

- Kinect with MS-SDK Unity package version: 1.12 [23]
- Microsoft Kinect SDK version 1.8
- Kinect Sensor for Xbox 360

In our project the purpose of this device is to provide full body tracking and mapping the user movements to a virtual character.

Android Smartphone (Hardware) as an IMU

Most of current smartphones incorporate sensor fusion capabilities to determine the device orientation by combining accelerometer, magnetometer and gyroscope data so one of this devices could be used as an Inertial Measurement Unit (IMU). This feature added to the fact that smartphones are very easily available today makes them a good candidate for a control device. For our project we used a Sony Xperia Z2 phone as a physical object that controls elements inside the virtual world.

Chapter 3

Design and Implementation

This chapter first gives an overview of the different parts of the system and how they interact with each other, then the three games that were created on top of this system as well as the Unity scripts developed for this purpose are explained in more detail.

3.1 System Architecture and Specifications

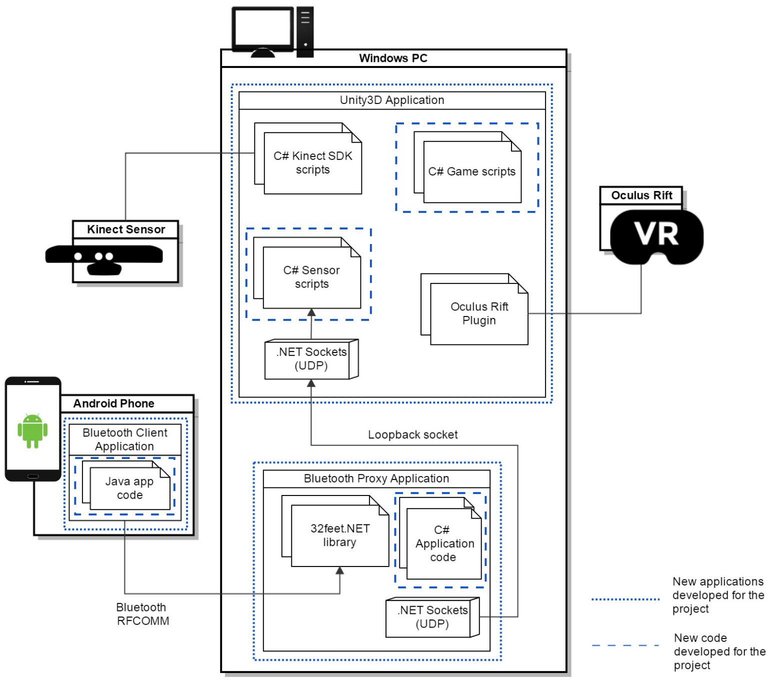
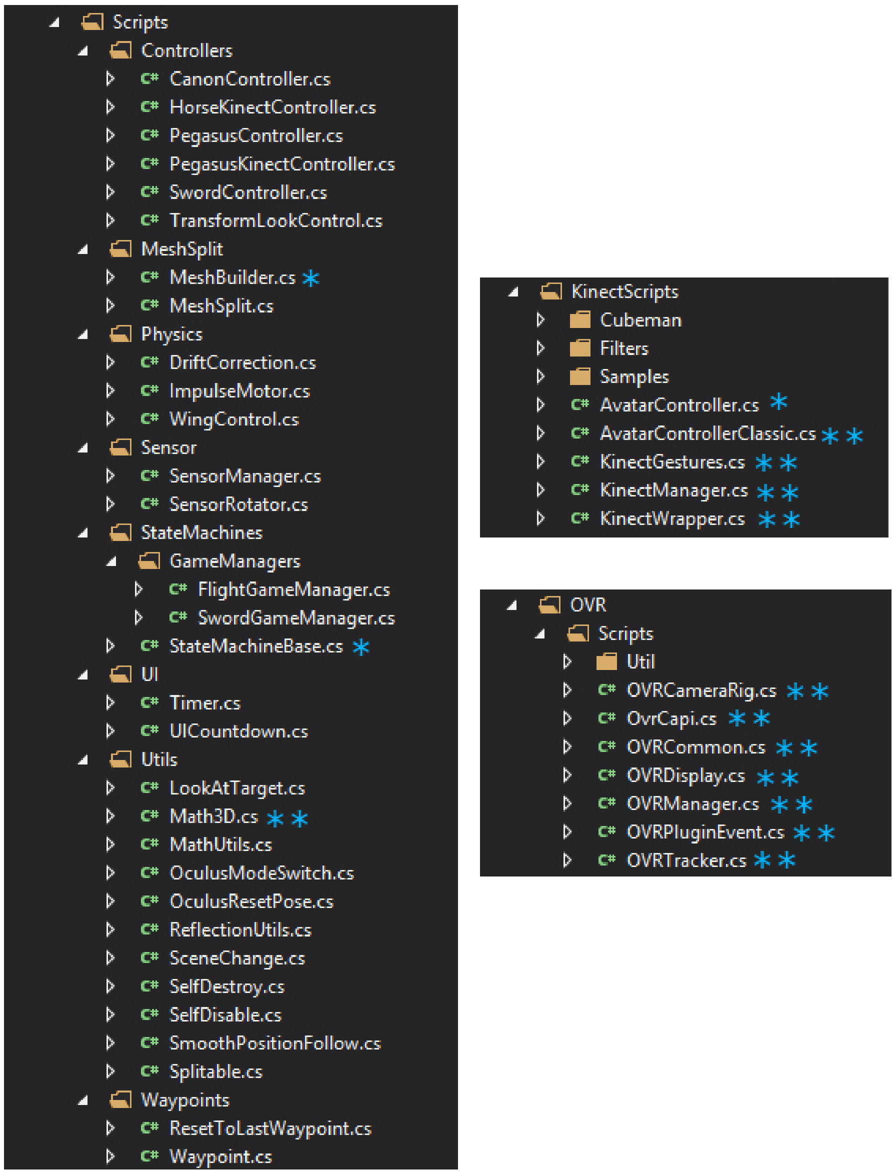


Figure 3.1: Hardware and software system architecture overview.



```
Scripts
  Controllers
    CanonController.cs
    HorseKinectController.cs
    PegasusController.cs
    PegasusKinectController.cs
    SwordController.cs
    TransformLookControl.cs
  MeshSplit
    MeshBuilder.cs *
    MeshSplit.cs
  Physics
    DriftCorrection.cs
    ImpulseMotor.cs
    WingControl.cs
  Sensor
    SensorManager.cs
    SensorRotator.cs
  StateMachines
    GameManagers
      FlightGameManager.cs
      SwordGameManager.cs
    StateMachineBase.cs *
  UI
    Timer.cs
    UICountdown.cs
  Utils
    LookAtTarget.cs
    Math3D.cs * *
    MathUtils.cs
    OculusModeSwitch.cs
    OculusResetPose.cs
    ReflectionUtils.cs
    SceneChange.cs
    SelfDestroy.cs
    SelfDisable.cs
    SmoothPositionFollow.cs
    Splitable.cs
  Waypoints
    ResetToLastWaypoint.cs
    Waypoint.cs
```

```
KinectScripts
  Cubeman
  Filters
  Samples
  AvatarController.cs *
  AvatarControllerClassic.cs * *
  KinectGestures.cs * *
  KinectManager.cs * *
  KinectWrapper.cs * *
```

```
OVR
  Scripts
    Util
    OVRCameraRig.cs * *
    OvrCapi.cs * *
    OVRCommon.cs * *
    OVRDisplay.cs * *
    OVRManager.cs * *
    OVRPluginEvent.cs * *
    OVRTracker.cs * *
```

Figure 3.2: An overview of the Unity scripts of the system:
The scripts not marked with asterisks were developed from scratch for the project.
* Indicates that the script is a modified version of an external one.
** Indicates that the script is completely external and used without any modifications.

3.2 Bluetooth Subsystem Development

To establish a wireless communication between the smartphone and the computer we developed two applications from scratch. One of them will be installed on the Android phone and will act as a client sending the data via bluetooth. The other one will be a Windows application on the computer that will play a double role: act as bluetooth server listening for incoming connections and redirecting the received data via UDP to the specified port where the game will be listening.

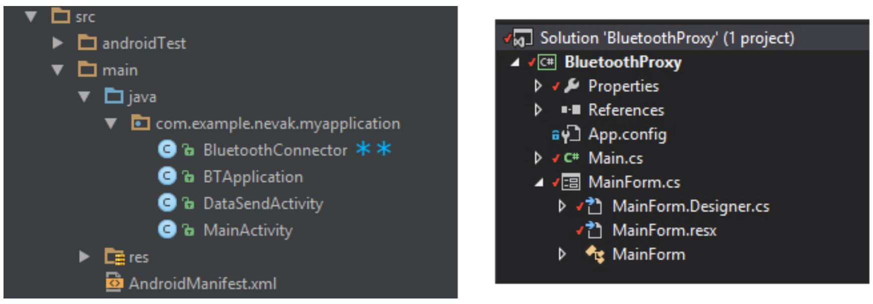


Figure 3.3: An overview of the code files involved in the development of the bluetooth subsystem. The Android project Java files are shown on the left and the Windows .NET C# files on the right:
The elements not marked with asterisks were developed from scratch for the project.
* Indicates that the element is a modified version of an external one.
** Indicates that the element is completely external and used without any modifications.

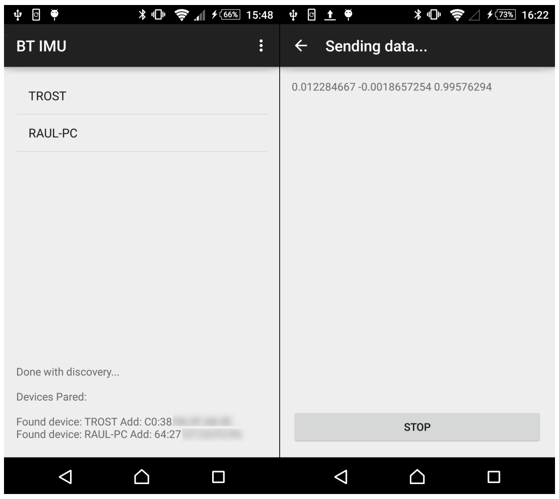
3.2.1 Development of the Bluetooth Client Application

For our scenario a custom made app (we called BT IMU) (figure 3.4) retrieves the phone sensor data and sends it via bluetooth to the computer so that it can be used inside Unity to control virtual objects. The app lists all paired devices and lets the user select one to connect to. Once the connection is successful it will start sending sensor data obtained from Sensor.TYPE_ROTATION_VECTOR [24]. Since this feature is only available from Android API level 18 onwards, the application **requires Android version 4.3 or later**.

The choice of bluetooth is not arbitrary. A few quick tests were performed sending the sensor data via WiFi and showed unacceptable results for real time interaction due to high latency spikes and packet loss. These tests were done using a third party app called Wireless IMU [25].(See section 4.1).

3.2.2 Development of the Bluetooth Proxy Application

The purpose of this program that runs on the Windows PC is to handle bluetooth pairing with the smartphone and receive the orientation data. This information is then forwarded via a UDP loopback socket to the specified port on



TROST

RAUL-PC

Done with discovery...

Devices Pared:

Found device: TROST Add: C0:38[redacted]
Found device: RAUL-PC Add: 64:27[redacted]

0.012284667 -0.0018657254 0.99576294

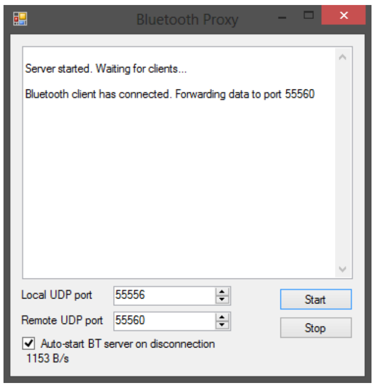
Figure 3.4: A screenshot of the BT IMU Android app we created to send orientation data over bluetooth. On the left a list of paired devices is displayed. On the right the rotation vector data being sent.

which the Unity game application is listening. To implement bluetooth functionality the 32feet library [26] for .NET was used. Ideally this library could be used directly inside the Unity's C# scripts so that it connects with the smartphone without the need for the UDP loopback hassle. The problem with this approach is that Unity uses Mono [27] internally instead of .NET to provide multi platform capabilities and 32feet library is currently not compatible with Mono 2.0 since it doesn't allow the creation of bluetooth sockets. To illustrate the problem, the code in listing 3.1 will work for a .NET 3.5 application, however it will fail if used inside Unity with Mono 2.0:

Listing 3.1: C# code fragment to create a bluetooth socket

```
using System.Net.Sockets;
[...]
//AddressFamily 32 indicates a Bluetooth address and
    ProtocolType 3 is for RFCOMM
Socket s = new Socket((AddressFamily)32, SocketType.Stream, (
    ProtocolType)3);
[...]
```

Another possible solution would be to develop a Unity plugin in C++ that handles bluetooth communication and provides wrapper methods to C# code, however this option requires a Unity Pro license [28].



Server started. Waiting for clients...

Bluetooth client has connected. Forwarding data to port 55560

Local UDP port 55556

Remote UDP port 55560

[✓] Auto-start BT server on disconnection

1153 B/s

Figure 3.5: A screenshot of the Bluetooth Proxy Application we developed for Windows. It shows current connection status and bitrate and allows to configure remote and local UDP ports.

3.3 Game Development

The core of this project consists of three small games that follow a similar pattern. In all of them there is a specific task that has to be completed by performing physical activities and the game will reward the player according to how well he performed. There is also an initial calibration phase in each game that helps the player to familiarize with the VR headset and controls.

The three games that will be described in detail in this section are:

- Sword Training: Implies manipulating a physical object (a smartphone) that is represented in the virtual world as a sword.
- Pegasus Flight: Involves arm and head motion to control a winged horse flying.
- Pegasus Ride: Requires the player to move his arms and torso to control a character riding a horse.

3.3.1 Sword Training

What follows is a description of the design for the different aspects of the game: what the game mission and goals are, how the player interacts with it and the flow sequence of the game states. At the end of this section we describe the relevant scripts we developed to materialize this design inside Unity 3D.

Game Mission Background

You are Edilmar, the elven warrior and protector of the Kingdom of Alistea. The kingdom is at peace now, but not for too long so we better get you training with the sword. I will start shooting some watermelons with the magic cannon and you will have to slash them in mid air. The more efficient you are at it, the better you will be rewarded.

In this game the user is strongly encouraged to accomplish the objectives since there is not freedom of movement around the scenery so he has no choice but slashing the fruits that come at him.

Game Goal

The game will end when the player manages to slash a certain (configurable) number watermelons, however the more efficient he is the better the reward will be. The degree of efficiency is measured by the ratio of watermelons cut and total watermelons thrown. So the rewards are as follows:

- **Gold medal:** Efficiency greater than 65%.
- **Silver medal:** Efficiency between 30% and 65%.
- **Bronze medal:** Less than 30% efficiency.

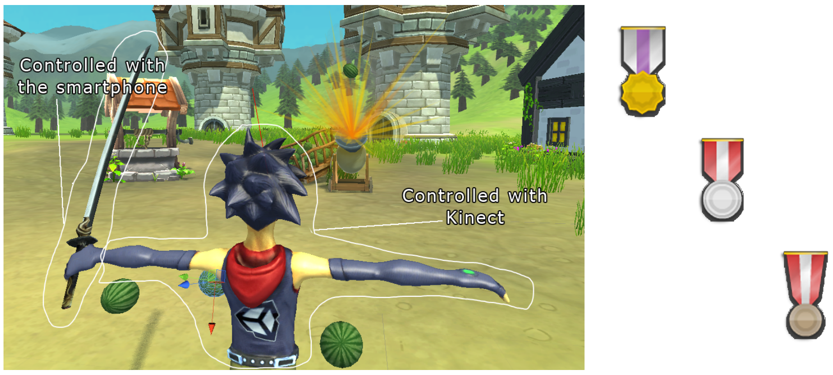


Figure 3.6: A screenshot from Sword Training scene showing the control mapping (left) and the possible awarded medals (right).

Interaction Design

In this game the player is required to perform accurate movements to hit a small object. Since the avatar is a humanoid and the camera is completely in first person mode, the user can see his virtual hands and body responding to his movements. This creates a high sense of embodiment.

The avatar is controlled with Kinect for body tracking (special interest is in the arms). The player also holds a smartphone in his hand as if he was wielding a sword. The device orientation sensors are used to track the player's hand rotations so that the sword in game accurately follows his movements. The use of a physical object or prop as an input method also provides some benefits [29]:

- Familiarity
- Direct actions
- Obvious use
- Palpability
- No tool moding (a single interface doesn't have different "modes" for different operations, thus each physical object or prop has exactly one function)
- Haptic feedback

Another feature that was crucial to implement was the option to select between left-handed or right-handed mode. Some considerations had to be taken when mapping the phone orientation to the virtual sword depending on which hand was chosen since the coordinate axis of the avatar's hands have a different initial orientation but we wanted the player to hold the phone in the same way, independently of the hand chosen.

The aural feedback for this game is present when the cannon shoots, when a fruit is sliced successfully and when the player waves the sword above a set minimum speed.

Finally it is important to remark that to create a more visually attractive experience, the fruits are sliced exactly at the position and angle that the sword has at the moment of the impact.

Game States

The game has different states that follow sequentially:

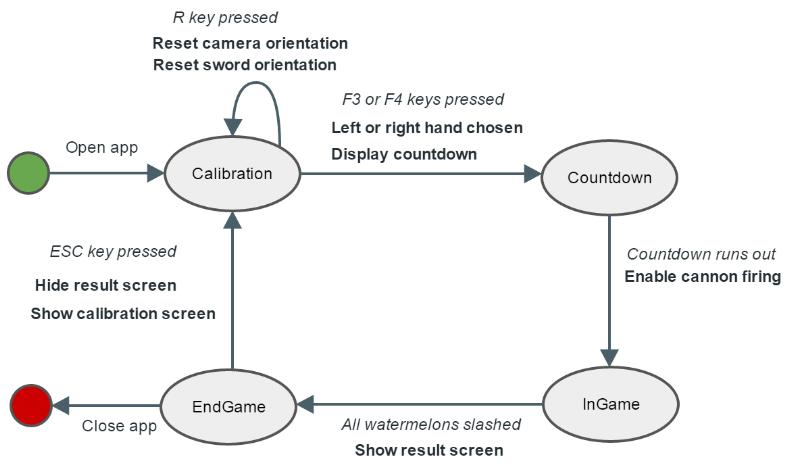


Figure 3.7: Global game state diagram for Sword Training

- **Calibration:** Some simple instructions for calibration are displayed. Such as placing the smartphone on a flat surface and pointing towards the Kinect sensor to set the initial rotations correctly. The player is also given the choice to use left or right hand to play.
- **Countdown:** A brief countdown starts to give some time for the player to get ready.
- **InGame:** The actual game takes place and the cannon begins to shoot watermelons at the player. It will end when the player slashes a certain number of watermelons that can be configured beforehand.
- **EndGame:** A summary of how good the player performed is displayed. He will be awarded with a medal according to how good he performed.

Scene Layout and Organization in Unity

The scene is comprised of four main GameObjects (a naming convention to denote singleton objects with an underscore as a prefix is used throughout this section):

- Character: A humanoid character that is controlled via AvatarController script. This script is included in [23] and maps the user's joints movements to the virtual character. The character's hand GameObject (right or left depending on what the player selects) has a SensorRotator script attached to control its rotation through a smartphone as explained in the next section. The Sword GameObject is parented to the hand and has a SwordController script that allows interaction with the environment.

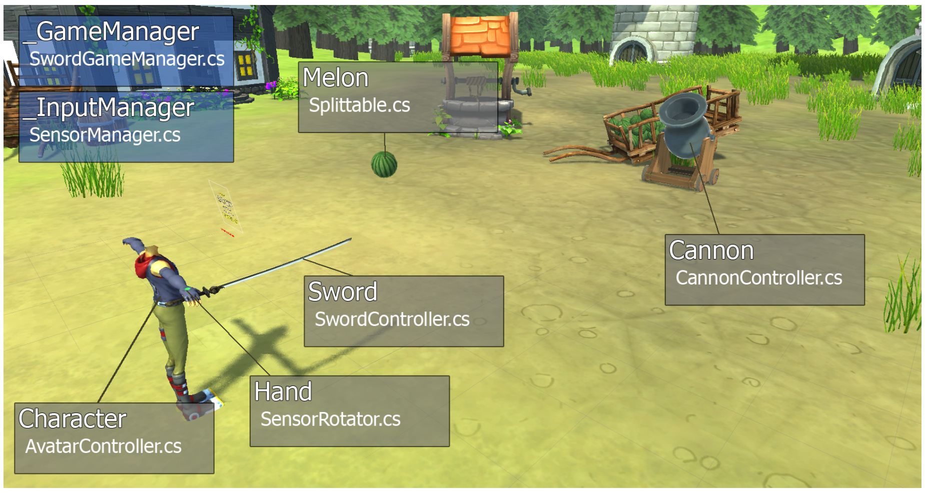


Figure 3.8: Sword Training scene showing the relevant GameObjects and attached scripts.

- Cannon: Controlled by CannonController it spawns Melon GameObject prefabs when shooting.
- _GameManager: It has a SwordGameManager script attached to control the game state.
- _InputManager: Acts as a centralized way to access user input. It has a SensorManager script attached which is explained in the next section and a KinectManager which is included in [23] and is in charge of abstracting the Kinect SDK functionality so that the AvatarController works properly.

Script Description

In this subsection we describe the most relevant scripts that we developed for the game:

SwordGameManager.cs
Implemented as a state machine, it controls the game flow and states described in subsection 3.3.1 in addition to some other functions:

- Listens to user input events such as the key pressed to start playing.
- Enables and disables cannon firing.
- Listens to game events such as when a watermelon is cut by the player and shot by the cannon and keeps a count of both to compute final score. The method used for event subscription is similar to the one described in listing 3.5 for the way point system.
- Controls all the GUI elements.

Just like FlightGameManager described in section 3.3.2, this class inherits from the StateMachineBase script by Mike Talbot [30].

SensorManager.cs
It is in charge of listening on the specified UDP port for incoming orientation data. This data comes from the Bluetooth Proxy Application (see section 3.2.2) which in turn receives sensor data from a connected smartphone via bluetooth. The data comes without processing from the device as provided by the Android API by using the "Rotation Vector" sensor type [24] (see section 3.2.1 as well).

This class exposes the smartphone orientation as quaternion that is calculated from the received vector data. It also performs an axis conversion so that Android axis convention matches Unity's coordinate system. Here is fragment of the code that performs these operations:

Listing 3.2: Code fragment for quaternion calculation

```
rawBytes = udpClient.EndReceive(res, ref groupEP);

// Sensor data correspoinding to TYPE_ROTATION_VECTOR
Vector3 fusro = new Vector3();
fusro.x = System.BitConverter.ToSingle(rawBytes, 0);
fusro.y = System.BitConverter.ToSingle(rawBytes, 4);
fusro.z = System.BitConverter.ToSingle(rawBytes, 8);

/*
Process the received orientation vector to convert it in a
    quaternion
See android API documentation
http://developer.android.com/reference/android/hardware/
    SensorEvent.html
*/
if(!float.IsNaN(fusro.x) &&
   !float.IsNaN(fusro.y) &&
   !float.IsNaN(fusro.z))
{
  mNativeFusedQuaternion.x = fusro.x;
```

```
        // X and Z axes are swapped to match Unity's coordinate
            system
        mNativeFusedQuaternion.y = fusro.z;
        mNativeFusedQuaternion.z = fusro.y;
        mNativeFusedQuaternion.w = Mathf.Sqrt(1.0f - fusro.
            sqrMagnitude);
    }
```

First the data received from the UDP socket is stored in a Vector3 object and checked to see if it is valid. This rotation vector $\tilde{\mathbf{r}}$ represents the orientation of the device as a combination of an angle and an axis, in which the device has rotated through an angle θ such that:

$$\tilde{\mathbf{r}} = \langle x\sin(\theta/2), y\sin(\theta/2), z\sin(\theta/2)\rangle$$
$$|\tilde{\mathbf{r}}| = \sin(\theta/2)$$

And the direction of $\tilde{\mathbf{r}}$ is equal to the direction of the axis of rotation. So it can be converted into a quaternion $\tilde{\mathbf{q}}$ as follows:

$$\tilde{\mathbf{q}} = \langle x\sin(\theta/2), z\sin(\theta/2), y\sin(\theta/2), \cos(\theta/2)\rangle =$$
$$= \langle \tilde{\mathbf{r_x}}, \tilde{\mathbf{r_z}}, \tilde{\mathbf{r_y}}, \sqrt{1-|\tilde{\mathbf{r}}|}\rangle$$

Notice the swap performed to the x and y components. This is done due to the fact that Android considers the Z axis as world's up direction, while Unity uses Y axis pointing upwards [24].

SensorRotator.cs
This script is used to actually rotate the GameObject it is attached to. The SensorManager class reports an absolute orientation (it ultimately relies on earth's magnetic north and gravity) however it would be unpractical to ask the player to always use the smartphone pointing north in the real world. That's why relative orientation is used i.e. we only are interested in orientation changes that we apply to the GameObject, a character's hand joint that is wielding a sword in the case of this game. Some of the concepts used to write this script were taken from [31].

In addition, the script provides an option to swap axes for the case when a player is left handed. This is made to account for the fact that the hand joints of used character model have the X and Z axis pointing in the opposite direction one from the other.

SmoothPositionFollow.cs
Smoothly interpolates the position of the GameObject towards the target specified. In this case it is used to make the VR camera rig follow the avatar's head position in a smooth way since directly parenting it to the character's head joint would cause jerky or sudden camera movements that can be uncomfortable for the player.

OculusResetPose.cs
A simple script that will re-center the VR camera when the specified key is pressed. It's used to calibrate the VR headset before the game starts.

CanonController.cs
Controls the cannon shooting, sound and visual effects (particle system and recoil animation). The firing rate, range and accuracy can be easily tweaked as well as the projectile prefab. The sound, apart from helping with immersion, helps the player to match the timing of the sword swings with the cannon firing rate.

MeshBuilder.cs
Initially taken from a script by Jens-Kristian Nielsen [32]. Contains useful functions to ease mesh construction out of vertex data. It has been modified to include a Triangle class that allows individual triangle splitting:

Listing 3.3: Fragment of the custom added code to MeshBuilder.cs to split triangles

```
/// <summary>
/// Cuts a triangle trough a given plane.
/// </summary>
/// <param name="planeNormal">The normal of the plane used to cut
    </param>
/// <param name="planePosition">A point lying on the cutting plane
    </param>
/// <param name="up">The mesh used to store the triangle or
    triangles (if any) that will be formed above the plane after
    the cut</param>
/// <param name="down">The mesh used to store the triangle or
    triangles (if any) that will be formed beneath the plane after
     the cut</param>
/// <returns>The vertices that lie on the intersection with the
    plane if any</returns>
public Vector3[] Split(Vector3 planeNormal, Vector3 planePosition,
     MeshBuilder up, MeshBuilder down)
{
      bool[] vUp = { false, false, false };

      // We determine if each vertex is above or beneath the plane
           and then consider all the combinations
      vUp[0] = IsAbovePlane(vertices[0], planeNormal,
          planePosition);
      vUp[1] = IsAbovePlane(vertices[1], planeNormal,
          planePosition);
      vUp[2] = IsAbovePlane(vertices[2], planeNormal,
          planePosition);

      int k1, k2;

      // If all three vertices are above the plane we just add the
           triangle to the upper half mesh since there's no
          intersection
      if (vUp[0] && vUp[1] && vUp[2])
      {
        up.AddTriangle(this);
        }
        // If all three vertices are beneath the plane we just add
             the triangle to the lower half mesh since there's no
            intersection
        else if (!vUp[0] && !vUp[1] && !vUp[2])
        {
          down.AddTriangle(this);
```

```
}
// In this case the plane actually intersects the
    triangle so we have stuff to do
else
{
  // We use this loop to iterate over the vUp boolean
      array and consider every case of vertices being
      above or beneath the plane
  // This is done to make the code more compact instead
      of having 4 possible if / else's
  for (int k = 0; k < 3; k++)
  {
    k1 = (k + 1) % 3;
    k2 = (k + 2) % 3;

    // New vertices created at the plane cut
    Vector3 nV0 = Intersec(vertices[k], vertices[k1],
        planeNormal, planePosition);
    Vector3 nV1 = Intersec(vertices[k], vertices[k2],
        planeNormal, planePosition);
    // New normals for the new vertices
    Vector3 nN0 = (normals[k] + normals[k1]) / 2f;
    Vector3 nN1 = (normals[k] + normals[k2]) / 2f;
    // New UV texture coordinates for the new vertices
    Vector2 nUV0 = UVLerp(uvs[k], uvs[k1], vertices[k],
        vertices[k1], nV0);
    Vector2 nUV1 = UVLerp(uvs[k], uvs[k2], vertices[k],
        vertices[k2], nV1);


    // If one vertex is above the plane and two are
        beneath it
    if (vUp[k] && !vUp[k1] && !vUp[k2])
    {
      // We add one new triangle to the upper half mesh
      // This triangle is formed by the original vertex
          above the plane and two new vertices at the
          cut plane position (nV0 and nV1 that we
          calculated before).
      up.AddTriangle(new Vector3[] { nV0, nV1, vertices[
          k] }, new Vector3[] { nN0, nN1, normals[k] },
          new Vector2[] { nUV0, nUV1, uvs[k] });

      // For the lower part two new triangles will be
          formed since there are two vertices beneath
          the plane and two vertices at the plane cut.
      // We first form a triangle with the vertices
          beneath the plane (k1 and k2) and one of the
          vertices at the plane cut (nV0).
      // We also store the indices of two of these new
          added vertices to reuse them for the other
          triangle.
      int i0 = down.AddVertex(nV0, nN0, nUV0);
      down.AddVertex(vertices[k1], normals[k1], uvs[k1])
          ;
      int i2 = down.AddVertex(vertices[k2], normals[k2],
           uvs[k2]);

      // For the second triangle we just reuse two of
          the added vertices by referencing them with
          the stored indices (i0 and i2)
      down.Triangles.Add(i0);
```

```
                down.Triangles.Add(i2);
                // And finally add the third vertex which is the
                    other vertex at the plane cut.
                down.AddVertex(nV1, nN1, nUV1);

                // We return the newly created vertices at the
                    plane cut so that they can be used to create
                    new geometry, for example to fill the gap.
                return new Vector3[]{nV0, nV1};
                }
                // If one vertex is beneath the plane and two are
                    above it
                else if (!vUp[k] && vUp[k1] && vUp[k2])
                {
                  // We add one new triangle to the lower half
                      mesh
                  // This triangle is formed by the original
                      vertex beneath the plane and two new
                      vertices at the cut plane position (nV0 and
                      nV1 that we calculated before).
                  down.AddTriangle(new Vector3[] { nV0, nV1,
                      vertices[k] }, new Vector3[] { nN0, nN1,
                      normals[k] }, new Vector2[] { nUV0, nUV1,
                      uvs[k] });


                  // For the upper part two new triangles will be
                      formed since there are two vertices above
                      the plane and two vertices at the plane cut.
                  // We first form a triangle with the vertices
                      above the plane (k1 and k2) and one of the
                      vertices at the plane cut (nV0).
                  // We also store the indices of two of these new
                       added vertices to reuse them for the other
                      triangle.
                  int i0 = up.AddVertex(nV0, nN0, nUV0);
                  up.AddVertex(vertices[k1], normals[k1], uvs[k1])
                      ;
                  int i2 = up.AddVertex(vertices[k2], normals[k2],
                       uvs[k2]);

                  // For the second triangle we just reuse two of
                      the added vertices by referencing them with
                      the stored indices (i0 and i2)
                  up.Triangles.Add(i0);
                  up.Triangles.Add(i2);
                  // And finally add the third vertex which is the
                       other vertex at the plane cut.
                  up.AddVertex(nV1, nN1, nUV1);

                  // We return the newly created vertices at the
                      plane cut so that they can be used to create
                       new geometry, for example to fill the gap.
                  return new Vector3[]{nV0, nV1};
                }
          }
        }
      return new Vector3[]{};
  }
```

More in depth information about triangular mesh splitting can be found in [33].

MeshSplit.cs
This class does the actual GameObject mesh splitting. It relies on the previously described MeshBuilder class and it is in charge of setting up the two new GameObjects that are produced after splitting (adding a MeshCollider and configuring the MeshRenderer material). The newly created pieces however have a hole that is not filled with new geometry. Since this could not be implemented due to time and complexity constrains, a simple solution using a custom shader to paint back faces with a solid color in a second pass was used. See figure 3.9. The code for this custom shader can be seen in listing 3.4.

Listing 3.4: Custom colored back faces shader

```
Shader "Custom/ColorBackfaces" {
  Properties {
    _Color ("Main Color", Color) = (1,1,1,1)
    _BfColor ("Backface Color", Color) = (1,0,0,1)
    _MainTex ("Base (RGB)", 2D) = "white" {}
  }
  SubShader {
    // Extra pass that renders only backfaces
    Pass {
      Cull Front
      Color [_BfColor]
    }

    // Use forward rendering passe from Diffuse shader
    UsePass "Diffuse/FORWARD"
  }
  Fallback "Diffuse/VertexLit"
}
```

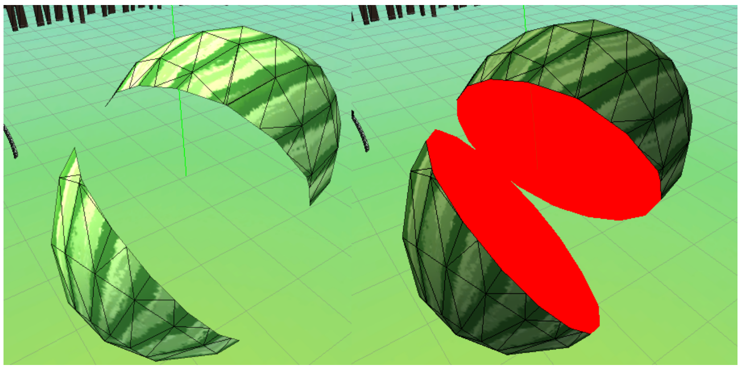


Figure 3.9: On the left the geometry of a split watermelon with culled back faces. On the right the same geometry but with a shader that renders back faces with a solid red color.

Splittable.cs
This component script is used to identify the GameObjects that are actually splittable and keeps track of the pieces after a split to delete them after a few seconds to save memory.

SwordController.cs
This scripts is attached to the sword GameObject and relies on a trigger collider. Its function is to fire events when a GameObject with a Splitable component attached is hit (watermelons in this case). These events are listened by the SwordGameManager class to account for score. When a collision with a Splitable GameObject occurs, its Split method is called passing the required information to it (specially the cut direction) so that a proper splitting can be made. Every time an object is cut the script plays a characteristic sound to provide aural feedback so that the user can know if he hit the target or not without actually having to look at the object.

3.3.2 Pegasus Flight

In this section we will describe the different aspects of the game: a brief background of the mission, its goal, the way the player interacts with the world and the game states. Finally a description of the most relevant C# scripts developed to make the design possible is presented.

Game Mission Background

You are Compay this time, you will notice that you have no arms but wings! That is because you are a winged horse now, a Pegasus. Your job is to take our visitors that come to Alistea from the magic entry portal up in the clouds to the village down in the ground. As an old Pegasus you are I am afraid you might not remember the path, so I have spawned some magic rings that will guide you. Just pass through those rings and I will give you a medal if you are fast at it.

In this game the player has freedom of choice: he can either pass through the rings to complete the circuit and be awarded a medal or just fly around and explore the environment.

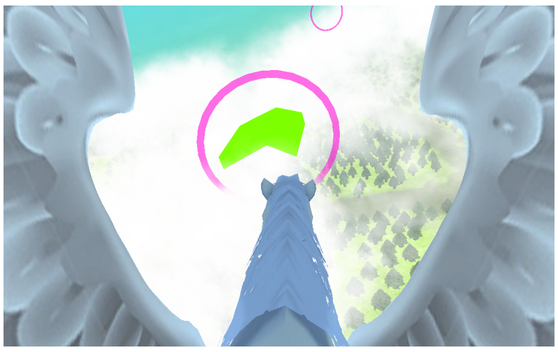


Figure 3.10: Pegasus Flight as seen from player's perspective

Game Goal

The goal is to complete a circuit which consists of a collection of way points as the ones seen in figure 3.10 that must be passed through in order in the shortest possible time. Rewards are given according to completion time:

- **Gold medal:** time less than 55 seconds.
- **Silver medal:** time between 55 and 85 seconds.
- **Bronze medal:** time greater than 85 seconds.

Interaction Design

For this game it is very important for the player to understand how the virtual character responds to his movements in order to control speed and making turns. The precision of these movements is not as crucial as the timing of them.

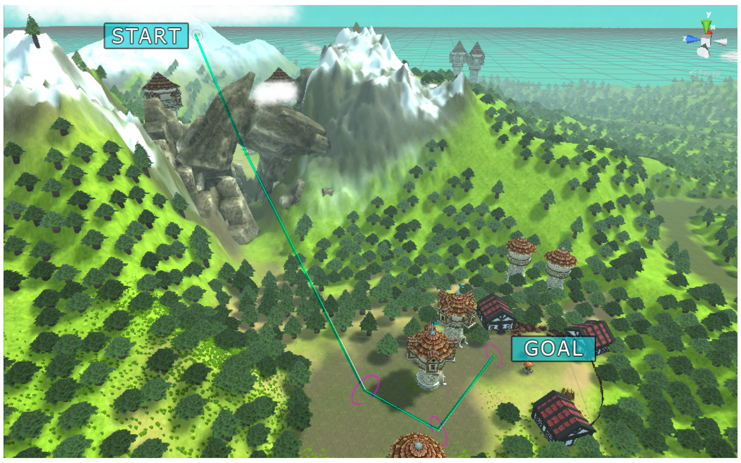


Figure 3.11: Circuit for Pegasus Flight.

The Oculus Rift orientation sensor data is used to control the direction of flight (i.e. the avatar turns in the direction that the player is looking).

The Kinect sensor is used to control flight speed. The game detects when the user flaps an arm or both at the same time. Depending on the amplitude of this arm movement the avatar will be given a stronger or weaker impulse forward. Player's arms orientation is also tracked for two purposes:

- Allowing the avatar to glide by adding a gravity compensating force proportional to how extended the player's arms are.
- Providing visual feedback to the player by mapping the player's arm movements to the avatar's wings.

In early tests it seemed that having a static frame of reference that moves with the camera helps reducing motion sickness when the horizon line moves. For this reason the camera partially shows the avatar's head and wings (see figure 3.10) instead of being positioned in the Pegasus head as it would be more logical for a first person experience. This setup also provides visual feedback for the player since he can see the wings moving like his arms.

Aural feedback was added too so that a sound is played when the player successfully flaps his arms and the gesture is detected.

Game States

The game has three different states that follow sequentially:

- **Calibration:** A brief calibration is required to align the Oculus Rift sensor with the in game camera orientation. This state is also intended for the player to familiarize with the avatar and see how it responds to his arms movements and head motion.
- **Ingame:** The game starts and a timer is displayed. If the player gets too far away from the next way point it is possible to reset the avatar's position to the last way point.

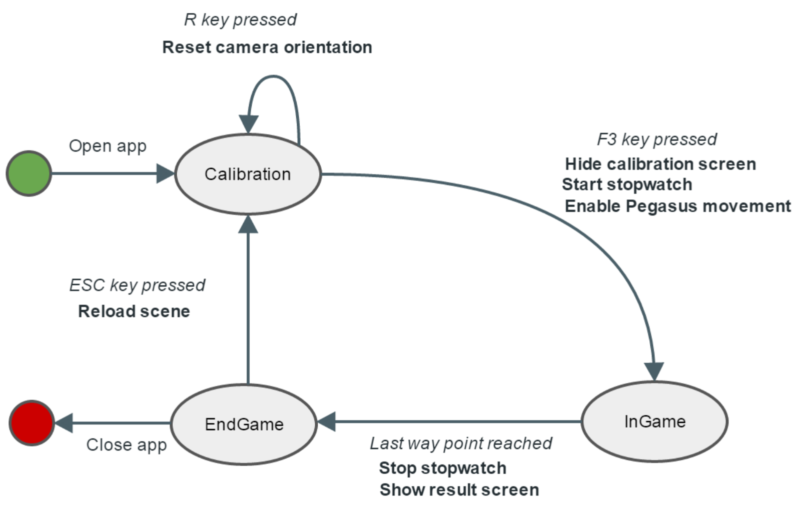


Figure 3.12: Global game state diagram for Pegasus Flight

- **Endgame:** The time taken to complete the circuit is displayed along with the medal awarded.

Scene Layout and Organization in Unity

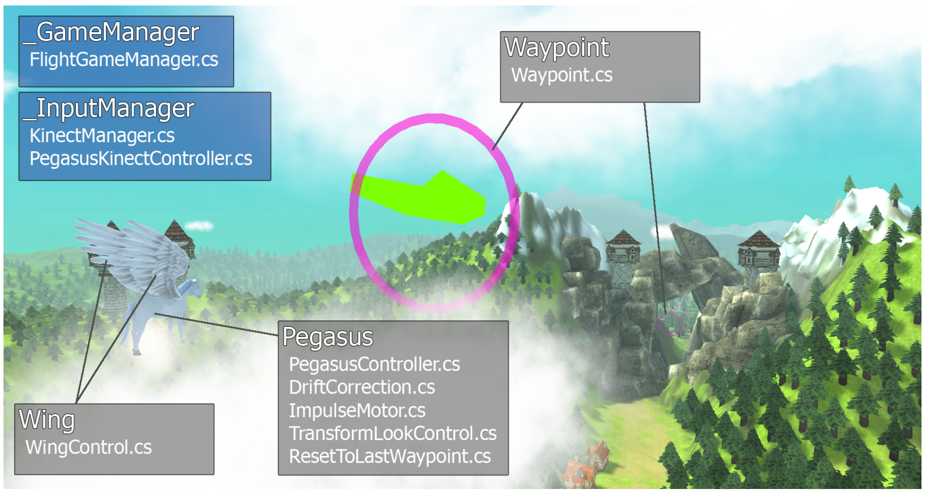


Figure 3.13: Pegasus Flight scene showing the relevant GameObjects and attached scripts.

The core functionality of the scene is grouped in these three GameObjects:

- Pegasus: The player's avatar itself. It has the following scripts attached:
  - PegasusController: Takes care of visual and sound effects and some

avatar logic such as detecting when it is landed or in the air to adjust the maximum reachable speed accordingly.

  - DriftCorrection: Reduces the drift caused by the physics engine so that the avatar is easier to control.
  - ImpulseMotor: Can apply an impulse that decays with time according to a configurable curve. It is responsible for the avatar's motion.
  - TransformLookControl: In charge of rotating the avatar towards the direction the player is looking.
  - ResetToLastWaypoint: When the configured key is pressed it returns the avatar to the last passed way point.

  These scripts are explained in more depth in the next section.

- _GameManager: Analogously to the Sword Training scene, this object has a FlightGameManager script attached to control the game state.
- _InputManager: It groups the input related scripts. In this case PegasusKinectController which talks to KinectManager [23] to extract the needed information, such as hand and shoulder positions, and listen to gesture detection events.

Script Description

Here we describe the most relevant scripts that we developed for the game:

FlightGameManager.cs
Implemented as a state machine, it controls the game flow and states described in section 3.3.2 in addition to some other functions:

- Listens to user input events such as the key pressed to start playing.
- Enables and disables the Pegasus movement depending on the game state.
- Listens to game events like when the Pegasus goes through a circuit way point or reaches the goal as shown in listing 3.5.
- Computes the final player score according to the time taken to reach the goal.
- Controls the display of all the GUI elements.

This class inherits from StateMachineBase, a useful script created by Mike Talbot [30].

Listing 3.5: Event subscription to the way point system

```
public void INGAME_OnEnterState()
{
  ...
  Waypoint.onWaypointReached += OnWaypointReached;
  Waypoint.onLastWaypointReached += OnLastWaypointReached;
  ...
}

public void INGAME_OnExitState()
{
  ...
  Waypoint.onWaypointReached -= OnWaypointReached;
  Waypoint.onLastWaypointReached -= OnLastWaypointReached;
  ...
}

public void OnWaypointReached(GameObject player)
{
  ++passedWaypoints;
  waypointsText.text = string.Format("{0} / {1}", passedWaypoints,
       totalWaypoints);
}

public void OnLastWaypointReached(GameObject player)
{
  currentState = States.ENDGAME;
}
```

Waypoint.cs
Way points work like a linked list of elements that must be traversed in order. For this reason each Waypoint object but the last one (considered to be the

goal) stores a reference to the next one in a chain-like fashion. A reference to the current Waypoint that must be gone through is also kept as a static field that can be accessed from the outside. Listing 3.6 shows how this behavior is implemented.

Listing 3.6: Code fragment for way point logic

```
public class Waypoint : MonoBehaviour
{
  ...

  public static Waypoint currentWaypoint;
  public Waypoint nextWaypoint;
  public bool isFirst;

  public static event Action<GameObject> onWaypointReached;
  public static event Action<GameObject> onLastWaypointReached;


  void Start ()
  {
  ...
    if (isFirst)
    {
      Waypoint.currentWaypoint = this;
    }
  ...
  }

  void OnTriggerEnter(Collider other)
  {
    if(other.tag == "Player")
    {
      // Is this the current waypoint that must be passed?
      if (Waypoint.currentWaypoint == this)
      {
        Waypoint.currentWaypoint = nextWaypoint;

        if(onWaypointReached != null)
        {
          onWaypointReached(collider.gameObject);
        }
        // Is this the last waypoint?
        if(nextWaypoint == null && onLastWaypointReached != null)
        {
          onLastWaypointReached(collider.gameObject);
        }
      }
      ...
    }
  }
  ...
}
```

The script fires events when a GameObject tagged as ”Player” enters its trigger collider only if the static field currentWaypoint is the same as the current script instance i.e., if the player is passing trough the way points in the correct order.

ResetToLastWaypoint.cs
When the specified key is pressed it will set the position of the GameObject to the last passed way point and stop it. In VR sudden changes of player's camera position can cause disorientation so this script fades out the screen to black before the position is changed and slowly fades in again once the operation is complete. The main purpose of it is to help players get back in track if they go too far away from the next way point.

PegasusKinectController.cs
This script interacts with KinectManager to control the Pegasus avatar in two ways:

- Implements the GestureListener interface provided in [23] to react to wing flap gestures when the user moves the arms up and down. When this flap gesture is detected the Pegasus avatar is given a forward impulse proportional to the amplitude of the arm movement. This is done through the PegasusController and ImpulseMotor scripts. It is also worth mentioning that aural feedback was added so that the wing flapping actually produces a sound with a volume proportional to the amplitude of the gesture.

- Keeps track of the player's hands position to know how the arms are rotated and move the avatar's wings accordingly via the WingControl script. The function shown in 3.7 returns a value of 1.0 when the user's right hand is exactly above his right shoulder and a value of 0.0 when it is exactly beneath it. This return value is passed to the WingControl script to use it as a normalized animation time and play the corresponding frame of the clip. The procedure is shown more clearly on figure 3.14.

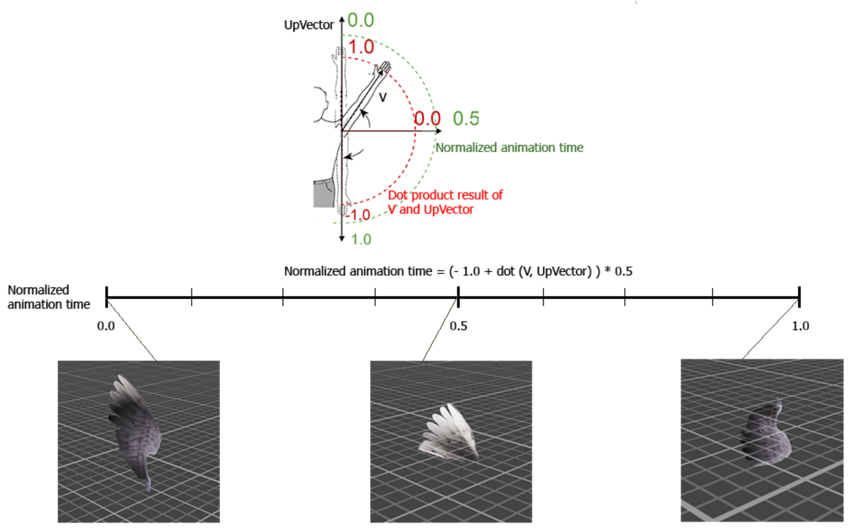

UpVector
0.0
1.0
V
0.0
0.5
Normalized animation time
Dot product result of V and UpVector
-1.0
1.0
Normalized animation time = (- 1.0 + dot (V, UpVector)) * 0.5
Normalized animation time
0.0
0.5
1.0

Figure 3.14: Calculation of animation time from arm rotation.

Listing 3.7: Code fragment calculate normalized animation time from hand position

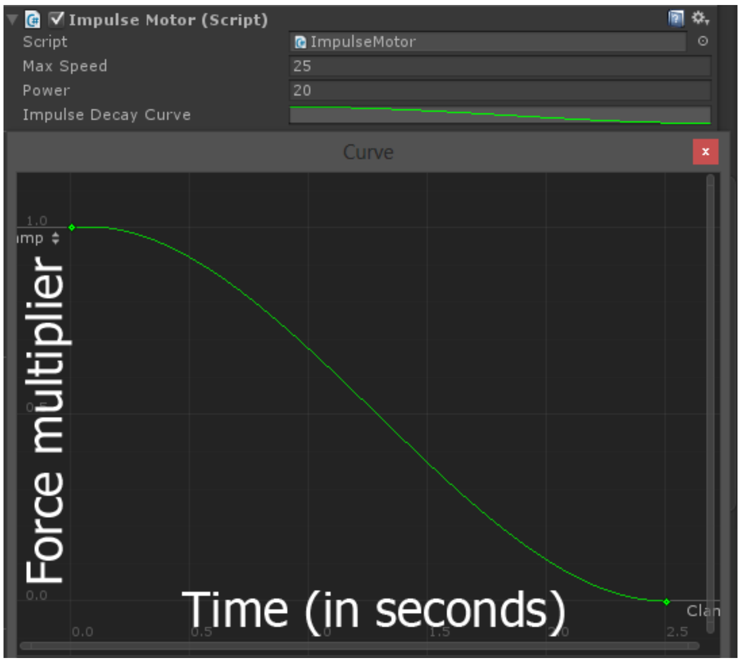
```
public float GetRightHandHeight()
{
  if (manager.IsJointTracked(userId, rShoulderIdx) && manager.
      IsJointTracked(userId, rHandIdx))
  {
    // Get hand and shoulder position
    Vector3 rightHandPos = manager.GetJointPosition(userId,
        rHandIdx);
    Vector3 rightShoulderPos = manager.GetJointPosition(userId,
        rShoulderIdx);

    // Calculate the vector that goes from shoulder to hand
        ignoring depth (z coordinate not used)
    Vector2 v = new Vector2(rightHandPos.x - rightShoulderPos.x,
        rightHandPos.y - rightShoulderPos.y).normalized;

    // Take the dot product of this vector with the (0,1) vector
    float r = Vector2.Dot(v, Vector2.up);

    // Map it to the [0,1] interval
    r += -1f;
    r *= -0.5f;
    return r;
  }
  return float.PositiveInfinity;
}
```

PegasusController.cs
It is in charge of controlling the avatar's movement (using the ImpulseMotor script) and rotation as well as visual and sound effects like a dust trail and footsteps when the Pegasus is walking on the ground. The script also sets the maximum speed that the avatar can reach depending on whether it is landed or in the air.

ImpulseMotor.cs
Exposes a public method called Impulse which when invoked applies a force over time to the GameObject. This force decays over time according to a configurable curve that can be tweaked to achieve the desired behavior. See figure 3.15. It also limits the GameObject's speed to the specified maximum value.

WingControl.cs
This script controls the wing animation playback according to the time value passed by PegasusKinectController and the vertical force added to the avatar so that it can glide and stay in the air. The magnitude of this force depends on the animation time in such a way that it is greater when the wing is closer to the horizontal position (or animation times around to 0.5).

DriftCorrection.cs
In order to make the avatar control easier and avoid possible simulator sickness caused by lateral motion (as described in [15] on page 21) it is useful to correct the drifting by adding a force that compensates it. This is simply done by taking the current velocity vector in world space and transforming it to local coordi-



Script: ImpulseMotor
Max Speed: 25
Power: 20
Impulse Decay Curve

Figure 3.15: Settings for the ImpulseMotor script.

nates to see how much of it is affecting the x-axis and add a force proportional to it with negative sign. The code for this script is shown in listing 3.8.

Listing 3.8: Code to correct rigidbody drifting.

```
public class DriftCorrection : MonoBehaviour
{
  public float driftCorrection = 4f;
  void FixedUpdate()
  {
    Vector3 relVelocity = transform.InverseTransformDirection(
        rigidbody.velocity);
    rigidbody.AddRelativeForce(-relVelocity.x * driftCorrection *
        Vector3.right);
  }
}
```

TransformLookControl.cs
This script maps the rotation of an external GameObject's transform to the GameObject it is attached to. This mapping can be modulated with a curve to smooth angle variations. In this scene its purpose is to map the player's head rotation obtained from the Oculus Rift to the avatar's rotation, this means it causes the Pegasus to rotate smoothly towards the direction the user is looking.

OculusResetPose.cs
A simple script that will re-center the VR camera when the specified key is pressed. It's used to calibrate the VR headset before the game starts.

3.3.3 Pegasus Ride

In this section we show a description of the different game aspects from the design point of view: the mission background we set, what the goal is, the way the player interacts with the game and the different states of it. Then we proceed with a description of the game scripts that make this design possible.

Game Mission Background

This time you are Edilmar the elf again. Now that your training with the sword is over and Compay the Pegasus is here, you might want to go for a ride and it will serve you both as a training. Again, I made it easier for you by spawning some magic rings that will guide you to the goal. By the way, you already know the deal: if you do it fast I will reward you with the gold medal!

Similarly to the previous game, in this experience the player has freedom of choice: he can either follow the rings to complete the circuit and receive a medal or walk around and explore.

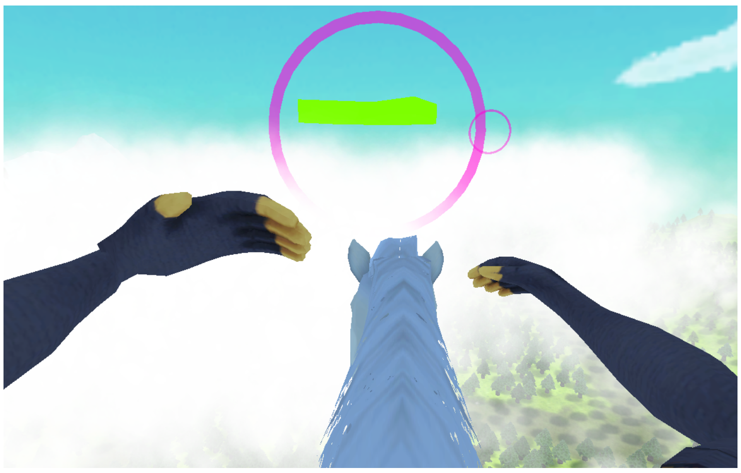


Figure 3.16: Pegasus Ride as seen from player's perspective. In this case the avatar's hands are visible.

Game Goal

Similarly to the previous game, the goal is to complete a circuit by going through a series of way points in order and as quickly as possible. The player is awarded with a medal according to the time taken to reach the goal:

- **Gold medal:** time less than 40 seconds.
- **Silver medal:** time between 40 and 60 seconds.
- **Bronze medal:** time greater than 60 seconds.

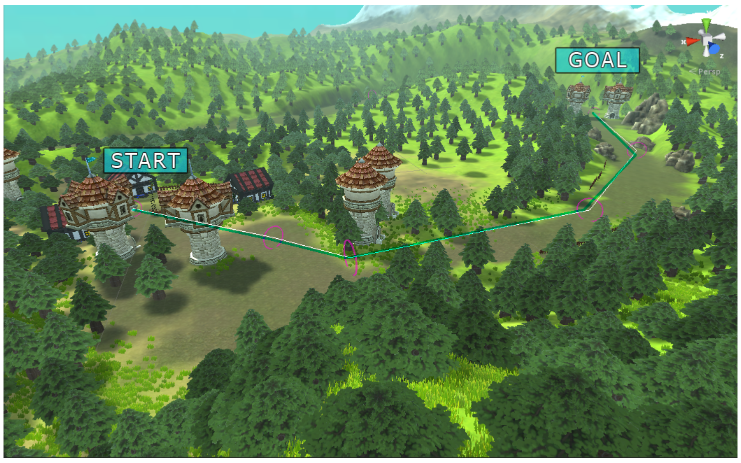


Figure 3.17: Circuit for Pegasus Ride.

Interaction Design

As for the previous game, the player must understand how the character responds to his movements in order to control speed and making turns. Although the timing of these movements is crucial, precision is important too. In this case however, the avatar is landed on the ground all the time and the movement speed is lower, creating a sense of more safety and lower difficulty.

The Kinect sensor is used to control both the speed and Pegasus direction.

Speed is controlled by moving both hands vertically and close to each other as if they were holding horse reins. The amplitude and frequency of this movement is used to determine the strength impulse transmitted to the Pegasus avatar.

The direction control is done by tracking the player's torso orientation and using it to rotate the Pegasus accordingly.

To increase the sense of immersion and provide visual feedback, the Kinect data is used to map player's movements to the human character riding the Pegasus.

Game States

The states for this game are the same as for the flying game:

- **Calibration:** A brief calibration is required to align the Oculus Rift sensor with in game orientation. This state is also intended for the player to familiarize with the avatar and see how it responds to his arms movements.

- **Ingame:** The game starts and a timer is displayed. In this state if the player gets too far away from the next way point it is possible to reset the avatar's position to the last way point.

- **Endgame:** The result screen showing the completion time and medal awarded is displayed.

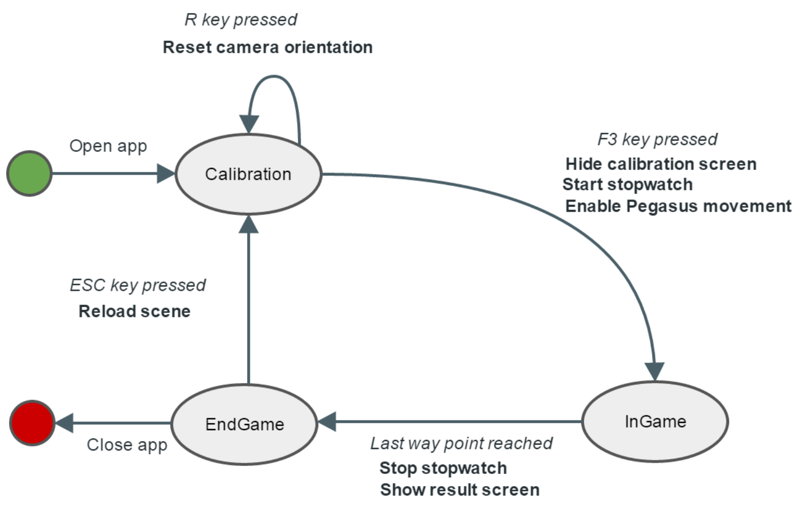


Figure 3.18: Global game state diagram for Pegasus Ride.

Scene Layout and Organization in Unity

The layout for this scene is almost identical to the one for Pegasus Flight since the game goal is the same. The only changes are in how the player controls the avatar and the location of the way points. (See figure 3.17).

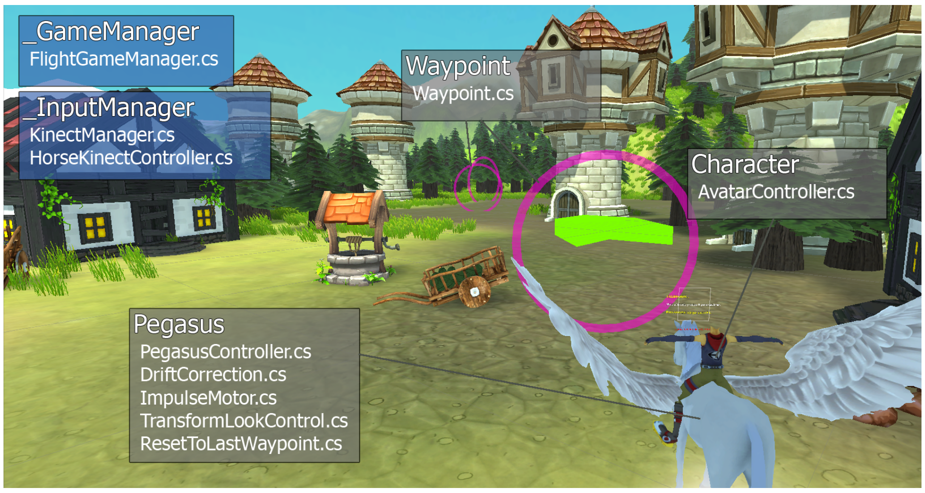
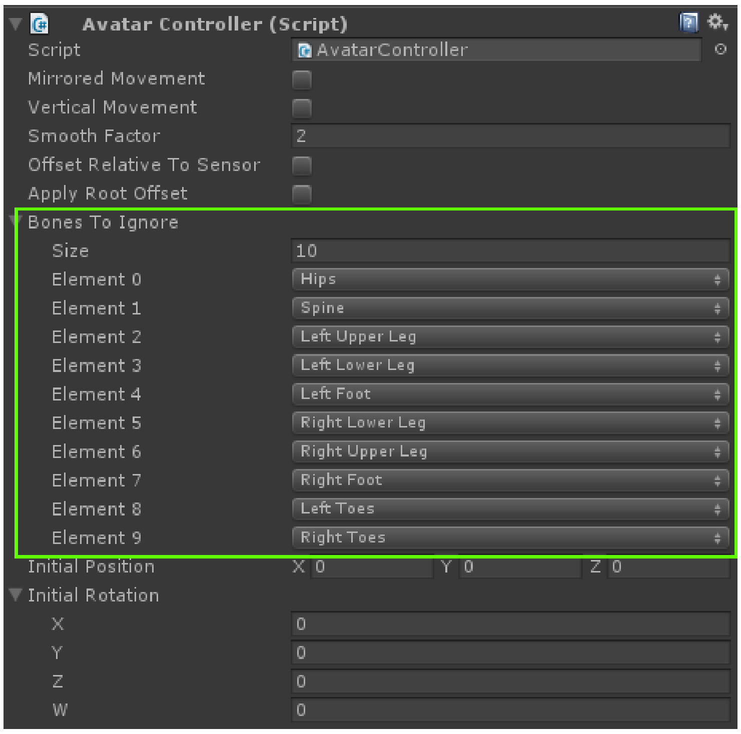


Figure 3.19: Pegasus Ride scene showing the relevant GameObjects and attached scripts.

- Character: The player's avatar itself. It is controlled by the AvatarController script provided in [23] that maps the skeletal data from kinect to the virtual character's skeleton. However for this case the script was modified to ignore certain bones and not to change their rotation as seen in figure 3.20. The reason for this is that we don't want to move the legs of the avatar since it is sitting on the Pegasus and it would not look right.



Avatar Controller (Script)

Script	AvatarController
Mirrored Movement	☐
Vertical Movement	☐
Smooth Factor	2
Offset Relative To Sensor	☐
Apply Root Offset	☐
Bones To Ignore	
Size	10
Element 0	Hips
Element 1	Spine
Element 2	Left Upper Leg
Element 3	Left Lower Leg
Element 4	Left Foot
Element 5	Right Lower Leg
Element 6	Right Upper Leg
Element 7	Right Foot
Element 8	Left Toes
Element 9	Right Toes
Initial Position	X 0 Y 0 Z 0
Initial Rotation	
X	0
Y	0
Z	0
W	0

Figure 3.20: AvatarController script configuration options showing the added "Bones To Ignore" list

The character GameObject is parented to the Pegasus so that it moves with it.

- Pegasus: The setup for this GameObject is almost the same with some small changes:
  - PegasusController: Just like in the Pegasus Flight scene, this script takes care of visual and sound effects and some avatar logic such as detecting when it is landed or in the air and adjusting the maximum speed accordingly. However, since most of the time the Pegasus will be grounded in this scene, the maximum speed on ground was tweaked to make it a bit lower.
  - TransformLookControl: In this case this script has been configured so that it causes the player to control the avatar direction with his torso.
  - DriftCorrection: Reduces the drift caused by the physics engine so that the avatar is easier to control.
  - ImpulseMotor: Can apply an impulse that decays with time according to a configurable curve. It is responsible for the avatar's motion.

  - ResetToLastWaypoint: When the configured key is pressed it returns the avatar to the last passed way point.

- _GameManager: Since the game states are the same as in Pegasus Flight scene, no changes had to be done for the FlightGameManager script so it has been reused as is.
- _InputManager: As before, this object holds the input related scripts. In this case however there is a HorseKinectController (instead of PegasusKinectController) that communicates with KinectManager [23] for gesture detection events and tracking of the player torso to control the avatar direction.

Script Description

Since an in depth explanation for all the scripts involved in this scene can be found in section 3.3.2, just the new one is described here.

HorseKinectController.cs
This script interacts with KinectManager to control the motion and orientation of the Pegasus in two ways:

- Implements the GestureListener interface provided in [23] to react to gestures when the user moves his hands together up and down. When this gesture is detected the Pegasus avatar is given a forward impulse proportional to the amplitude of the arm movement. This is done through the PegasusController and ImpulseMotor scripts.
- Keeps track of the player's torso rotation. This rotation is then applied to an external GameObject that, for the case of this scene, is being watched by the TransformLookControl 3.3.2 script controlling the Pegasus look direction. This setup enables the player to control the movement direction by rotating the torso.

3.4 Summary of Art Assets

Most of the art assets used (3D models and textures) were taken from the Internet for free, however some of them were modified or created from scratch to match the project requirements. Next there is list that summarizes all of them.

3.4.1 Environment

Name	**Author**	**Source**
Baker House	Evgenia	`https://www.assetstore.unity3d.com/en/content/26443`
Cartoon Lowpoly Water Well	Antonio Neves	`https://www.assetstore.unity3d.com/en/content/29717`
Cannon	Raúl Araújo	Modeled and hand painted in Blender.
Fortress Watchtower	Ma-at Art	`https://www.assetstore.unity3d.com/en/content/16779`
Free Rocks	TripleBrick	`https://www.assetstore.unity3d.com/en/content/19288`
FX Mega Pack	Unluck Software	`https://www.assetstore.unity3d.com/en/content/8933`
Medieval Toon House	Night Forest	`https://www.assetstore.unity3d.com/en/content/16674`
Medieval Wagon Pack	PantherOne	`http://opengameart.org/content/medieval-wagon-pack`
Mushroom Land	Manufactura K4	`https://www.assetstore.unity3d.com/en/content/1035`
Skybox	Raúl Araújo	Hand painted in Photoshop.
Watermelon	Raúl Araújo	Modeled in Blender and hand painted in Photoshop.

Table 3.1: 3D models and packages used for the environment design.

3.4.2 Characters

Name	**Author**	**Source**
Humanoid Avatar	Unity Technologies	`https://www.assetstore.unity3d.com/en/content/5328`
Animated Horse*	Dootsy Development*	`https://www.assetstore.unity3d.com/en/content/16687`

Table 3.2: 3D Character models used.

*Due to the difficulty of finding a usable Pegasus 3D model, this horse asset was modified by adding wings (modeled and animated from scratch in Blender) and a simple re-texturing (in Photoshop) as shown in figure 3.21.

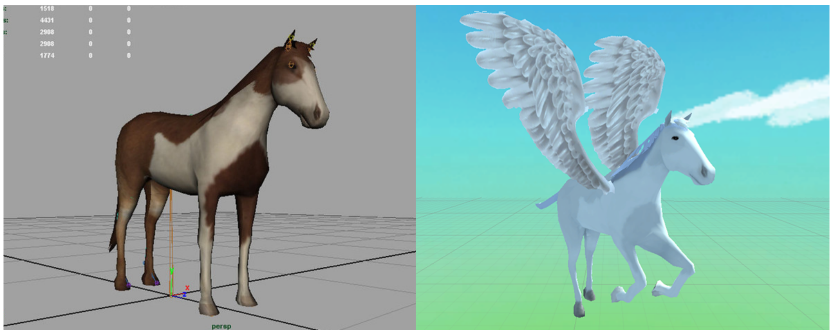


Figure 3.21: The Animated Horse model before and after having wings modeled and being re-textured.

Chapter 4

Results and Evaluation

The purpose of this chapter is to describe the different tests and validations we ran during the development and once we had our final system ready.

4.1 Development Tests

These tests are intended to decide which device will be used since the theoretical analysis of the state of the art was not fully conclusive. So in this section we describe the tests that we performed to find the most optimal device setup for our technical and budget requirements.

We started the testing phase using a Kinect sensor which was already available at the University and a Wii Remote controller (without Wii Motion Plus) due to the low price. However we soon realized that the Wii Remote had some important drawbacks:

- Non robust bluetooth connection and complex setup in Windows.
- High latency and low precision.
- Lack of Y axis rotation data.
- Tracking is lost when the LEDs are out of sight from the Wii Remote camera. This is an important issue since the player will not be able to see the external LED bar when using a VR headset so the tracking can be lost very often.

At this point we decided to try a completely different solution by taking advantage of the fact that most of current smartphones have an Inertial Measurement Unit inside. The problem was to transmit this data to the computer without using any wires. We tried an existing Android application called Wireless IMU[25]. This application sends raw data of each individual sensor in the device (accelerometer, gyroscope and magnetometer) over UDP via WiFi to a remote host in the specified IP address.

After a few tests performed with this application some issues were found:

- High jitter and latency spikes due to the WiFi connection making it non suitable for real time applications.

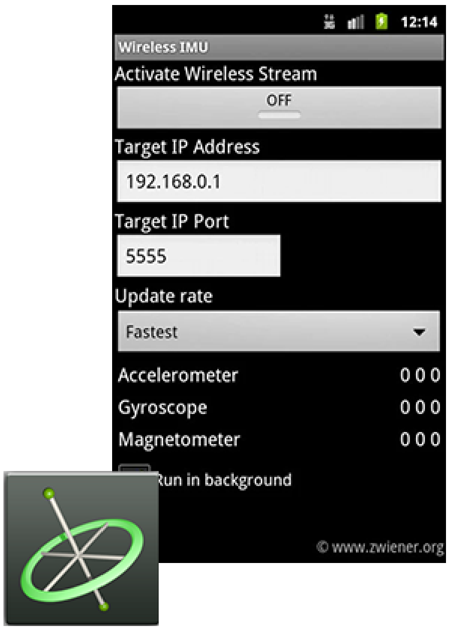


Figure 4.1: A screenshot from Wireless IMU showing some configuration options.
Source: `https://play.google.com/store/apps/details?id=org.zwiener.wimu`

- Requires the device to be connected to the same network than the computer making it inflexible.

- The application provides raw sensor data so the device orientation must be calculated from it in the computer side. Although it could be made since this is a well documented problem, it is not trivial [34]. For the tests a quick attempt was made to extract orientation from raw sensor data with non acceptable results. These results could be improved if further work was done, however we found an easier solution thanks to an Android API feature which internally makes all the required calculations and provides the device orientation directly as a vector [24] so no further processing is needed.

Knowing that the Andoid API can provide accurate orientation data easily and directly, all we needed was to develop a custom application to send this data to the computer as described in section (3.2). Our new Android application features:

- Bluetooth connection for lower latency. Also since bluetooth pairing is handled automatically by Windows and Android in both sides, the connection will be easy and quick unlike with the Wii Remote.

- Sending orientation data directly instead of raw sensor information.

4.2 Development Validation

During the latest stages of the development we wanted to perform some tests to know what changes and adjustments we had to make for the final validations at the primary school. We emailed some university students asking for volunteers to test the system and we finally gathered a group of ten that would be helping us.

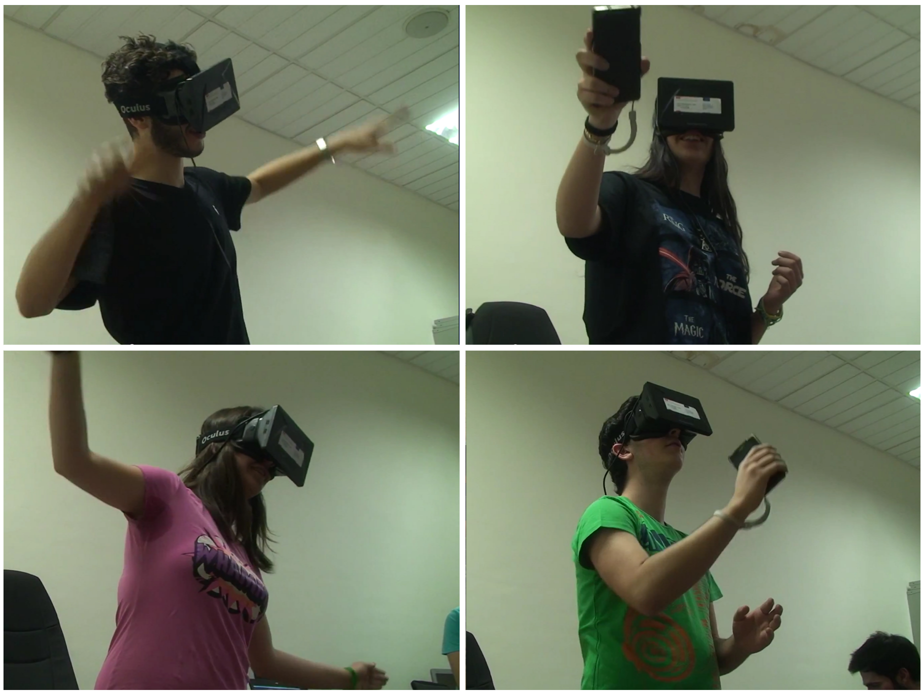


Figure 4.2: Volunteers testing the system in the lab.

For the tests, the users came in groups of two. They were given some basic indications on how to use the system and tried all the three games.

Although the users were adults instead of children, we extracted very useful feedback and conclusions:

- Most users got the general impression that the game was realistic.
- All the users thought the experience was fun.
- Most users got the general impression that the system was easy to use.
- Just one user mentioned the fact that the head of the Pegasus is visible in the Pegasus Flight game despite of being a first person game (see section 3.3.2).
- Most users thought that the Pegasus Flight circuit to the goal was too long and should be shorter.
- Most users thought that the Sword Training game was too short and the target number of watermelons to slice should be increased.

4.3 Final Validation

For the final validation we planned an experience within the context of the multiple intelligence development activities that were being carried out at ”CEIP Seseña y Benavente”. These activities were performed in groups of four or five children of ages between 7 and 9 and based on the book ”El Secreto de Marcos”[12]:

1. Drawing and describing a character from the book. Every team member does this individually for the same character. This activity is intended to exercise intelligences: visual, linguistic, intra-personal.

2. Showing the drawings and descriptions to the rest of the team members and putting together the similarities. This activity is intended to exercise intelligences: inter-personal, logical-mathematical and linguistic.

3. Matching different types of music to the previously described characters according to their personalities. First listening to the music for one minute and then discussing which character fits it better. Then moving around the classroom imitating that character to the rhythm of the music paying attention to the emotions and gestures that are transmitted. This activity is intended to exercise intelligences: intra-personal, musical, bodily-kinesthetic, linguistic and interpersonal.

4. Modeling the favorite character with clay and explaining why that character was chosen. This activity is intended to exercise intelligences: visual, intra-personal, interpersonal and linguistic.

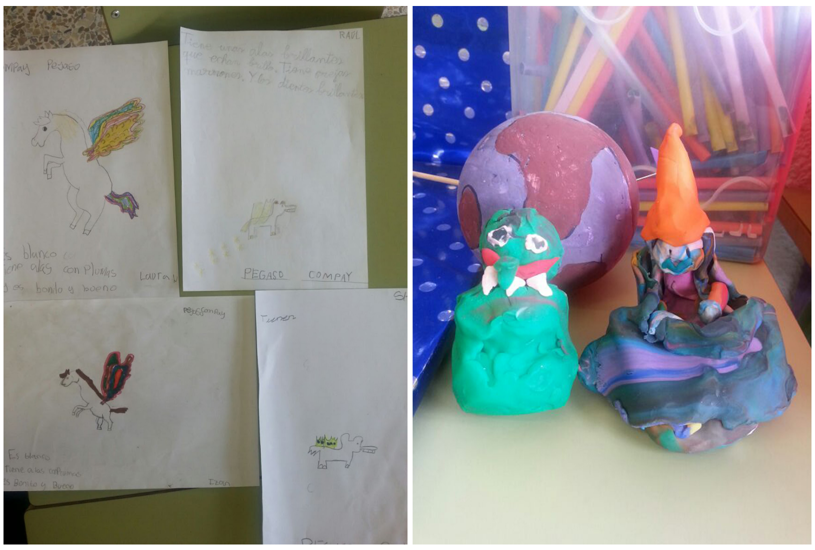


Figure 4.3: Some of the drawings and clay modeled characters made by the children.

Our experience took place right after the first activity (drawing the characters) in a separate room that was considered a magical cave and decorated for the occasion. The children came with their teammates (groups of four or five) while the rest of the class continued with the other activities. The children that tried the games were told to not say anything to the rest until everyone had tried it.

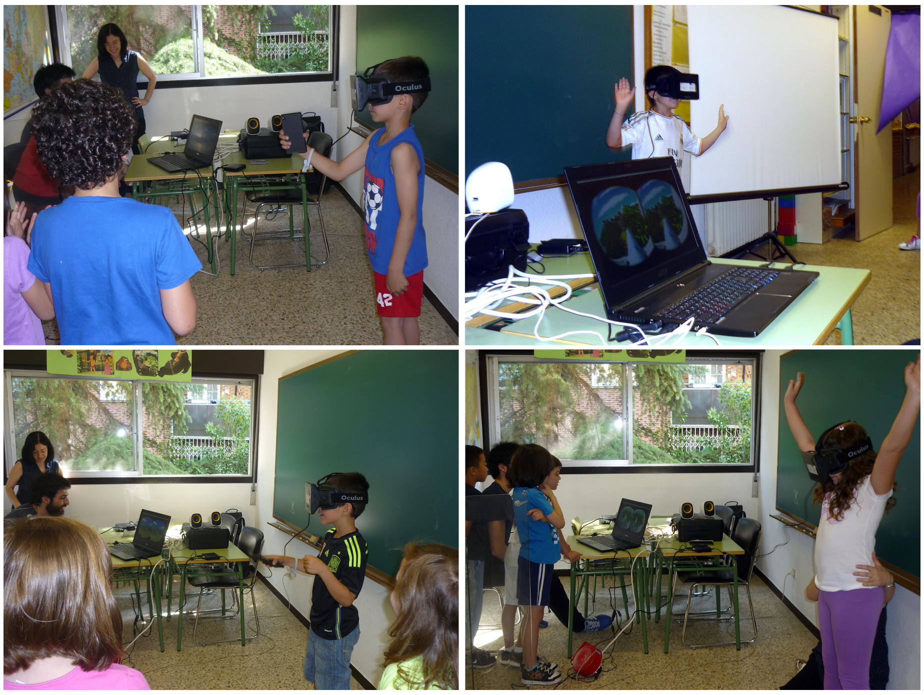


Figure 4.4: The intrepid players training with the sword (left) and flying as a Pegasus (right).

After running the tests with over 20 children, here are some of the conclusions we drew:

- Some of the children felt dizzy during the experience and some others felt alright all the time.
- In general terms they found it easy to slash watermelons but passing through the rings flying was difficult.
- The Pegasus Flight game was preferred over Sword Training.
- We thought of creating a sword-like prop or gadget where the smartphone could be attached to instead of grabbing the device directly to increase the realism. However most children liked the fact that of the "magical transformation" of a real everyday object into a virtual world fantastic item: The phone is just a phone but inside the real world it magically turns into a fancy sword.
- Most of the children expressed that they would buy the system if it was for sale.
- Most of the children would like to participate in a similar experience again.

- The VR experience was the best rated among all the activities by the children. In general, children preferred the activities involving psychomotor skills such as drawing, modeling, dancing or our games, rather than other type of activities (decision making, group discussions, etc.).
- After the activity most of the children showed an increased interest for reading the book and knowing more about the characters.

We also found some technical problems with the system. However none of them was crucial:

- Issues with Kinect and the room lighting affecting its correct functioning.
- The cables in the Oculus Rift can cause discomfort and accidents with the equipment if care is not taken.
- Having to manually adjust the height of the Kinect sensor to adapt to the different heights of the children. However the tracking was robust to the different heights, not being affected by the separation between joints and the movement mapping to the avatar was correct.

Additionally to these conclusions, we conducted a survey for the children to fill at home with their parents. Currently we are still collecting and analyzing the data to reflect the results in an article for the IEEE Transactions on Learning Technologies (TLT).

Chapter 5

Project Management and Planning

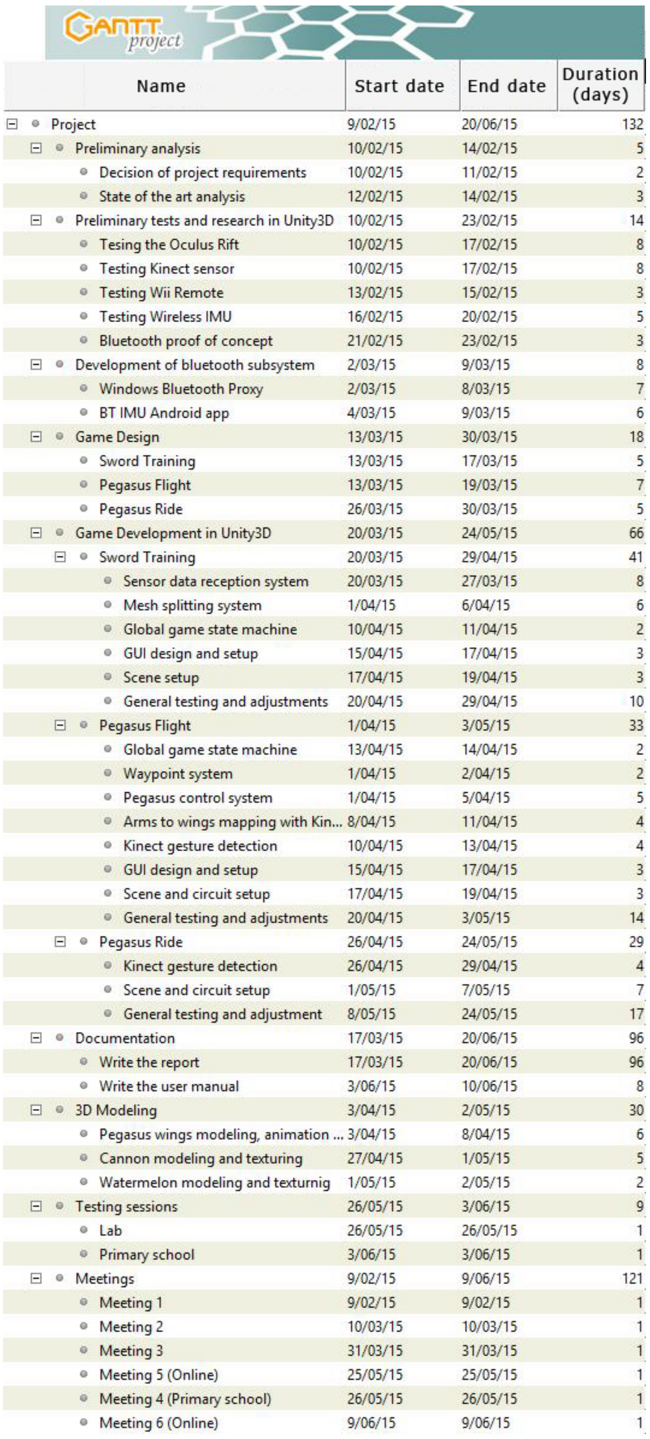
In this chapter we present the time plan for the project as well as a budget estimation for the it.

5.1 Project Time Plan

Since the objectives of the project have been modified and new ones were defined during the early stages, it was not possible to establish a strict time plan from the very start. In addition to that, the final phase of the development overlapped with class schedule and the exam season. For this reason, the Gantt diagram (figure 5.1) does not reflect the actual hours required for each stage, instead it represents the starting and finishing dates of them. However, an estimation of the real work hours dedicated to each part is presented in tables 5.1 and 5.2.

Most of the development tasks could be developed simultaneously or within overlapping time periods since the modularity of the system allowed to develop and test different parts independently. Nevertheless, one of the most time consuming stages was the testing and adjustment of parameters for each game once its different systems had been developed. The purpose of these adjustments was to create a comfortable experience as well as balancing the difficulty and tweaking of controls.

The time distribution of each task is reflected in figure 5.2 as a Gantt diagram along with a textual list of them in figure 5.1.



Name	Start date	End date	Duration (days)
Project	9/02/15	20/06/15	132
Preliminary analysis	10/02/15	14/02/15	5
Decision of project requirements	10/02/15	11/02/15	2
State of the art analysis	12/02/15	14/02/15	3
Preliminary tests and research in Unity3D	10/02/15	23/02/15	14
Tesing the Oculus Rift	10/02/15	17/02/15	8
Testing Kinect sensor	10/02/15	17/02/15	8
Testing Wii Remote	13/02/15	15/02/15	3
Testing Wireless IMU	16/02/15	20/02/15	5
Bluetooth proof of concept	21/02/15	23/02/15	3
Development of bluetooth subsystem	2/03/15	9/03/15	8
Windows Bluetooth Proxy	2/03/15	8/03/15	7
BT IMU Android app	4/03/15	9/03/15	6
Game Design	13/03/15	30/03/15	18
Sword Training	13/03/15	17/03/15	5
Pegasus Flight	13/03/15	19/03/15	7
Pegasus Ride	26/03/15	30/03/15	5
Game Development in Unity3D	20/03/15	24/05/15	66
Sword Training	20/03/15	29/04/15	41
Sensor data reception system	20/03/15	27/03/15	8
Mesh splitting system	1/04/15	6/04/15	6
Global game state machine	10/04/15	11/04/15	2
GUI design and setup	15/04/15	17/04/15	3
Scene setup	17/04/15	19/04/15	3
General testing and adjustments	20/04/15	29/04/15	10
Pegasus Flight	1/04/15	3/05/15	33
Global game state machine	13/04/15	14/04/15	2
Waypoint system	1/04/15	2/04/15	2
Pegasus control system	1/04/15	5/04/15	5
Arms to wings mapping with Kin...	8/04/15	11/04/15	4
Kinect gesture detection	10/04/15	13/04/15	4
GUI design and setup	15/04/15	17/04/15	3
Scene and circuit setup	17/04/15	19/04/15	3
General testing and adjustments	20/04/15	3/05/15	14
Pegasus Ride	26/04/15	24/05/15	29
Kinect gesture detection	26/04/15	29/04/15	4
Scene and circuit setup	1/05/15	7/05/15	7
General testing and adjustment	8/05/15	24/05/15	17
Documentation	17/03/15	20/06/15	96
Write the report	17/03/15	20/06/15	96
Write the user manual	3/06/15	10/06/15	8
3D Modeling	3/04/15	2/05/15	30
Pegasus wings modeling, animation ...	3/04/15	8/04/15	6
Cannon modeling and texturing	27/04/15	1/05/15	5
Watermelon modeling and texturnig	1/05/15	2/05/15	2
Testing sessions	26/05/15	3/06/15	9
Lab	26/05/15	26/05/15	1
Primary school	3/06/15	3/06/15	1
Meetings	9/02/15	9/06/15	121
Meeting 1	9/02/15	9/02/15	1
Meeting 2	10/03/15	10/03/15	1
Meeting 3	31/03/15	31/03/15	1
Meeting 5 (Online)	25/05/15	25/05/15	1
Meeting 4 (Primary school)	26/05/15	26/05/15	1
Meeting 6 (Online)	9/06/15	9/06/15	1

Figure 5.1: List of project tasks with start and end dates as well as duration in days.

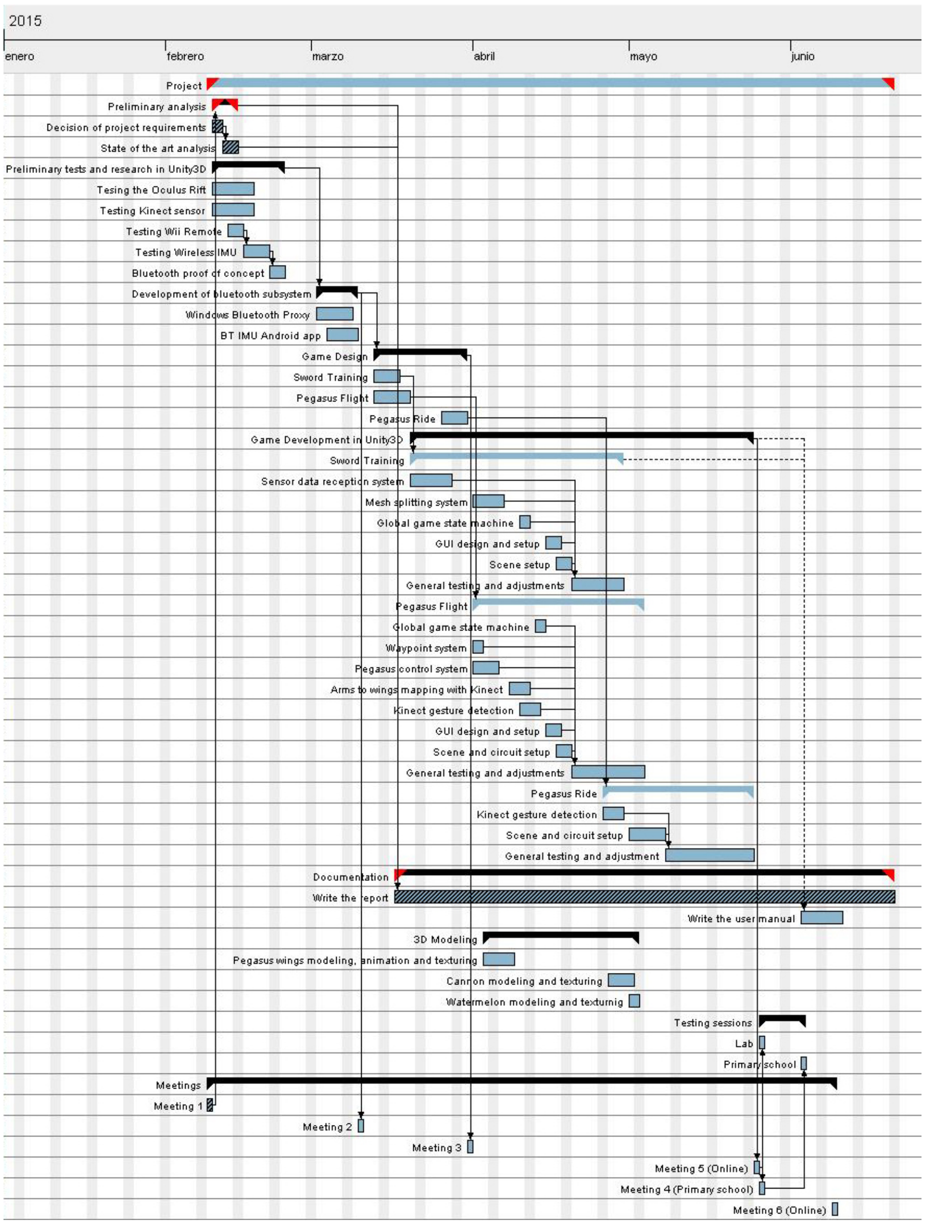


Figure 5.2: Gantt diagram of the project with the critical tasks grayed out.

5.2 Budget Considerations

In this section we will describe the different costs involved in the development of the project. According to their origin we will classify them in personnel, equipment and software license costs.

5.2.1 Staff Costs

The development of the system required the work of a project tutor (Ph.D.) and a developer (undergraduate). Considering the average salary per hour (20€/h for the developer and 37€/h for the project tutor) and the total work hours we can deduce an estimation of the final personnel associated costs.

Developer

Stage	**Description**	**Time**	**Cost**
Analysis	State of the art analysis and scope of the project.	12h	240€
Preliminary tests	Practical analysis and testing of the different technologies and their possible integration.	52h	1040€
Bluetooth subsystem	Development of the bluetooth related software components.	38h	760€
Game design	Prototyping the game goals, mechanics and controls.	50h	1000€
Game development	Integration of the different systems and art assets.	63h	1260€
Documentation	Writing of this document and manuals.	81h	1620€
Testing sessions	Tests and demos run in the lab and primary school.	11h	220€
Meetings	On-line and physical meetings.	6h	120€
Total		**323h**	**6460€**

Table 5.1: Estimated personnel costs for the project developer.

Project Tutor

Stage	**Description**	**Time**	**Cost**
Tutoring	Tutoring and supervision of the different project stages.	22h	814€
Management	Management and arrangement of meetings and project milestones.	10h	370€
Meetings	On-line and physical meetings.	6h	222€
Testing sessions	Tests and demos run in the lab and primary school.	11h	407€
Total		**49h**	**1813€**
Total		**362h**	**8073€**

Table 5.2: Estimated personnel costs for the project tutor.

5.2.2 Software License Costs

All the software used for the development and the price for the respective licenses is listed in table 5.3.

*The Windows 8 license was obtained for free thanks to the DreamSpark program (`https://www.dreamspark.com/`) so it had no costs.

5.2.3 Equipment Costs

All the hardware and peripherals used for the development and tests are included in this section (table 5.4). Note that some of the components are no longer available or the prices might be different now from those at the purchase date so some estimations were made.

Software	**Price**
Unity3D Free	0€
Adobe Photoshop (6 month license)[35]	14.99€/month x 6
Blender	0€
Visual Studio Community 2013	0€
TeXstudio	0€
Windows 8	0€*
GoldWave Free Edition	0€
Total	**89.94€**

Table 5.3: Software license costs.

Component	Cost	Lifetime	Usage time	Chargeable cost
Development Computer				
Intel i5-2500k CPU	189€	7 years	3 months	6.75€
Asus P8Z68-V LE Motherboard	129€	7 years	3 months	4.60€
MSI GeForce GTX 970 GPU	389€	5 years	3 months	19.45€
Samsung 840 250Gb SSD	180€	7 years	3 months	6.43€
Seagate Desktop 7200.14 2TB	72.95€	7 years	3 months	2.60€
Corsair VS650 650W PSU	67€	8 years	3 months	2.09€
Corsair 600T Tower	160€	10 years	3 months	4€
Peripherals				
x2 LG IPS234 Monitors	2x126€	6 years	3 months	10.5€
Steelseries Rival Mouse	49.95€	5 years	3 months	3.24€
Logitech G110 Keyboard	64.95€	5 years	3 months	2.49€
Testing Laptop				
MSI GS60 Ghost 415ES	1529€	6 years	3 months	63.70€
Project Peripherals				
Oculus Rift DK1	(*) 266.42€	4 years	3 months	16.6€
Sony Xperia Z2	399€	3 years	3 months	33.25€
Kinect Sensor	69.99€	5 years	3 months	3.49€
Total	3818.26€			**179.28€**

Table 5.4: Estimated equipment costs.

*The Oculus Rift DK1 model is no longer available for purchase so its original selling price of $300 (approximately 266€) was considered.

5.2.4 Total Cost Summary

A summary and the estimated total cost for the project is presented in table 5.5.

Concept	**Cost**
Personnel	8073€
Hardware	179.28€
Licenses	1189.94€
Total	**9442.22€**

Table 5.5: Summary of the total project costs.

Chapter 6

Conclusions

In this chapter we make a review of the objectives we had set initially to check whether or not we accomplished them. Additionally we present some conclusions drawn from the development and tests.

6.1 Review of the Project Objectives

- **Create a VR experience, in the form of a game, that is intense, immersive and fun.** As we could see during the testing sessions, most users found the system realistic and immersive and practically all of them expressed that they had fun using it.

- **The game atmosphere, locations and characters have to be based on the novel "El Secreto de Marcos"[12].** We successfully managed to reproduce some of the book's locations and characters. We had the author's approval since the beginning, he tried the games and he included the experience as a section of the book's website: `http://www.elsecretodemarcos.com/realidad-virtual`. In addition he wants to be involved further in the activities we have planned for next July in a hospital.

- **Testing the game with final users and gathering their feedback, reactions and opinions.** We could successfully test the system in a laboratory and a real world environment (10 university students for the testing phase and over 20 children from second course of primary education for the deployment). The experience we performed in the primary school was only for a group of children, however the school requested us to repeat the experience with more groups from first and second course and with students from the "Aula Arcoiris" which involves children with autism spectrum disorder. In addition the parents collective whose children took part of the experience have requested an extra session for parents to participate as well.

- **Develop at least one experience where the user controls the avatar using his body.** This objective was achieved since we could develop two of these experiences: Pegasus Flight and Pegasus Ride.

- **Develop at least one experience where a physical object is used to interact with the virtual world.** This objective was successfully accomplished as shown by the Sword Training game.
- **Develop a system that is easy to setup.** We were able to successfully try the system with twenty children in around three hours, which in our opinion is a reasonable amount of time, and with almost no technical issues except for the amount of cables involved.
- **Develop a system that is affordable.** As shown in section 5.2, we could finish the project with a reasonably low budget. However, if this system was to be sold commercially, the required special hardware would be a kinect sensor, which is indeed affordable and an Oculus Rift which currently is not cheap but as explained in the next section (A.6), there are alternative devices that will be available in the future and without requiring cables to allow for increased mobility.

6.2 Development and Test Conclusions

- **Development of Oculus Rift experiences in Unity.** The used version of Unity (4.6) did not support the Oculus Rift natively. This made the development and debugging process more complex that it should be. However during the development phase, Unity version 5.1.1 was released, minimizing this problem.
- **Creating a visually appealing game with very low budget.** When developing a videogame or a virtual experience, the visuals are usually the most budget and time consuming part. However it has been shown by general user feedback that we managed to create a visually acceptable experience with minimum time consumption and near to zero cost. This was possible thanks to the multiple on-line asset stores that provide this content for free and the custom modifications made. See section 3.4.
- **Real world tests are very valuable.** From this experience we can conclude that testing with final users always provides new and valuable information and brings to light issues that otherwise could be unseen.

6.3 Personal Conclusions

- Being able to complete a project of this size from star to finish provided a high amount of valuable experience.
- Testing the system in a real world environment, seeing that it has a real application and receiving real feedback was very rewarding.
- It would be a very interesting experience to take part in a full, bigger scale project of this kind.

Chapter 7

Future Development

Although the project achieved its technical goals, there are plenty of areas that could be improved or extended or even new scenarios where it could be used. In this section we mention some of them.

Smartphone Controls

- Currently the system supports only one smartphone as input, however it would be possible to support a higher number by, for example, modifying the Bluetooth Proxy Application so that it accepts more bluetooth connections and forwards them to different UDP ports. This could have numerous uses such as tracking of both hands or different parts of the body where a smartphone can be attached, similar to what the Sixense STEM system does with its additional sensor modules (see section 2.1.2).
- Extending the capabilities of the BT IMU application so that it can send data of pressed buttons or touchscreen input.
- Adding vibration and sound for haptic and aural feedback in the device.

Porting the Game to the New Unity version
With the announcement and release of Unity version 5.1.1 the integration with virtual reality devices is much easier since it is supported natively, not only for the Oculus Rift but also for Samsung GearVR and Microsoft HoloLens.

Porting this project to the new unity version would allow taking profit of all the new features and make future developments easier.

Seamless Bluetooth Integration with Unity
Running the external Bluetooth Proxy application every time before starting the game is not an optimal solution, especially for a final user. Two possible ways to address this issue could be:

- Creating a bluetooth plugin with C++ and the Pro version of Unity [28].
- Using an external application like the currently solution but making it run in the background as a service.

Reduce Keyboard Dependency
At the current stage, the user has to press a button to start playing. While this is not an important problem, it can be solved by using a user interface completely adapted to VR which implements look-based input (i.e., keep looking at a certain interface element to interact with it).

Therapeutic Purposes

The game was presented in the congress *"I Jornadas Internacionales de Actualización, Evidencia y Accesibilidad en Silla de Ruedas"* which took place at Universidad Rey Juan Carlos de Madrid and it was very well received by therapists for its potential uses as a therapy tool. Currently we are assessing the possibility of trying the system with children which due to a disease must stay in a hospital.

Appendix A

Extended Abstract

This annex provides an extended overview of the relevant ideas and sections presented throughout the full document.

A.1 Introduction and Project History

Since its early beginnings, Virtual Reality (VR) has been a technology with an enormous potential due to its inherent three-dimensional structure, both in terms of display and interaction. According to Briston [Brison, 1995], the vision of a three-dimensional environment applied to three-dimensional tasks and of highly intuitive interfaces which make the computer hardware "invisible" seems to be an entirely reasonable and desirable vision.

However, the adoption of VR has been neither as fast nor as extensive as expected at the beginning. There could be a variety of reasons for this failure, but one of the most important was that the available interface hardware (Head-mounted Display (HMD), trackers, etc.) during the first decades failed to deliver the effects of immersion or presence required for many tasks mainly due to performance issues.

Recent advances in stereoscopic displays such as VR glasses (e.g: Oculus Rift, Samsung Gear) and other consumer devices available on the mass market such as hand-held terminals (smartphones, tablets), and game platforms (Wii, PlayStation) makes the technology of sensors and trackers much more affordable for the users. This fact has allowed virtual reality to emerge from the simulation field to a wider range of different applications such as manufacturing [2], neuropsychology [3] ,rehabilitation [4], [5] and for leisure (games, films) [6].

Additionally, the relationships between body, mind and emotions have been widely exploited in high level sports and dance performance, but its penetration in academic environments is still very slow [7]. The theory of multiple intelligences by Gardner [8], the growing interest in embodied cognition [9] and studies supporting the connection between motor activity and brain neuroplasticity [4], [10] have once again drawn attention to the potential of body-mind-emotions connections in learning environments.

Taking advantage of these two tendencies and the fact that several authors have highlighted the potential of VR to provide a multi-sensory learning feedback, the main objective of this project is to assess the integration possibilities of the current commercial devices and develop a prototype that consists of a set of simple games, which involve physical activity and explore different ways of interaction with the virtual world. Finally it would be necessary to tests these games in a primary school with children in order to explore their reactions and impressions.

Presenting the idea of this activity in a school before starting the actual development allowed to redefine the initial objectives for a better integration of the games within the activities carried out there as well as a more practical application.

A.2 Summary of the Project Objectives

After all the initial modifications and definitions, we set the following objectives:

A.2.1 Functional Objectives

- Create a VR experience, in the form of a game, that is intense, immersive and fun.
- The game atmosphere, locations and characters have to be based on the novel "El Secreto de Marcos"[12].
- Testing the game with final users and gathering their feedback, reactions and opinions.

A.2.2 Technical Objectives

- Develop at least one experience where the user controls the avatar using his body.
- Develop at least one experience where a physical object is used to interact with the virtual world.
- Develop a system that is easy to setup.
- Develop a system that is affordable.

A.3 State of the Art and Implementation Process

We analyzed the most widely used sensor technologies as well as alternatives of commercial tracking devices. Most of them were quickly discarded due to either complexity or high price. However after this analysis we still needed to perform tests with some of the devices we had available before making the choice.

Finally the preferred choice was a combined solution consisting of a Kinect sensor and taking advantage of the capabilities of current smartphones as an Inertial Measurement Unit (IMU).

This setup required us to develop a system to send the smartphone sensor data to the computer so we chose bluetooth for this purpose and created both an Android and a Windows application that could successfully communicate using this technology.

Once this system was ready we started integrating it inside Unity 3D and did the same for Kinect and the Oculus Rift.

The next step was to develop the actual games in Unity and gathering and adapting the different art assets (models, animations, textures, sound effects...) that made part of the system.

Finally adjustments were made by trial and error to adjust the difficulty and controls.

A.4 Conclusions

A.4.1 Development and Test Conclusions

In this section a brief summary of the relevant conclusions drawn is presented. Additionally a summarized validation of the project objectives and their final degree of accomplishment is shown.

A.4.2 Validation of the Project Objectives

- Create a VR experience, in the form of a game, which produces a sense of embodiment being intense, immersive and fun. **Successfully accomplished.**
- The game atmosphere, locations and characters have to be based on the novel "El Secreto de Marcos"[12]. **Successfully accomplished.**
- Testing the game with final users and gathering their feedback, reactions and opinions. **Successfully accomplished.**
- Develop at least one experience where the user controls the avatar using his body. **Successfully accomplished.**
- Develop at least one experience where a physical object is used to interact with the virtual world. **Successfully accomplished.**
- Develop a system that is easy to setup. **Accomplished but improvable.**
- Develop a system that is affordable. **Accomplished but improvable.**

A.4.3 Technical Conclusions

Despite of the fact that most of the technologies used are not new, the integration of all of them was not a trivial process and it is still experimental. However this situation will be improved as the VR systems consolidate making future developments easier.

Another important point we realized is that currently one can create a visually appealing game or simulation, with a very low budget, by using free resources that are available on-line specially on the Unity Asset Store.

Finally we could check first hand how valuable the testing phase with final users is.

A.5 Personal Conclusions

- Being able to complete a project of this size from star to finish provided a high amount of valuable experience.
- Testing the system in a real world environment, seeing that it has a real application and receiving real feedback was very rewarding.
- It would be a very interesting experience to take part in a full, bigger scale project of this kind.

A.6 Future Development

Although the project achieved its technical goals, there are plenty of areas that could be improved or extended or even new scenarios where it could be used. In this section we mention some of them.

Smartphone Controls

- Currently the system supports only one smartphone as input, however it would be possible to support a higher number by, for example, modifying the Bluetooth Proxy Application so that it accepts more bluetooth connections and forwards them to different UDP ports. This could have numerous uses such as tracking of both hands or different parts of the body where a smartphone can be attached, similar to what the Sixense STEM system does with its additional sensor modules (see section 2.1.2).
- Extending the capabilities of the BT IMU application so that it can send data of pressed buttons or touchscreen input.
- Adding vibration and sound for haptic and aural feedback in the device.

Porting the Game to the New Unity version
With the announcement and release of Unity version 5.1.1 the integration with virtual reality devices is much easier since it is supported natively, not only for the Oculus Rift but also for Samsung GearVR and Microsoft HoloLens.

Porting this project to the new unity version would allow taking profit of all the new features and make future developments easier.

Seamless Bluetooth Integration with Unity
Running the external Bluetooth Proxy application every time before starting the game is not an optimal solution, especially for a final user. Two possible ways to address this issue could be:

- Creating a bluetooth plugin with C++ and the Pro version of Unity [28].
- Using an external application like the currently solution but making it run in the background as a service.

Reduce Keyboard Dependency
At the current stage, the user has to press a button to start playing. While this is not an important problem, it can be solved by using a user interface completely adapted to VR which implements look-based input (i.e., keep looking at a certain interface element to interact with it).

Therapeutic Purposes

The game was presented in the congress *"I Jornadas Internacionales de Actualización, Evidencia y Accesibilidad en Silla de Ruedas"* which took place at Universidad Rey Juan Carlos de Madrid and it was very well received by therapists for its potential uses as a therapy tool. Currently we are assessing the possibility of trying the system with children which due to a disease must stay in a hospital.

Appendix B

User Manual

B.1 System Requirements

The minimum system requirements to run a Unity3D game are the following[36]:

- OS: Windows XP+
- Graphics card: DX9 (shader model 2.0) capabilities.
- CPU: SSE2 instruction set support.

However, for VR experiences, playing at 60 frames per second is a must. For that reason we provide the following recommended system specifications that should achieve such frame rate:

- CPU: Intel Core 2 Quad CPU Q6600 @ 2.40GHz / AMD Phenom 9850 Quad-Core Processor @ 2.5GHz
- System Memory: 4 GB
- OS: Windows 7 32/64 bit Service Pack 1
- Video Card: NVIDIA GeForce 8800 GT 1GB / AMD Radeon HD 4870 1GB
- Bluetooth 2.0 Adapter (required only for the Sword Training game)
- Free Disk Space: 128MB

B.1.1 Third Party Software

- Microsoft Kinect Runtime (version 1.8):
  `https://www.microsoft.com/en-us/download/details.aspx?id=40277`
- Oculus Rift Runtime (version 0.4.4):
  `https://developer.oculus.com/downloads/pc/0.4.4-beta/Oculus_Runtime_for_Windows/`

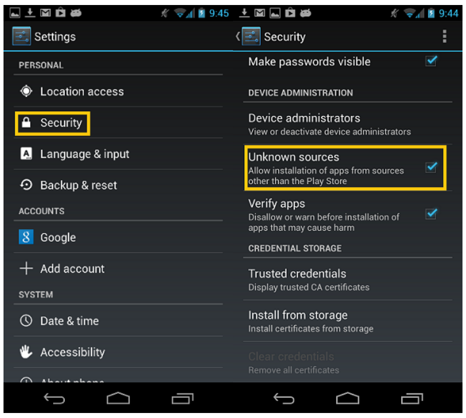
B.2 First Time Configuration

All the steps described in this section are only required if this is the first time using the game or you are going to use a new Android phone as a controller.

B.2.1 Android Application Installation

NOTE: BT IMU application requires Android version 4.3 or later.

Once the **bt_imu.apk** file has been copied to the phone, open it with your preferred Android file browser and follow the typical procedure to install an Android application. You might need to allow installation of applications from unknown sources. See figure B.1.


9:45

Settings

PERSONAL

Location access

Security

Language & input

Backup & reset

ACCOUNTS

Google

Add account

SYSTEM

Date & time

Accessibility

9:44

Security

Make passwords visible ✓

DEVICE ADMINISTRATION

Device administrators
View or deactivate device administrators

Unknown sources ✓
Allow installation of apps from sources other than the Play Store

Verify apps ✓
Disallow or warn before installation of apps that may cause harm

CREDENTIAL STORAGE

Trusted credentials
Display trusted CA certificates

Install from storage
Install certificates from storage

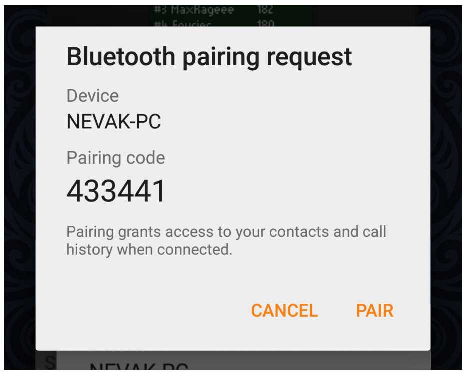
Clear credentials
Remove all certificates

Figure B.1: How to allow installation from unknown sources.
Image from `www.axiamo.com`

B.2.2 Computer and Phone Bluetooth Pairing

In order to establish a bluetooth connection between two devices the must be paired first. Here we show the procedure for the case of Android 5.0.2 and Windows 8 however this procedure is similar across other operative systems and devices.

1. Navigate to Settings - Bluetooth and turn on bluetooth.
2. The phone will automatically scan for nearby devices and will show them on a list. Select your computer from that list.
3. A window will open on showing a pin code, but do not click on "Pair" yet. (fig. B.2)



Bluetooth pairing request

Device

NEVAK-PC

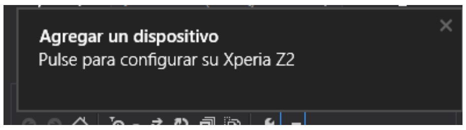
Pairing code

433441

Pairing grants access to your contacts and call history when connected.

Figure B.2: Android pin confirmation.

4. A notification will pop up on your Windows computer asking for permission to allow your phone to connect. Click on it to accept. (fig. B.3)

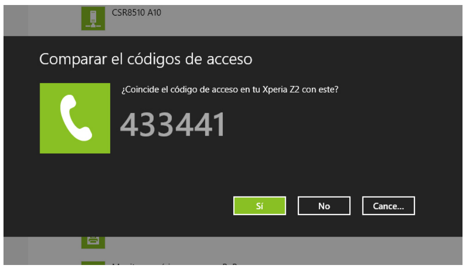


Agregar un dispositivo

Pulse para configurar su Xperia Z2

Figure B.3: Windows 8 connection request.

5. You will now see in your computer a screen with the same pin code that showed up on your phone. Click on "Yes". (fig. B.4)



Comparar el códigos de acceso

¿Coincide el código de acceso en tu Xperia Z2 con este?

433441

Figure B.4: Windows 8 pin confirmation.

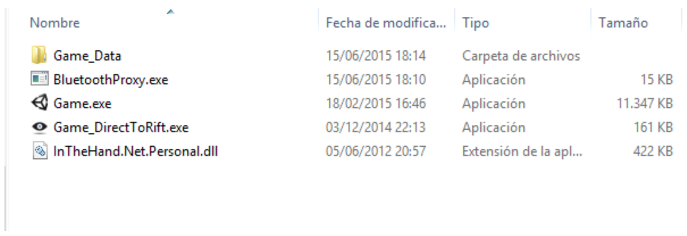
6. Now you can go back to your phone and click on the "Pair" button.

7. After a few seconds your phone and computer will be paired and ready to use the connection.

From now on every time bluetooth is enabled on both devices and they are in range they will automatically get paired.

B.2.3 Game Installation

Once a copy of the game has been obtained and the third party software has been installed it is enough with extracting all the game files to the desired directory as shown in figure B.5.



Nombre	Fecha de modifica...	Tipo	Tamaño
Game_Data	15/06/2015 18:14	Carpeta de archivos	
BluetoothProxy.exe	15/06/2015 18:10	Aplicación	15 KB
Game.exe	18/02/2015 16:46	Aplicación	11.347 KB
Game_DirectToRift.exe	03/12/2014 22:13	Aplicación	161 KB
InTheHand.Net.Personal.dll	05/06/2012 20:57	Extensión de la apl...	422 KB

Figure B.5: Once extracted, the game directory should look like this.

B.3 Playing the Games

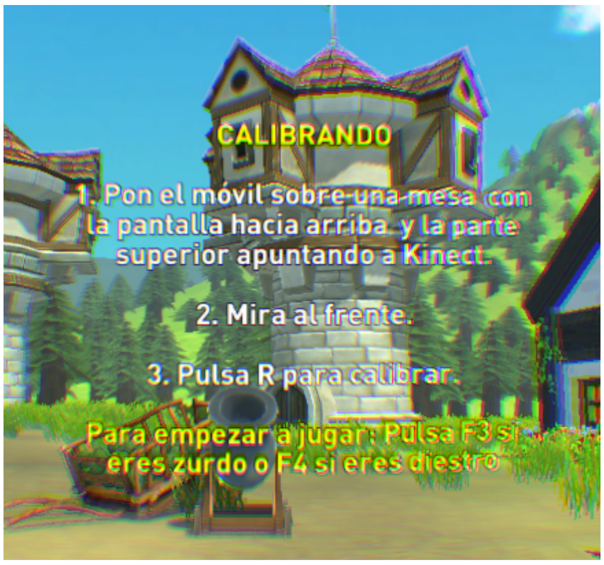
First run the BluetoothProxy.exe application (it should be running while you play the game to use the smartphone as a controller) and then start the BT IMU application in your smartphone. You will see your computer on the list of paired devices inside the application, so select it. Once you do so the phone will start sending data to the computer as it will be shown in the Bluetooth Proxy application ().

Now run the game executable (Game_DirectToRift.exe or Game.exe depending on how your Oculus Rift display mode is configured [37]) and the game will start.

B.3.1 General Controls

These key bindings apply to the three games:

- **Keyboard R:** Reset the virtual camera orientation.
- **Keyboard 1:** Load Sword Training game.
- **Keyboard 2:** Load Pegasus Flight game.
- **Keyboard 3:** Load Pegasus Ride game.


CALIBRANDO

1. Pon el móvil sobre una mesa con la pantalla hacia arriba y la parte superior apuntando a Kinect.

2. Mira al frente.

3. Pulsa R para calibrar.

Para empezar a jugar: Pulsa F3 si eres zurdo o F4 si eres diestro

Figure B.6: Sword Training calibration screen.

B.3.2 Sword Training Controls

At the calibration screen (figure B.6):

- **Keyboard R:** Reset sword* and camera orientation.
- **Phone movement:** Control the sword.
- **Keyboard F3:** Start the game as a left handed player.
- **Keyboard F4:** Start the game as a right handed player.

*For the calibration of the sword and phone orientation simply leave the phone on a flat surface such that it points towards the kinect's sensor plane. See figure (B.7).

In game:

- **Keyboard R:** Reset sword and camera orientation.
- **Phone movement:** Control the sword.
- **Keyboard 1:** Restart the game.

B.3.3 Pegasus Flight Controls

At the calibration screen (figure B.8):

- **Keyboard R:** Reset camera orientation.
- **Keyboard F3:** Start the game.

In game:

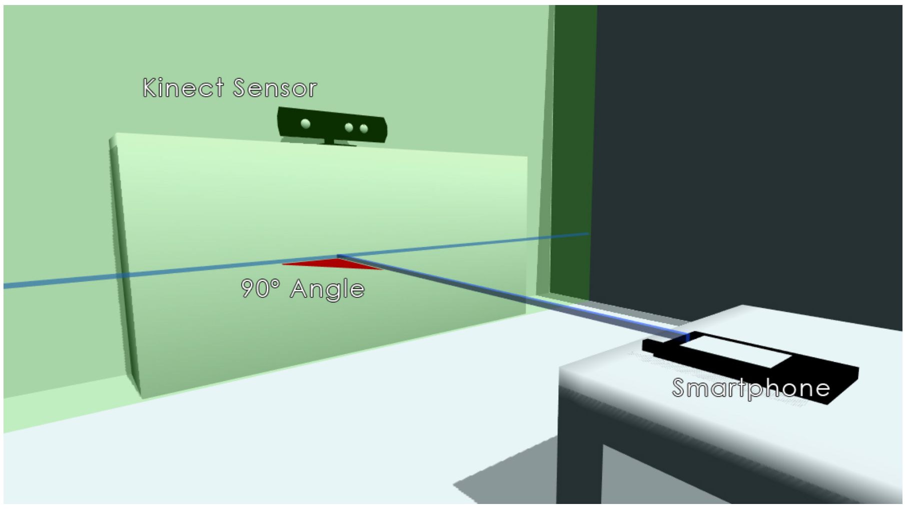


Figure B.7: How to calibrate the smartphone orientation for the Sword Training game. Place it on a flat surface pointing perpendicularly to the Kinect sensor's plane.

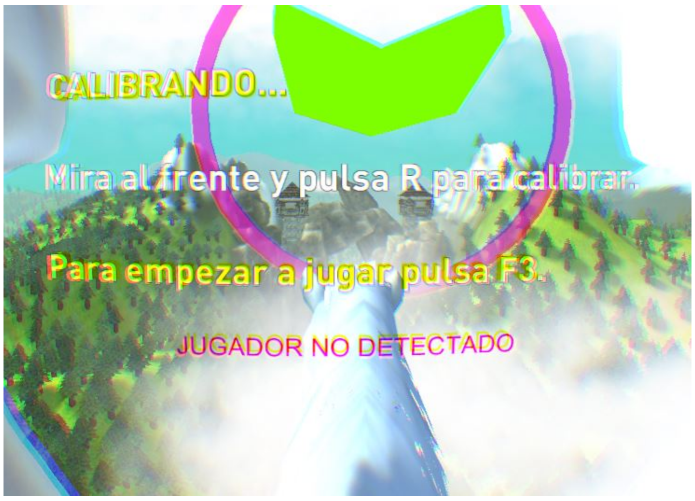

CALIBRANDO...
Mira al frente y pulsa R para calibrar.
Para empezar a jugar pulsa F3.
JUGADOR NO DETECTADO

Figure B.8: Pegasus Flight calibration screen.

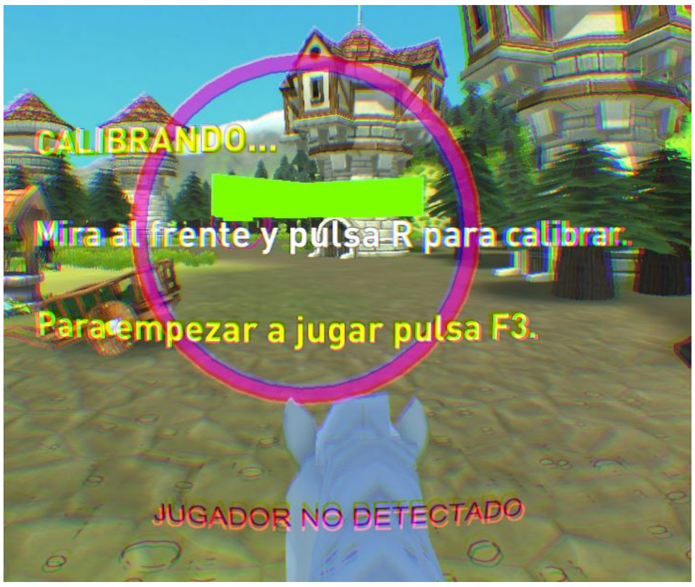
- **Keyboard R:** Reset camera orientation.
- **Keyboard Tab:** Reset the player position to the last passed way point.
- **Arms movement:** Control the Pegasus speed and vertical lift force. If you move your arms in such a way that your hands go higher than your shoulders the Pegasus will get an impulse proportional to the amplitude of this movement. If you keep your arms extended you will glide, otherwise if you get your arms close to your body the gravity will make you start falling.
- **Head movement:** Controls the Pegasus direction of movement such

that it will move towards the direction you are looking.

- **Keyboard 2:** Restart the game.

B.3.4 Pegasus Ride Controls

At the calibration screen:


CALIBRANDO...

Mira al frente y pulsa R para calibrar.

Para empezar a jugar pulsa F3.

JUGADOR NO DETECTADO

Figure B.9: Pegasus Ride calibration screen.

- **Keyboard R:** Reset camera orientation.
- **Keyboard F3:** Start the game.

In game:

- **Keyboard R:** Reset camera orientation.
- **Keyboard Tab:** Reset the player position to the last passed way point.
- **Hands movement:** Control the Pegasus speed such that if you move your hands up and down close to each other the Pegasus will gain speed proportionally to the amplitude of the movement.
- **Torso and shoulders movement:** Controls the Pegasus direction of movement.
- **Keyboard 3:** Restart the game.

Appendix C

Developer Manual

This part of the manual explains how to tweak some of the game parameters to modify the player experience as well as how to create new levels. Of course more features can be modified or added at source code level, however describing that process is out of the scope of this manual.

C.1 System and Software Requirements

Apart from Unity Editor no other software is required to modify the game as described in this manual. The standard system requirements to install Unity Editor are [36]:

- OS: Windows XP SP2, 7 SP1, 8; Mac OS X 10.8+. *Windows Vista is not supported; and server versions of Windows and OS X are not tested.*
- GPU: Graphics card with DX9 (shader model 2.0) capabilities. Anything made since 2004 should work.

The Unity Editor version recommended is **4.6.3f1** since it is the one used during development.

C.2 Modifying the Sword Training Game

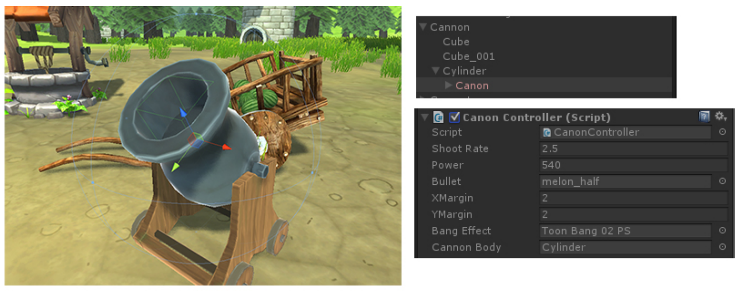
C.2.1 Adjusting the Cannon Parameters

By adjusting the cannon parameters one can greatly affect the difficulty of the game. These adjustments are made via the CannonController script. You will find this script inside the Cannon GameObject hierarchy (see figure C.1 at the top right). Here is a list of what can be tweaked:

- **Shoot Rate:** Time in seconds between shots. The smaller the value the faster the cannon will shoot.
- **Power:** The force that the cannon will apply to the projectiles. The higher the value the faster and further away the watermelons will go.
- **X Margin:** Amount of random deviation for the projectiles in the horizontal axis.

- **Y Margin:** Amount of random deviation for the projectiles in the vertical axis.

Additionally the cannon inclination can be modified to increase or decrease the falling time of the watermelons which will in turn affect the difficulty.



```
▼ Cannon
    Cube
    Cube_001
  ▼ Cylinder
    ▶ Canon
```

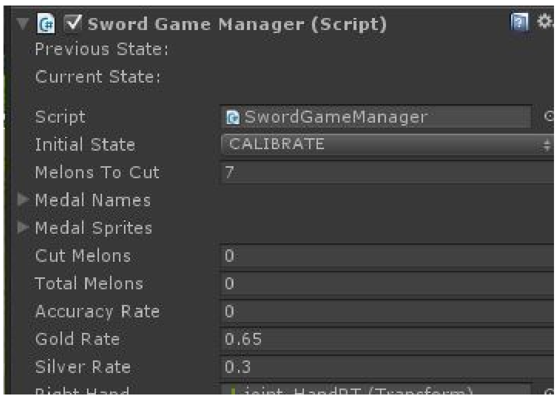
```
▼ Canon Controller (Script)
  Script          CanonController
  Shoot Rate      2.5
  Power           540
  Bullet          melon_half
  XMargin         2
  YMargin         2
  Bang Effect     Toon Bang 02 PS
  Cannon Body     Cylinder
```

Figure C.1: Left: The cannon GameObject. Top right: Cannon hierarchy. Bottom right: Exposed CannonController parameters.

C.2.2 Adjusting General Parameters

The _GameManager GameObject has a SwordGameManager script attached that controls the global game logic and score calculations. The relevant parameters that can be modified are:

- **Melons To Cut:** The goal number of watermelons that the player has to cut to complete the game.
- **Gold Rate:** The minimum accuracy (melons cut to melons shoot ratio) to obtain a gold medal.
- **Silver Rate:** The minimum accuracy to obtain a silver medal. Any accuracy ratio lower than this value will be awarded a bronze medal.



```
▼ Sword Game Manager (Script)
  Previous State:
  Current State:

  Script            SwordGameManager
  Initial State     CALIBRATE
  Melons To Cut     7
▶ Medal Names
▶ Medal Sprites
  Cut Melons        0
  Total Melons      0
  Accuracy Rate     0
  Gold Rate         0.65
  Silver Rate       0.3
```

Figure C.2: SwordGameManager exposed parameters.

C.3 Modifying Pegasus Flight and Ride Games

Due to the similarity in the scene setup between this two games the possible modifications are the same.

C.3.1 Modifying or Creating a New Circuit

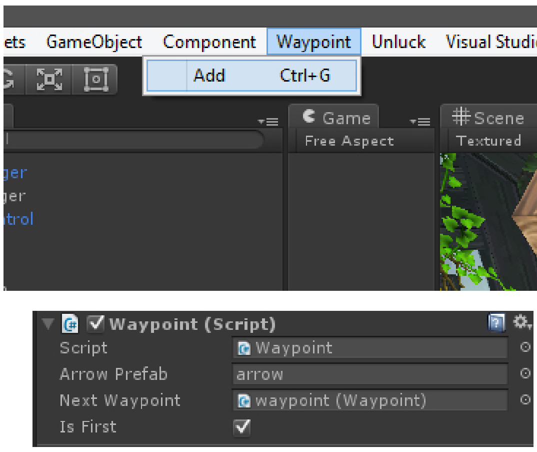
The circuit is formed by the linked way points that have to be traversed in order. During the development some simple editor scripts were written in order to make the circuit editing process easier.

You can:

- Modify the existing circuit by moving the way points (located inside the WAYPOINTS GameObject).

- Add new way points via the menu option (see figure) or by pressing the shortcut "Ctrl G". If you have a way point selected at the moment of adding a new one it will be automatically linked to the old one. This way you can extend the circuit as you like.

- Create a new circuit. To do so just start adding new way points as explained above and set the starting way point by checking its **Is First** field (see figure C.3 at the bottom). This will be indicated with a white sphere. Any way point that does not have a **Next Waypoint** is considered the goal. You can also delete the old way points completely.



Waypoint (Script)

Script	Waypoint
Arrow Prefab	arrow
Next Waypoint	waypoint (Waypoint)
Is First	✓

Figure C.3: Way point menu at the top and way point parameters at the bottom.

To make sure that the circuit is set up correctly, check that all the way points are linked by a white line in the scene window (see figure C.4). If this is not the case you can fix the problem by manually setting the **Next Waypoint** field of the broken way point in the editor (see figure C.3).

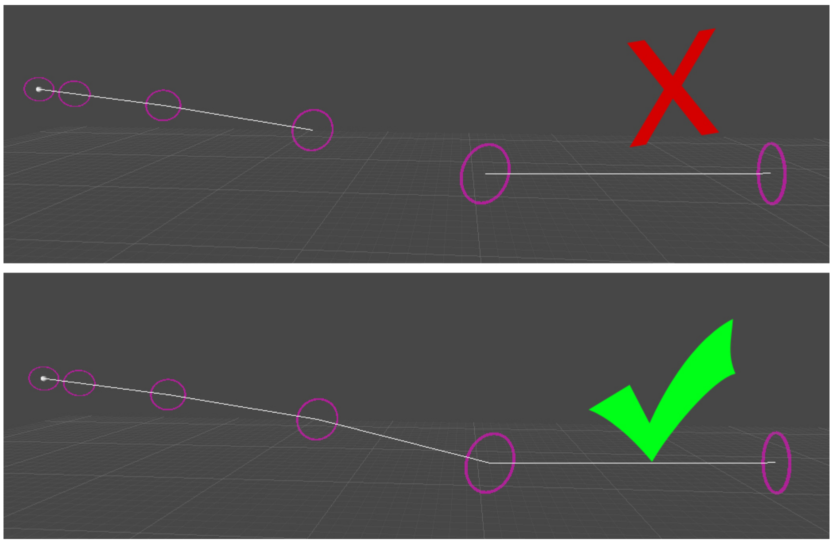
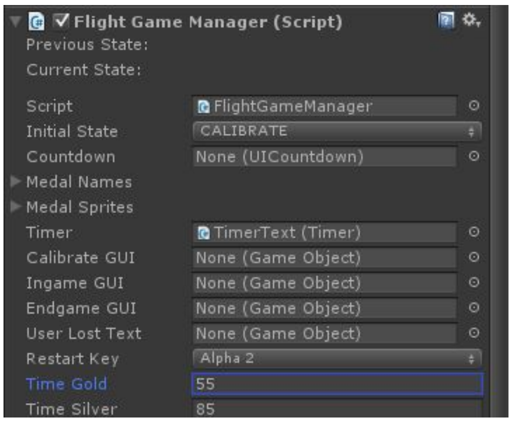


Figure C.4: Possible way point connection issues.

C.3.2 Adjusting General Parameters

Similarly to Sword Training, in these two games you can also adjust the game rewards by changing the values in the FlightGameManager script attached to the _GameManager object (figure C.5):

- Time Gold: Minimum time in seconds to achieve a gold medal.
- Time Silver: Minimum time in seconds to achieve a silver medal. Any time greater than this value will be awarded a bronze medal.



Flight Game Manager (Script)

Previous State:

Current State:

Script	FlightGameManager
Initial State	CALIBRATE
Countdown	None (UICountdown)
Medal Names	
Medal Sprites	
Timer	TimerText (Timer)
Calibrate GUI	None (Game Object)
Ingame GUI	None (Game Object)
Endgame GUI	None (Game Object)
User Lost Text	None (Game Object)
Restart Key	Alpha 2
Time Gold	55
Time Silver	85

Figure C.5: Exposed parameters for the FlightGameManager script.

Appendix D

Image rights agreement document

Universidad Carlos III de Madrid

AUTORIZACIÓN- CESIÓN GRATUITA DERECHOS DE IMAGEN Y DATOS ENTREVISTAS

D. ...., mayor de edad, con plena capacidad de obrar, con D.N.I.........................., y domicilio en ...., actuando en nombre y representación de su hijo/a D/ña. ....

AUTORIZA a la Universidad Carlos III de Madrid (UC3M), entidad sin ánimo de lucro, a realizar un reportaje audiovisual incluyendo encuestas y entrevistas sobre la actividad **SIMULADORES IT (PhyMEL)** en el que aparezca una imagen de la persona de su hijo/a, o su persona. El reportaje podrá ser divulgado en cualquier medio (TV, prensa, internet, CD, DVD, Tecnología móvil, nuevas redes de distribución, material promocional, revistas y congresos científicos, etc.) en cumplimiento de sus finalidades de investigación, formación, divulgación, exposición, promoción, y/o publicidad. Esta cesión se hace por el plazo máximo que permite la legislación vigente para todo el mundo, con posibilidad de ceder a terceros. La obra podrá ser fraccionada o modificada respecto a la información capturada para la elaboración de nuevos materiales.

La presente autorización es completamente gratuita, y me comprometo a no reclamar ninguna compensación y/o pago, y/o reembolso, y/o indemnización a cambio del permiso acordado con la Universidad Carlos III de Madrid para la utilización de mi imagen.

Los datos en poder de la Universidad Carlos III de Madrid, serán custodiados de conformidad con lo dispuesto en la Ley Orgánica de Protección de Datos de Carácter Personal.

Madrid, a ….. de ………………… de …………..

D. ....

NOTA: En la actividad se hará uso de gafas de realidad virtual. No está recomendado su uso en caso de que el participante tenga algún problema de epilepsia.

Acronyms

API Application Programming Interface. 23, 25

CC0 Creative Commons Zero. 22

EULA End User License Agreement. 22

HMD Head-mounted Display. 1, 79

IDE Integrated Development Environment. 24, 25

IMU Inertial Measurement Unit. 8, 15, 26, 27, 80

SDK Software Development Kit. 26, 27

TLT IEEE Transactions on Learning Technologies. 66

VR Virtual Reality. 1–3, 5, 7, 26, 34, 66, 75, 78–81, 83, 85

Bibliography

[1] S. Bryson, "Approaches to the successful design and implementation of vr applications," *Virtual reality applications*, pp. 3–15, 1995.

[2] S. K. Ong and A. Y. C. Nee, *Virtual and augmented reality applications in manufacturing.* Springer Science & Business Media, 2013.

[3] A. A. Rizzo, M. Schultheis, K. A. Kerns, and C. Mateer, "Analysis of assets for virtual reality applications in neuropsychology," *Neuropsychological Rehabilitation*, vol. 14, no. 1-2, pp. 207–239, 2004.

[4] B. Peñasco-Martín, A. De los Reyes-Guzmán, Á. Gil-Agudo, A. Bernal-Sahún, B. Pérez-Aguilar, and A. I. De la Peña González, "Aplicación de la realidad virtual en los aspectos motores de la neurorrehabilitación," *Rev Neurol*, vol. 51, no. 481, p. 8, 2010.

[5] C. F. Panadero, V. de la Cruz Barquero, C. D. Kloos, and D. M. Núñez, "Phymel-ws: Physically experiencing the virtual world. insights into mixed reality and flow state on board a wheelchair simulator," *Journal of Universal Computer Science*, vol. 20, no. 12, pp. 1629–1648, 2014.

[6] D. Jonathan, F. Eduardo, and L. Rodrigo, "Tecnologías de interacción avanzadas aplicadas a videojuegos," *Blucher Design Proceedings*, vol. 1, no. 8, pp. 149–152, 2014.

[7] C. Fernández-Panadero and C. D. Kloos, "Phymel. a framework to integrate physical, mental and emotional learning in meaningfull experiences and multidimensional reports," *London, 28-29 November 2013 King?s College London, UK*, p. 203, 2013.

[8] H. Gardner, *Multiple Intelligences: New Horizons.* BasicBooks, 2006.

[9] Y. Hao-Sheng, "Embodied cognition: A new approach in cognitive psychology [j]," *Advances in Psychological Science*, vol. 5, p. 001, 2010.

[10] M. A. Dimyan and L. G. Cohen, "Neuroplasticity in the context of motor rehabilitation after stroke," *Nature Reviews Neurology*, vol. 7, no. 2, pp. 76–85, 2011.

[11] "Real decreto-ley 89/2014, de 24 de julio, por el que se establece para la comunidad de madrid el currículo de la educación primaria.."

[12] R. N. Río, *El secreto de Marcos.* Uno Editorial, 2014.

[13] M.-L. Ryan, *Narrative as virtual reality: Immersion and interactivity in literature and electronic media.* Johns Hopkins University Press, 2001.

[14] M. B. Ibáñez, D. Morillo, P. Santos, D. Perez Calle, J. J. García Rueda, D. Hernández-Leo, and C. Delgado Kloos, “Computer assisted assessment within 3d virtual worlds,” 2011.

[15] I. OCULUS VR, *Best Practices Guide.* [Online; accessed 20-June-2015].

[16] I. Sixense Entertainment, “Sixense stem key components - stem base.” `http://sixense.com/wireless`. [Online; accessed 20-June-2015].

[17] M. Livingston, J. Sebastian, Z. Ai, J. W. Decker, *et al.*, “Performance measurements for the microsoft kinect skeleton,” in *Virtual Reality Short Papers and Posters (VRW), 2012 IEEE*, pp. 119–120, IEEE, 2012.

[18] Sixense, “Evolution of sixense technology: From hydra to stem system.” `https://www.youtube.com/watch?v=gC2pMfoyEiA`. [Online; accessed 20-June-2015].

[19] “Asset Store.” `https://www.assetstore.unity3d.com/en/`. [Online; accessed 20-June-2015].

[20] “Open game art.” `http://opengameart.org/`. [Online; accessed 20-June-2015].

[21] “Public domain dedication.” `https://creativecommons.org/publicdomain/zero/1.0/`. [Online; accessed 20-June-2015].

[22] I. OCULUS VR, “Unity 4 integration.” `http://static.oculus.com/sdk-downloads/ovr_unity_0.4.4_lib.zip`. [Online; accessed 20-June-2015].

[23] R. Solutions, “Kinect with ms-sdk unity package.” `https://www.assetstore.unity3d.com/en/?gclid=CjwKEAjw7YWrBRCThIyogcGymQsSJAAmz_nd6eCbYHmOVGBkNp_IJSEtys8PEfJhhZSdHXBr03ZSUhoCabjw_wcB#!/content/7747`. [Online; accessed 20-June-2015].

[24] Google, “Andoid api - sensorevent.” `http://developer.android.com/reference/android/hardware/SensorEvent.html`. [Online; accessed 20-June-2015].

[25] J. Zwiener, “Wireless imu.” `https://play.google.com/store/apps/details?id=org.zwiener.wimu`. [Online; accessed 20-June-2015].

[26] I. T. H. Ltd, “32feet.net.” `http://32feet.codeplex.com/`. [Online; accessed 20-June-2015].

[27] “Mono project.” `http://www.mono-project.com/`. [Online; accessed 20-June-2015].

[28] “Bluetooth support on win and osx.” `http://answers.unity3d.com/questions/858073/bluetooth-support-on-win-and-osx.html`. [Online; accessed 20-June-2015].

[29] W. Sherman and A. Craig, *Understanding Virtual Reality: Interface, Application, and Design.* The Morgan Kaufmann Series in Computer Graphics, Elsevier Science, 2002.

[30] M. Talbot, "Finite state machines in unity." `https://www.youtube.com/watch?v=l0XY7zwag_g`. [Online; accessed 20-June-2015].

[31] S. LaValle, "Sensor fusion: Keeping it simple," [Online; accessed 20-June-2015].

[32] J.-K. Nielsen, "Meshbuilder.cs." `http://tothemathmos.com/files/MeshBuilder.cs`. [Online; accessed 20-June-2015].

[33] G. Fabian and L. Gergó, "Fast algorithm to split and reconstruct triangular meshes," [Online; accessed 20-June-2015].

[34] P. Lawitzki, "Android sensor fusion tutorial." `http://www.codeproject.com/Articles/729759/Android-Sensor-Fusion-Tutorial`. [Online; accessed 20-June-2015].

[35] "Precios y Planes de Abono a Creative Cloud." `https://creative.adobe.com/es/plans?single_app=photoshop&store_code=es`, 2015. [Online; accessed 20-June-2015].

[36] "Unity system requirements." `https://unity3d.com/unity/system-requirements`. [Online; accessed 20-June-2015].

[37] "Oculus rift display modes." `http://www.glfw.org/docs/latest/rift.html#rift_direct`. [Online; accessed 20-June-2015].