UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

BACHELOR IN COMPUTING ENGINEERING

ACADEMIC INTENSIFICATION:
COMPUTER SCIENCE

FINAL PROJECT

UnRehab: Exergaming-based Platform for the Physical Rehabilitation of Patients

Roberto Plaza Romero

September, 2019

UnRehab: Exergaming-based Platform for the Physical Rehabilitation of Patients

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Tecnologías y Sistemas de Información

TECNOLOGÍA ESPECÍFICA DE
COMPUTER SCIENCE

TRABAJO FIN DE GRADO

UnRehab:
Exergaming-based Platform for the Physical Rehabilitation of Patients

Autor: Roberto Plaza Romero

Director: D. Santiago Sánchez Sobrino

Director: Dr. David Vallejo Fernández

September, 2019

Roberto Plaza Romero

Ciudad Real – Spain

E-mail: Roberto.Plaza@alu.uclm.es
Teléfono: 601124347

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	**VOCAL**	**SECRETARIO**
Fdo.:	Fdo.:	Fdo.:

Abstract

Physical rehabilitation of patients affected by neurological diseases commonly means that both the patient and the therapist are in the same physical space so that the latter can adequately guide the former when carrying out exercises.

With the aim of removing these limitations, in recent years telemedicine tools have been developed to facilitate the remote rehabilitation of patients. In addition to reducing the overall cost of the rehabilitation process, this approach democratizes the access to technological solutions that would otherwise not be available to a large part of the population.

This work proposes a platform that facilitates and motivates the physical rehabilitation of patients through active mini-games, or exergames, that simulate exercises or activities that patients carry out in their daily lives. The platform will integrate an interactive system that guides the patient during the execution of rehabilitation routines and a hardware device for motion capture that avoids the need for the patient to carry additional markers or sensors.

Resumen

La rehabilitación física de pacientes afectados por enfermedades neurológicas supone, comúnmente, que tanto el paciente como el terapeuta se encuentren en el mismo espacio físico para que el segundo pueda guiar al primero adecuadamente a la hora de realizar ejercicios.

Con el objetivo de eliminar estas limitaciones, en los últimos años se han creado herramientas de telemedicina orientadas a facilitar la rehabilitación remota de los pacientes. Además de reducir el coste global del proceso de rehabilitación, este enfoque democratiza el acceso a soluciones tecnológicas que, de otra forma, no estarían disponibles para una gran parte de la población.

En este trabajo se propone una plataforma que facilite y motive la rehabilitación física de pacientes mediante mini-juegos activos, o exergames, que simulen ejercicios o actividades que los pacientes llevan a cabo en su día a día. La plataforma integrará un sistema interactivo que guíe al paciente durante la ejecución de las rutinas de rehabilitación y un dispositivo hardware de captura de movimientos que evite que el paciente tenga que llevar marcadores o sensores adicionales.

Agradecimientos

Este TFG ha sido posible gracias a una improbable cadena de infortunios, por ende, agradezco a cualquier ente creador y/o presencia astral que se haya dado.

Primero me gustaría agradecer a Alf, Marcos y Monescillo, que me ofrecieran su amistad. Después del bachillerato necesitaba amigos y no hay en el mundo nadie mejor que vosotros. Se que no hablamos mucho, pero sois personas alucinantes y ojalá no perdeos nunca la pista.

Gracias a mi familia, padre y madre, por haber confiado en mi, y haberme dado su apoyo (y también por soportar el ruido del teclado a altísimas horas de la madrugada). Gracias a mi hermano por ser a la vez hermano y colega, U Rock Bruh!

Gracias al equipo de desarrolladores de Furious Koalas, por su incansable trabajo por la universidad y por todo lo que han hecho con la gamificación.

Y por último, y tal vez por ello, más importante mis supervisores, por su apoyo, por su paciencia, por su dedicación y por ser capaces de sacar algo útil de mi.

Muchas gracias.

Roberto P. Romero

A la promoción 2018/2019 de Ciencias de Computación.
Por ser la mejor generación.

Contents

List of Tables

List of Figures

List of Listings

List of Acronyms

AI	Artificial intelligence
UI	User interface
IP	Internet protocol
OO	Object Oriented
OS	Operating system
DTW	Dynamic Time Warping
UE4	Unreal Engine 4
FPS	frames per second
SDK	Software Development Kit
CEDV	Curso de Experto en Desarrollo de Videojuegos
FOV	Field of view
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory
USB	Universal Serial Bus
DDR	Double Data Rate
GUI	Graphical User Interface
ISO	International Organization for Standardization
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CV	Computer vision
BSD	Berkely software distribution
API	Application programming interface
VR	Virtual Reality
UWP	Universal Windows platform
IOT	Internet of things

RTS	Real-time strategy game
AR	Augmented reality
IOT	Internet of things
JVM	Java virtual machine
NDK	Native development kid
GPL	General public license
OGRE	Object-oriented graphics rendering engine
UML	Unified modeling language
HTTP	HyperText Transfer Protocol
XML	Extensible Markup Language
IDE	Integrated development environment

Chapter 1

Introduction

PHYSICAL rehabilitation of patients affected by neurological diseases commonly means that both the patient and the therapist are in the same physical space so that the latter can adequately guide the former when carrying out several exercises that will make the patient have a proper rehabilitation.

What is a stroke? A stroke is a "brain attack". It can happen to anyone at any time, it occurs when the blood flow to an area of the brain is cut off. When this happens, brain cells are deprived of oxygen and begin to die. When brain cells die, abilities controlled by the area affected as much as memory and muscle control are lost.

In modern society we still have many bad habits that favour the appearance of stroke. Use of smokes, diabetes, obesity, insufficient sleep and genetics are factors involved in the appearance of this diseases.

How each person is affected by a stroke can vary depending on the type of stroke, the zone of the brain affected, and the physiognomy of the affected person. For example, a light stroke survivor can suffer from weakness in a limb, whereas people who had suffer larger strokes may be permanently **paralyzed on one side of their body** or lose their ability to speak. The ratio of fully recovered patients after suffering a stroke is less than one-third, as presented by the National Stroke Association [2][21].

Strokes may impair or nullify vital functions such as eating, speaking and moving, they can also lead to a loss of control of emotions and sexuality and other symptoms like dementia and personality disorders. In order to manage an adequate recovery there must be guarantee a plan consisted of both, physical and cognitive/emotional activities.

When rehabilitating from stroke, the goal is to help patients to re-learn the skills they lost after the stroke in order to make them regain their quality of life and became **independent** again. As mentioned before, results of stroke rehabilitation can vary widely, nonetheless, most of the research in this topic agrees that people who participate in focused stroke rehabilitation programs **end up with more quality of life along all boards**.

According to a study carried out by the World Stroke Organization[15], the crude rate of stroke chance between 1990 and 2016 is of 185 per 100,000, but the stroke death rate is lower than 75 per 100,000. This diseases affect mainly both, men and women over the age of 44, but there has been a slight increase in child stroke cases.

Global Number of stroke survivors was estimated at nearly 26 Million[1] in 2013 and this number can only increase the rate of strokes worldwide is know to be approximately about 1 every 2 seconds. Also, the proportional contribution of deaths from stroke to all causes of deaths has risen from 9.6% to 11.4% since 1990. Fortunately, as mention before, the stroke fatality rate could be a lot higher.

On the other side, technology an video games share did exponentially grew since the market was established. New generations have video games as their primary hobbies and old generation started accepting them and even enjoying what they have to offer.

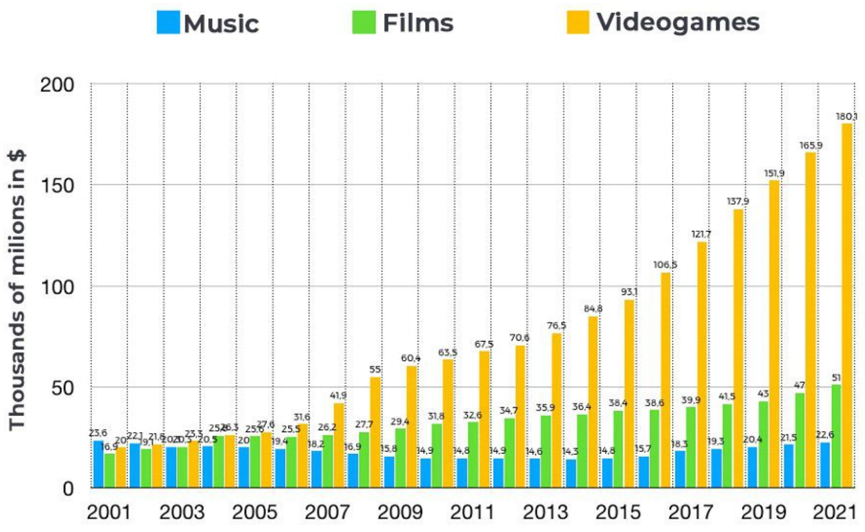


Figure 1.1: Video game, cinema and music shares.

This exponential grow of video game industry allowed it to raise more than the cinema and music industries together. According to "eedar"[4] 70% of the people in US play video games (at least) weekly, 46% of them are women and most of them are played on **mobile platforms**.

[1]Data provided by the study "Global Epidemiology of Stroke with Special Reference to Sub - Saharan Africa"

The ’gamer’ profile has a great variant, but the most common one is what ’newzoo’ calls "The time filler"[14], a profile that only plays video game at social meetings or when they have time to spare. Other major group are the so-called "cloud gamers", users that enjoy playing, but spend the minimum amount of money in their hobby, mostly playing free-to-play games on non game-specific hardware.

According to IGN[20] in an article of the 10 top selling games of the world, the only story-driven game of those 10 was "Grand Theft Auto V" and many of the games are made with the idea of repetitive challenges.

We can also see most of those games are developed by Nintendo and also all single-platform games are developed by Nintendo for Nintendo platforms, being ’Wii’ the most repeated. At last, but not least, we should mention the games "Wii Sports" and "Wii Fit", both are what we called "**exergames**", games that encourage physical activity, and this is important, because Nintendo is doing a good job trying to democratize the world of console games, specially bringing them to older people.

Figure 1.2: Wii advertisment with old people having fun.

As we see video games are becoming more and more popular, and so is, the idea of incorporating video game mechanics to every day life activities, including rehabilitation of stroke patients.

Video games can provide enough brain stimuli to help patients with their rehab journey. Not just in terms of physical activities, thanks a brand new wave of movement-based controllers, but thanks to the colourful environments and spatial positioning of those. The idea of

challenge is also important because it may help some patients to develop tougher personalities in order to overcome the harder parts of the rehabilitation or even a personality disorder.

The use of tabletop games is quite common on the rehab plans due to their light-to-moderate brain exercise and that's why so often we can find in those programs games like Scrabble, Jenga, Checkers, Poker or Uno.

A common trend in modern-day game development is making them more accessible, this could be achieved via more ergonomic controllers or by means of simplifying the game mechanics (Keep in mind old NES controller had **8 buttons** and modern controllers have **12 buttons 2 joysticks and 2 triggers**).

There has been several experiments of including video games in rehabilitation plan, most of the documented in the papers: "Serious games for upper limb rehabilitation following stroke"[7] and "Video Games and Rehabilitation: Using Design Principles to Enhance Engagement in Physical Therapy"[11], but, sadly **there are not many games completely suited for rehabilitation** or developed with rehabilitation in mind.

Also, the introduction of new interaction paradigms like Virtual Reality (VR) and Augmented reality (AR) may help with some initial rejection towards the video games, in the case of the AR thanks to it's more natural approach ant in the case of VR thanks to the possibility of creating any world.

The use of technology in health-related issues is common nowadays, and thanks to modern technology we can create movement-tracking systems and games using the extracted information using a single device that takes less that $700cm^3$ (0.7 litters) of space.

For this reasons we believe that a system based in the gamification of rehabilitation is a good idea, specially since rehabilitation is not always carried out correctly.

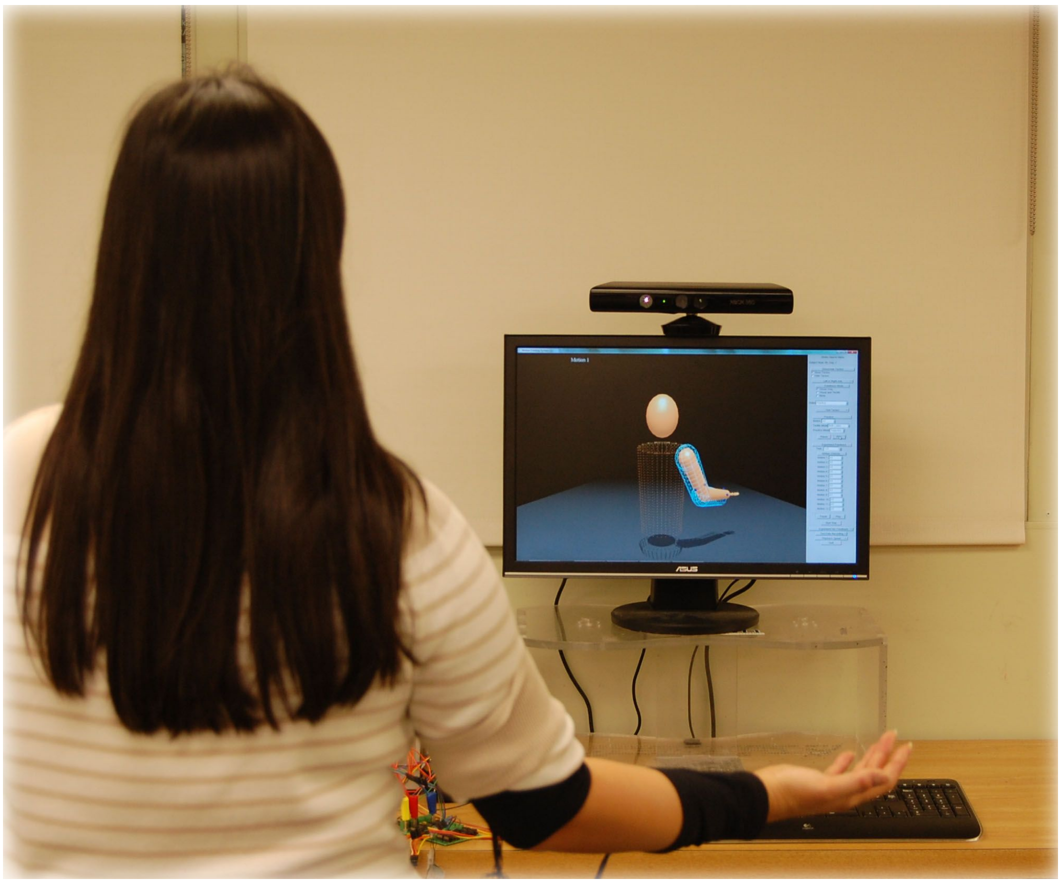


Figure 1.3: Rehabilitation based game.

Chapter 2

Objectives

IN this chapter we will define the general objectives of the system alongside some desirable features for the system.

2.1 UnRehab

"UnRehab" is the name given to the proposed platform for the patient rehabilitation. The aim of this project is not only to**design a scalable architecture that allows the integration of new rehabilitation exercises that motivates and encourages patients** but also to try to **keep the hardware deployment as reduced as possible**.

2.2 Overview of the objectives

With the aim of removing the limitations of common rehabilitation techniques, in recent years telemedicine tools have been developed to facilitate the remote rehabilitation of patients. In addition to reducing the overall cost of the rehabilitation process. This approach democratises the access to technological solutions that would otherwise not be available to a large part of the population.

The object of this work is to create a platform that **facilitates and motivates the physical rehabilitation of patients through active mini-games**, or exergames, **that simulate exercises or activities that patients carry out in their daily lives**. The platform will integrate an interactive system that guides the patient during the execution of rehabilitation routines and a hardware device for motion capture that avoids the need for the patient to wear additional markers or sensors.

2.3 Specific objectives

This platform ready to **facilitate the rehabilitation of stroke patients** through the use of gamification will need to:

- **Reject the use of traditional controllers.** Traditional computing is known for being old-fashioned, counter-intuitive, and frustrating. The use of modern technologies brings the possibility to explore interaction paradigms that are more natural to the user.

- **Possibility to include new games.** The application should allow the fast development and integration of the new mini-games. The game engine decision should be of a great importance for this sub-objective since most of them are either scene-oriented or level-oriented and every mini-game should be one of those.
- **Small, portable hardware infrastructure.** The hardware needed to run the architecture should be as reduced as possible. We will use a TVico as a standalone device, this reduces the hardware needed too a display and the previously mentioned device, not even internet connection should be needed.
- **Medium-to-low activity collection of exercises.** The idea is not to saturate the user with game-imposed mechanics and allow a rehabilitation in the firs stages after the stroke.
- **Intuitive, user-friendly UI.** The user should be available to understand what to do in every moment without tutorials or intrusive messages to avoid counter-intuitive controls. See figure 2.1.
- **Movement-based control system.** Related to the second point. The whole set of controls of the game should be given by the use of the body, no external device remote, command or controller should be used in order to navigate nor play the games.

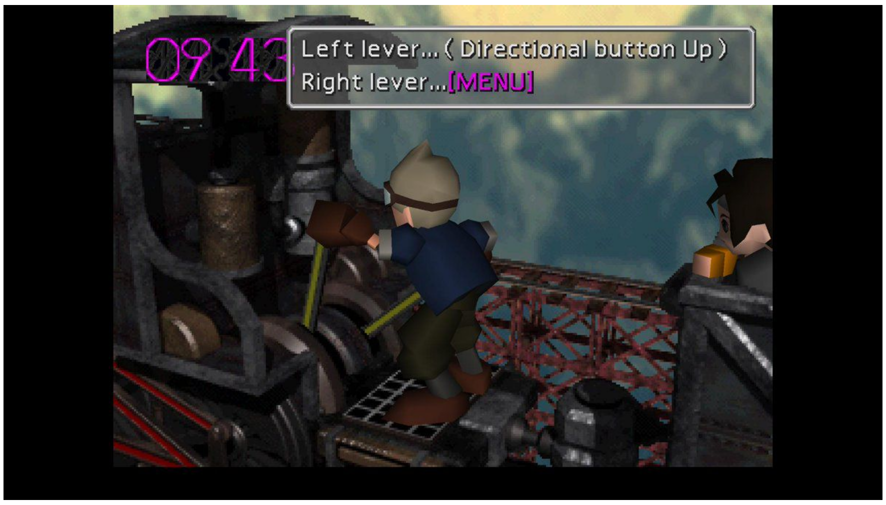


Figure 2.1: Example of a bad interface

Chapter 3
State Of Art

DURING this chapter we will discuss how far technology has come in terms of pose and gestures recognition as well as gamification and exergames. In the later pages we will also see a set of technologies, both hardware and software, related to this concerns.

3.1 Automatic evaluation of physical movement.

Extracting information of images is a discipline usually called CV (Computer vision), and it relies on the increment of computational power as much as in the refined techniques of machine learning but with modern hardware and some tweaks to some algorithms the possibility to analyse image in real time (and therefore video) is a reality.

When we speak about any kind of image recognition technique in reality we are speaking about artificial neural networks, more specifically convolutional neural networks.

Artificial Neural Network are a kind of machine learning algorithms and its theoretical approach exists since the early 40's, to put simply, this algorithm is a graph of "neurons". This neurons are just a thing that stores value, value called "activation". Commonly neural networks follow a layered architecture and the activation values of one layer affect the activation values of the next, usually there will be weights and biases assigned to these connections to let some neurons be more important than other in some scenario. This structure is called a perceptron, or multilayered perceptron and was introduced in 1958 by F. Rossenblatt in the article "The perceptron" [18].

These biases and weights should be different for every network, the process of finding the best values for every bias and every weight is called training the network and it is the main reason why this algorithm is that heavy at a computational level, it requires a lot of examples and a lot of time in order to acquire the optimal layout for a particular network.

If you formalize this computational model it will resemble a common linear regression, commonly used in descriptive statistics. In order to process image we need more than a

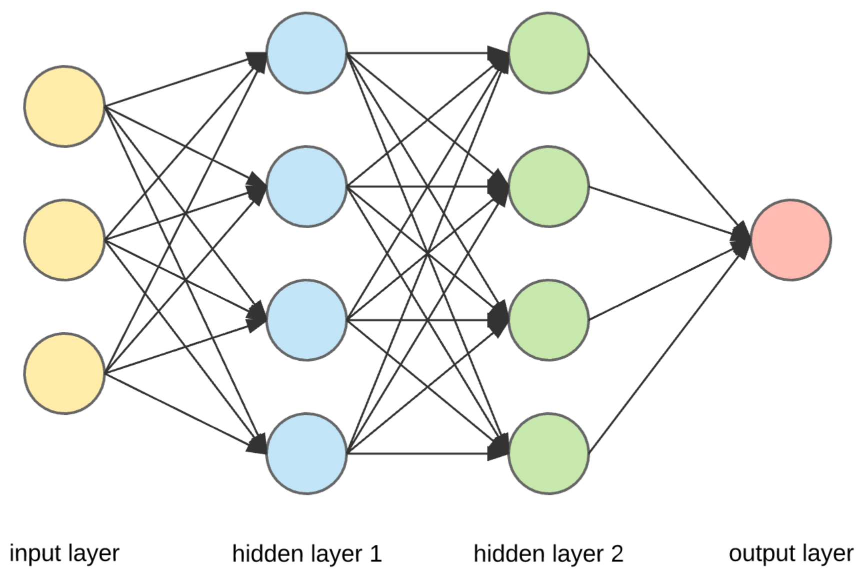


Figure 3.1: Architecture of a Multilayered Perceptron.

linear regression, therefore, the Convolutional Neural Networks allow to apply a convolution operation to the data in order to make it easier to recognise patterns, most commonly edges because they are the most useful pattern that can be computed.

The development and training of a Convolutional Neural Network that allows less complicated tasks like tracking human faces can be tremendously expensive and time consuming. Thankfully there are several Artificial Neural Network frameworks in the market and they have proven to work flawlessly.

3.1.1 Common Techniques.

This is a compilation of the most popular CV techniques out there.

OpenCV and algebra libraries.

OpenCV is probably the most popular CV tool out there. It is the most *close-to-the-metal* approach to Computer vision. In essence OpenCV lets you transform certain image formats to color grids in order to analyse them, this meaning, you will need to code up the algorithms if you want to analyse anything using OpenCV [17].

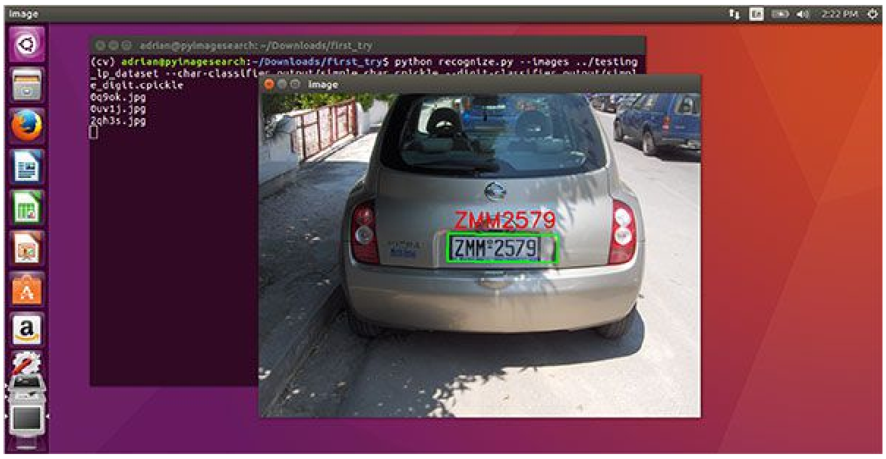
On the other hand, OpenCV is written in C++ and has the binds to use them with some of the most popular programming languages, making it the, arguably, most portable and flexible solution. You can use OpenCV with Python (2.7 and 3.X), C (if you compile the sources using C linkage), C++ (out of the box), Java (with Android support) and JavaScript (in the latest version).

Most of the following environments are build with OpenCV, because it runs under a BSD license[1], meaning it is free for both, academic and commercial usages.

Author's recommendation is to use OpenCV to extract information and then treat it with a CNN coded in CUDA using cuDNN if needed. CUDA is a C-like language developed by NVIDIA that allows the user to use the of any GPU of any brand for computation purposes, the only expense in this is that code in foreign devices can not be recursive due to the lack of a program counter, cuDNN by its side is a machine learning algorithm library for CUDA.

Unfortunately CUDA needs a CUDA driver, and whereas this driver has at least one version for every operating system it can conflict with your current graphic driver and complicate the task to build up a correct development environment.

In the end what OpenCV offers us is a highly **flexible, highly portable and fast** CV library at the expense of not having pre-built algorithms or trained CNN models.



```
(cv) adrian@pyimagesearch:~/Downloads/first_try$ python recognize.py --images ../testing
_lp_dataset --char-classifier output/simple_char.cpickle --digit-classifier output/simpl
e_digit.cpickle
0q9ok.jpg
0uv1j.jpg
2qh3s.jpg
```

Figure 3.2: Example of OpenCV.

Python TensorFlow.

Community's favorite machine learning environments is TensorFlow. TensorFlow is an open-source [2] library written in C++ and Python with some CUDA plugins for the programming language Python and recently to C++. TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy [16].

[1]OpenCV repository: `https://github.com/opencv/opencv`
[2]TensorFlow repository: `https://github.com/tensorflow/tensorflow`

One of the best things about TensorFlow is its community. So many people use TensorFlow and any doubt could be easily solved via community forums, similarly you can find many tutorials and documentation through youtube channels and cheap, or even free courses.

Another good thing about TensorFlow is the ease of setup. The easiest way to setup a development environment for TensorFlow is using python-pip, the packet manager of python. In order to install pip in Windows installing python from the official website [3] should also get you PIP.

The pose estimation solution that TensorFlow offers consist on a CNN with an input layer of at best 29x29 pixels times 17 gray-scale colours, this may provide a *good enough set of tools* for most of the situations.

Summing up, TensorFlow **is probably the greatest AI (Artificial intelligence) platform** in term of user base, is the easiest environment to install and work with by far and has a Python stub. However, the downside is that the default solution for pose estimation/movement tracking is very limited.

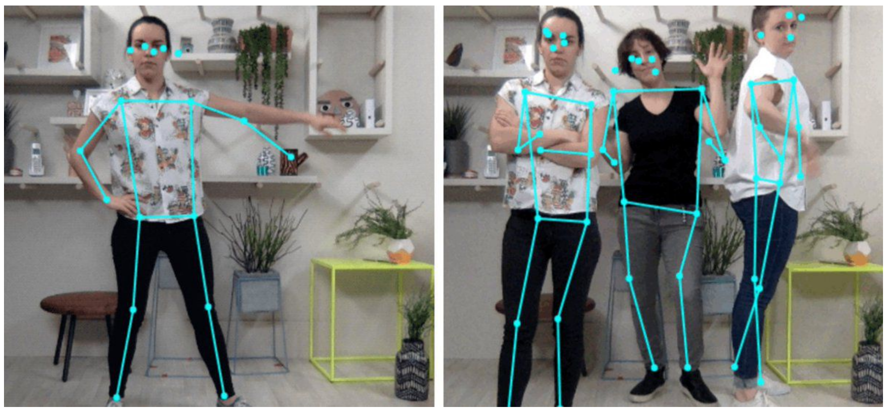


Figure 3.3: TensorFlow example.

OpenPose.

Despite not being as known or famous as TensorFlow or OpenCV, OpenPose is an interesting piece of software that brings up very different tools together. Besides being written in C++, OpenPose uses libraries coming from OpenCV, Caffe (A Berkeley-developed CNN), and even some modules from TensorFlow.

OpenPose[4] is both, a program and a set of libraries prepared to track skeleton, face and hand keypoints, and at this point is the only one that is not free for commercial purposes.

[3]Python's site: `https://www.python.org`
[4]OpenPose at Github: `https://github.com/CMU-Perceptual-Computing-Lab/openpose`

An example of the use of OpenPose for rehabilitation purposes can be found at [10]. This article gathers the results and accuracy of a CV driven rehabilitation experiment over patients during ACL (Anterior cruciate ligament) reconstruction.

The biggest downside about OpenPose is its poor performance. In order to test the library. The test were made in a system with an Intel Core i5-8250U and a Nvidia GeForce GTX 1050, the performance output was roughly 3FPS, could be boosted up to 5 if tweaked in order to track just one skeleton.

The other mayor downside is its closed build environment. CUDA, despite being fast and convenient, has several problems with its own drivers and OpenPose uses CUDA for GPU acceleration, that makes OpenPose only available to Windows 8 and 10, Mac OSX, Nvidia TX2 and Ubuntu 14.04 or 16.04. Sadly, since Ubuntu 14.04 is now out of support, Mac OSX doesn't allow GPU acceleration and Windows builds can't import the generated libraries due to the construction tool used, the only real OS alternative is Ubuntu 16.04, reason why OpenPose offers several shell scripts that make the installation easier.

The main way to use OpenPose is to build the program and tweak it via command line flags, because OpenPose has plenty[5]. If you know what inputs to give, OpenPose can either treat one image, one video, take a camera connected to the computer and track in real time, save the output media to video or image, track hands, face, skeleton, all of 3 or even 'feed' it an image and make the program treat it, and save the results to a file.

The interesting part comes with the library. OpenPose uses CMake as its construction tool. That means that generating a library is not difficult, but you can only try this method with all the plugins if you run OpenPose in Ubuntu 16.04, as previously said. These libraries allow you to perform easily every everything the compiled version can, plus, it gives you access the rotation and position of all the key-points the library tracks.

On the other Hand, OpenPose, has the highest CNN resolution of all the software we have and will see. The net resolution can reach a size of 1312x736 full RGB pixels. This resolution allows to trap up to 25 different key-points without taking into account hands or face.

OpenPose can also output a json file with the record of tracked joints through an execution.

[5]Main to be founded at:https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/demo_overview.md#main-flags

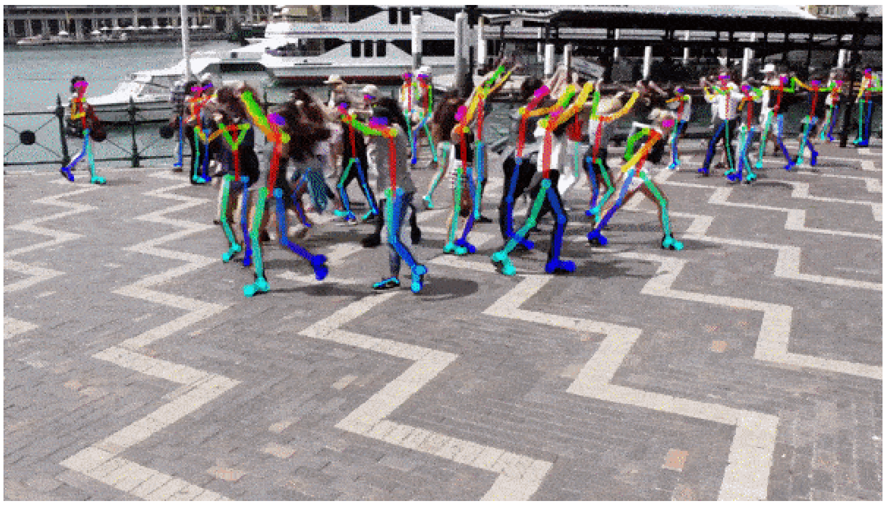


Figure 3.4: OpenPose output example, without face nor hand tracking.

This library offers us a fully trained CNN for a lot of use cases that can be accessed via C++ python and C# at the expense of a very closed development environment and lo performance.

Nuitrack.

The software we are going to use through this project will be Nuitrack. Nuitrack is a proprietary API for skeletal tracking and gesture recognition developed by 3DiVi. It is supports many sensors that currently on sale such as Intel RealSense D415/D435, ASUS Xtion2, Orbbec Astra as claims is not sensor dependent. It is a powerful choice as a human pose estimation middleware and can be easily used with all these sensors. Nuitrack enjoys also a gesture tracking plugin and a hand tracking plugin.

There is a trial license that allows the developer to use the API for up to 3 minutes straight before shutting down. If you wish to keep the middleware running for longer the license pricing goes from 40 to 100 dollars depending on the license type and warranties.

Despite a license being needed you can download the Nuitrack SDK easily just submitting a form in their web page [6].

Nuitrack is self-promoted as being compatible with C++, C# and with plugins for Unity3d and Unreal Engine 4. Sadly, most of the developers had problems when used by Unreal Engine 4 in Android platforms.

The architecture of Nuitrack is composed by 6 different sub-modules:

[6]Nuitrack web page: `https://nuitrack.com`

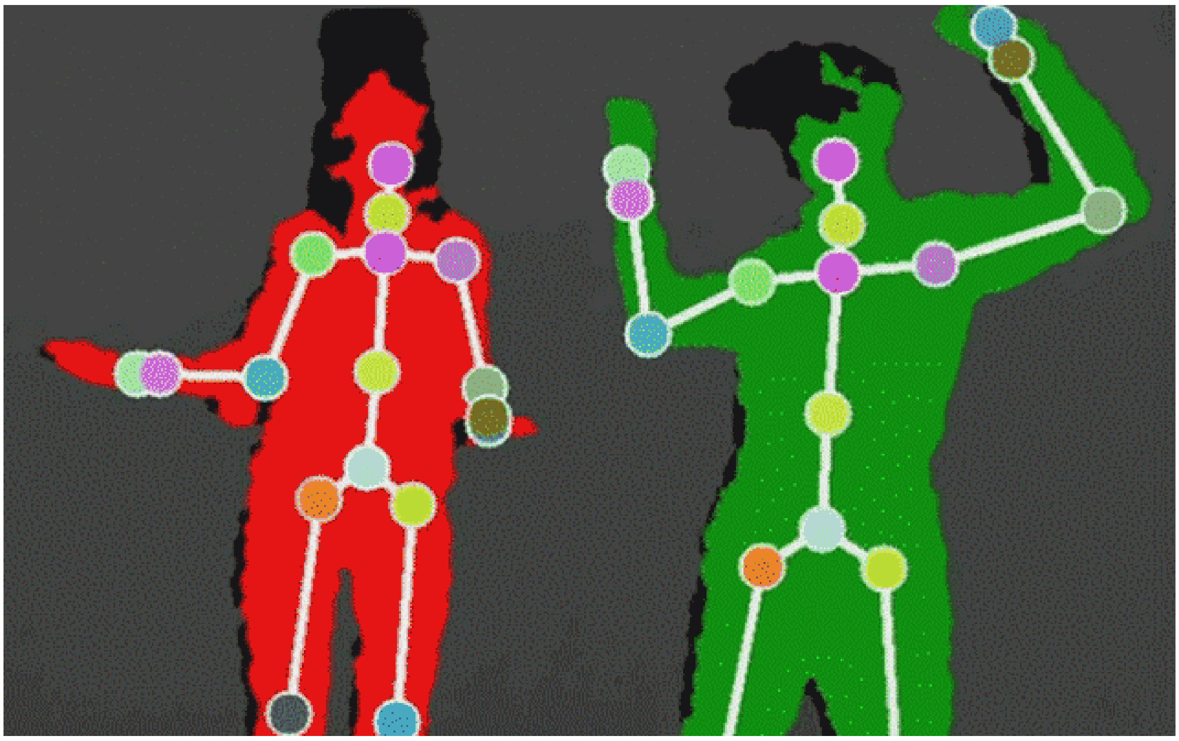


Figure 3.5: Nuitrack example.

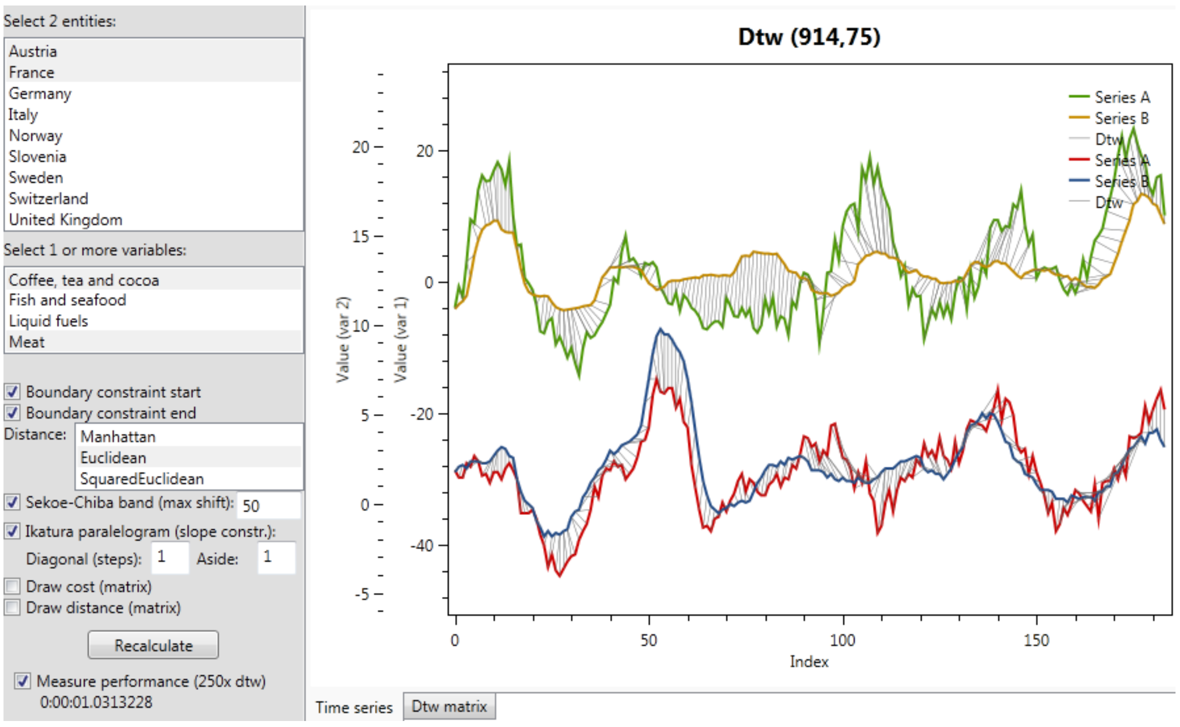
- Depth Sensor, that tracks depth data.
- Colour Sensor, so the user can access colour data.
- User Track, that stores information regarding users.
- Hand Tracker, because hands need a custom CNN.
- Gesture Recogniser, that recognises up to 6 different gestures.

In the end, the only downside of Nuitrack is its need for license, for the rest it has proven to be a reliable, effective and fast solution.

3.1.2 Time series.

So far we haven't speak about video analysis, but about image analysis. This is so because video is not a different media. The fast succession of images is what makes up a video, therefore theoretically video can be analysed by means of image analysis and treat it as a time series problem.

We can see an early approach to the use of time series analysis for human movement patterns and proportions in 2004 in the paper[7]. The object of the article was to create an algorithm to estimate human gestures based on epipolar geometry.


Select 2 entities:

Austria
France
Germany
Italy
Norway
Slovenia
Sweden
Switzerland
United Kingdom

Select 1 or more variables:

Coffee, tea and cocoa
Fish and seafood
Liquid fuels
Meat

[✓] Boundary constraint start
[✓] Boundary constraint end
Distance: Manhattan / Euclidean / SquaredEuclidean
[✓] Sekoe-Chiba band (max shift): 50
[✓] Ikatura paralelogram (slope constr.):
Diagonal (steps): 1 Aside: 1
[ ] Draw cost (matrix)
[ ] Draw distance (matrix)

[✓] Measure performance (250x dtw)
0:00:01.0313228

Figure 3.6: DTW Time series analysis.

Time series analysis can be done with traditional statistics, but some refined algorithms had proven to be better and more efficient that those ones. Through this project We are going to use one particular algorithm called Dynamic Time Warping (DTW).

Dynamic Time Warping.

The Dynamic Time Warping is an algorithm used for time series analysis that shows the similarity between two different time series. This algorithm has two particularities, being the first one a greedy approach, the DTW does not get the best fit as the result but doesn't worsen the worst case scenario.

The second one is that the algorithm is somewhat lag-proof, where other algorithms drastically fail when the two time series are the same but lagged the DTW just removes a bit of the similarity.

Given 2 time series of length N and M DTW has a time complexity of $O(N * M)$ and a space complexity of $O(N * M)$, there are several ways to improve performance, but there are of no concern for this writing.

```
Data: 2 time series N and M
Result: Difference between the 2 time series
Create Table[length(N) + 1][length(M) + 1];
Cost = 0 ;
for i = 0 to length(N) + 1 do
    Table[i][0] = infinite;
end
for j = 0 to length(M) + 1 do
    Table[0][j] = infinite;
end
Table[0][0] = 0;
for i = 0 to length(N) do
    for j = 0 to length(M) do
        Table[i][j] = distance(N[i], M[j])
        + min(Table[i-1][j-1], Table[i][j-1], Table[i-1][j]);
    end
end
Create Position;
Position.x = length(N);
Position.y = length(M);
while Position is not (0, 0) do
    if Table[Position.x - 1][Position.y - 1] < Table[Position.x][Position.y -1] AND
     Table[Position.x - 1][Position.y - 1] < Table[Position.x - 1][Position.y ] then
        Position NextPosition = [Position.x - 1, Position.y - 1];
    else
        if Table[Position.x][Position.y - 1] < Table[Position.x-1][Position.y] AND
         Table[Position.x - 1][Position.y - 1] < Table[Position.x - 1][Position.y ] then
            Position NextPosition = [Position.x, Position.y - 1];
        else
            Position NextPosition = [Position.x - 1, Position.y];
        end
    end
    Cost += Table[NexPosition];
    Position = [NextPosition.x, NextPosition.y];
end
return Cost;
```

Algorithm 1: DTW pseudocode.

In this pseudocode we can distinguish 3 different stages of the algorithm. At first the costs matrix is calculated, this is where the heavy computation is done. Then algorithm calculates the cheapest path from end to begin and after that the final cost is calculated.

This algorithm is so popular it can be found in the main algorithm libraries. The programming language 'R' has this by default, an implementation for C# can be found at `https://github.com/doblak/ndtw`. A PIP DTW python implementation can be found at `https://github.com/pierre-rouanet/dtw`.

3.2 Gamification, Exergaming and Serious games.

Serious game are those games designed with a primary goal different to plain entertainment. There are may ways of approaching game development and this one is gaining relevance and popularity because this opens the world of video games to markets aside from entertainment. We can nowadays play video games whose intention is not to (just) make us have a good time, but also have use as a learning tool, productivity tool, exercise tool or even advertisement.

In a larger spectrum, gamification is one of the resulting techniques of what today are called "Serious Games". Serious Games is a term used to describe those video games whose propose is not just joy. The market share of Serious games wasn't clear until 2006, and it was estimated to around $20 Million worldwide. With the popularization of these techniques and products the share grew up to $2,731 Million in 2016 and is expected to grow roughly 20% annually until it peaks in 2023 with $9,167 million.

With this new trend of making games with miscellaneous proposes a fast conclusion could be the idea of make a game out of everything. This idea, that at first seemed rushed, slowly turned to be an interesting approach to make repetitive tasks less boring. "Gamification" is the term used to describe the technique of adding game mechanics into everyday life activities.

Examples of the gamification trend could be found at Samsung, Starbucks, Nike, Valve and even the US army.

One common mistake is not to know the difference between learning (or educational) game and gamification of learning, and there is nothing to be ashamed of, because layers are very fuzzy. Examples of educational game can be found at school computers, they have their own mechanics and work as a whole. Kahoot, on the other side, represents a very interesting example in gamification. In its basis, Kahoot is just a quiz, but is tweaked in a way students can compete between them and therefore be more motivated than doing the same test by themselves.

Figure 3.7: US Army simulations through serious games.

One important niches of gamification is the capability of simulating complex environments in the process of learning. Games like 'Kerbal Space Program' can make a teenager understand advanced knowledge of physics through a very accurate simulation.

Another example can be found with the 'Microsoft Flight Simulator' saga, which is based in the idea of simulating the task of flight several air vehicles like airships, planes and helicopters. These flight simulators had nowadays become so realistic and complex they are even encouraged to student pilots as part of their learning plan.

Before moving to more serious matters, we would need to have a few words about advergaming. Advergaming is based on the idea that people are more focused while playing games, therefore, the promotion at that time is more effective than the promotion found in, for example, a YouTube video. Advergames are a trend right now and most of the important software enterprises are building up advergame-develop teams.

Exergaming on the other hand can be understand in two ways, the first of them is to add exercise to a video game in order to make it more enjoyable and the other one is the complete opposite, use game mechanics to make exercise more appealing to people that doesn't exercise daily. In the end "exergames" are those games considered a form of exercise.

Figure 3.8: Pepsiman, 1999, PlayStation. Possibly the first advergame. This game took product placement to a whole new level.

3.2.1 Application domains.

When we speak about the reach of exergames it is hard not to speak about the 4th most selling game at this time, Wii Sports. Wii Sports is, for sure, the greatest and most straight example about exergames. The game is played using the Wii-mote, a controller that can track movement and consists in the simulation of 5 different sports: tennis, baseball, bowling, golf and boxing.

Another game from the same company was released a year later with the name of "Wii fit". This one was less popular than the first and it is played through a 'balance' controller, this balance can guess in real time the posture of the user based on the pressure distribution on it.

Both two games were meant to get the exercise closer to people, but in the end they manage to get more people into games than into exercise, despite, they were great software products for their time and the first milestone of an extensive path.

We have spoken about exercise and games, but gamification can also reach productivity. Recently the Android and iOS stores were flooded with some apps claiming to increase productivity, the most common one being 'Forest'. Forest allows you to bury a seed and let it grow, if you unlock your phone and exit the app the plant dies, simple but effective.

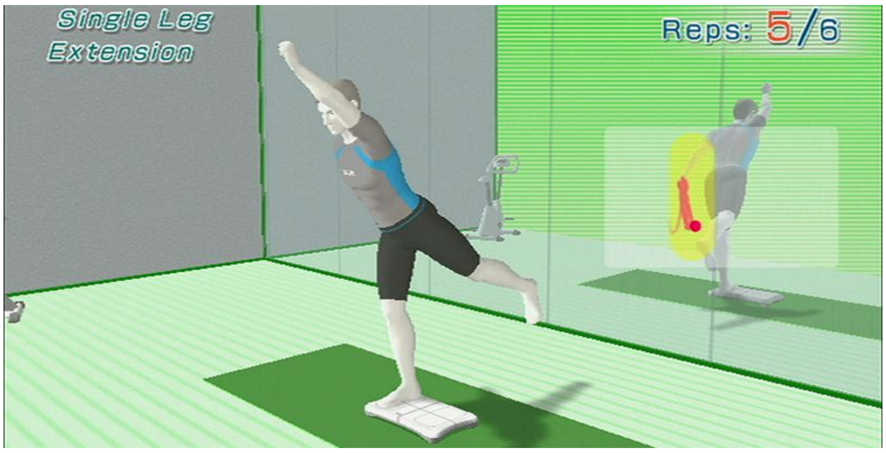


Figure 3.9: Screenshot of Wii fit

We mentioned Kahoot earlier in an example and that's because education can also benefit from gamification. One of the main concern of teachers is (or at least should be) to develop efficient ways of educate their students, and one of the main problems of today's education is a drastic decrease of average attention among population.

Modern problems required modern solutions, and several learning platforms had been released at this point. As personal preference I need to speak about `https://www.codingame.com`, these pages offers several exercises to learn how to code, these exercises cover a wide variety of difficulties and estimated times, all the exercises have graphic representation and a experience reward (because registered users have levels) if you complete it.

One of the most interesting and niche cases is the idea of using gamification for the study and/or autonomous learning of music theory and music practice.

We find a first example of what was previously explained in "HoloMusic XP". HoloMusic XP is a project that uses mixed reality and gamification in order to teach music theory and piano notions. This project original aim was to softened the learning curve every musician faces in its first years[13].

Another example in the use of gamification coming from the university is 'Allegra'. Allegra is a gamificaton tool to enhance and practice melodic dictation. This software translates to an Android application that 'guesses' the note given[8].

A great example of exergame is found in the app 'Zombies, Run!'. Zombies, Run! is an app for joggers (in Spanish "runners"), that offers several challenges in the form of missions. The base idea is to start walking, jogging or running with headphones on, if you are able hear zombies that means they are close, so you must run to escape them. This app offers result tracking, rewards among completing quests, a town you must rebuild seeking resources and a global ranking with a 1 Million player base.

At last, but not least, "Brain Training", released for Nintendo DS was a game whose purpose was close to a "cerebral exergaming platform". Brain Training consist on a series of small games ready to test an exercise your mind in very different ways and track the progress of its users. It includes also a Sudoku collection with some arcade mechanics (error are told to you by default and they are timed).

3.2.2 Techniques used in rehabilitation.

A common problem for rehabilitation patients is that they do not meet the necessary movement dosage required to induce neuroplastic adaptations underlying behavioural improvement. Hemiparesis is a common outcome following stroke and of children diagnosed with cerebral palsy.

Function of the paretic upper extremity (UE) is impaired for up to 6 months after stroke in about 50% of survivors of stroke, and only 5% to 20% of this population regains complete functionality of UE. While up to 50% of functional recovery following stroke is spontaneous, additional recovery can only be acquired through prolonged, intensive therapy .

In the paper "Video Games and Rehabilitation: Using Design Principles to Enhance Engagement in Physical Therapy" [11] written by Keith Lohse, Navid Shirzad, Alida Verster, Nicola Hodges and H. F. Machiel Van der Loos, roughly 1.1 million stroke survivors with UE functionality loss were interviewed. Most of these patients did not received the adequate treatment, most of them due to a lack of financial support from health care systems insurances to let them stay in therapist-supported programs for the time needed.

To complicate even more the situation, most of the patients suffer depression and a great decrease in motivation during long-term rehabilitation programs, in addition, several of these exercises cause rejection within the patients. On the other hand, video games produce an accessible and affordable way of engaging people for long time periods, in the United States, 183 million people spend more than 13 hours of playtime a week.

Despite most of the rehabilitation on the first 6 months being somewhat spontaneous, physicist insists on helping this kind of rehabilitation with collections of upper limb movements and exercises, since those case of exercises offer also a bilateral training of the cortex.

One of the favourite techniques for bilateral training is the use of Virtual Reality (VR) constraint introduced movement therapy exercises. The retention percentage of patients submitted to VR training is significantly higher than the retention of those who coursed it in real life. To maximise effectiveness, therapy needs to have high repetition, supervision, clear rewards, and long duration over time.

The first great problem comes with the need of constant supervision and monitoring, a thing that can be specially problematic if we take into account the exercises must be repeated a high number of times. The first solution we find resides int he Microsoft Kinect, a motion sensor originally released in November, 2010 as an add-on for the Xbox 360.

Figure 3.10: Versions of the Kinect.

The original Kinect sold 35 million units and it was packed with a 640x480 camera @ 30Hz. Was connected via USB 2.0 and was capable of skeletal tracking, gesture recognition, and even some speech recognition. When the Xbox 360 became outdated and the Xbox One was launched a new version of the kinect was also released, called Kinect V2. This one had improved specs:

	Kinect V1	**Kinect V2**
Field of View:	70º horizontal x 60º vertical	57.5º horizontal x 43.5º vertical
Resolvable Depth:	0.8m < Resolvable Depth < 4.0m	0.8m < Resolvable Depth < 4.0m
Colour Stream:	1920x1080x16 bpp @ 30FPS	640x480x24bpp @ 30FPS
Depth Stream:	512 x 424 x 16 bpp, 13-bit depth	320 x 240 16 bpp, 13-bit depth
Infrared Stream:	512 x 424, 11-bit dynamic range	no IRS
Output data path:	USB 3.0	USB 2.0
Avg. processing latency:	60ms	90ms

Table 3.1: Kinect specifications.

Price-wise when the first Kinect was released it had a price of around $150, but you can purchased a bundle with the Xbox 360 for an extra charge of $100, prices stayed put for the new versions. Nowadays you can not buy a kinect, but the successor is the Azure Kinect DK, a standalone computer intended for Internet of things (IOT).

Originally intended as a game add-on, the Kinect son proved to be a better research tool than a game controller, mostly thank to its **ease of development**. Kinect sdk is compatible with any Universal Windows platform (UWP) language, this includes C/C++ supporting CLI (Common language infrastructure, not to be confused with command line interface), C# and even some non standard game developer languages like Visual Basic.Net and JavaScript.

Now we have a possible solution the question remains: **Do games really help with stroke rehabilitation?**. During the previously mentioned paper, video games showed to have many positive behavioural and physiological effects. Rehabilitation-wise video game proved to have positive impacts on cognitive performance, motor performance and affect. The study also yielded results related to performance of surgeons that play video games and surgeons that don't. Gamer surgeons made 37% fewer errors, were 27% faster with laparoscopic drills and were 33% better at suturing tasks.

Studies regarding consequences of video game play in behaviour, physiology and rehabilitation were also made.

Real-time strategy games (RTS) proved to be most effective tool to enhance **working-memory capacity, task-switching abilities, visual short-term memory and verbal reasoning** whereas action games only prove to improve the space reasoning to younger patients. Also, a common

Figure 3.11: Umeji Narisawa, at the age of 99 has spend the last 26 years of her life beating Bomberman once a day in order to her brain active.

feeling through the board is the increase of **self-esteem** after completing any kind of difficult challenge

The self-reward of finishing a game is a reality, and a lot could be done in that way, like tricking the patients to thing they a facing a challenge more difficult that it is since the results are astonishing. Dopamine levels follow game-performance levels, that means an average of 13% increase over resting baseline levels, to put this into perspective, an intravenous injection of amphetamine traduces itself into a 16% increase and a methamphetamine one implies a 23% increase. These results suggest that video game play, when applied to therapy, could lead to durable improvements that **may allow further rehabilitation with daily life activities**.

Most of the early studies were directed with finger-controlled games, but thanks to the increased interactivity of motion controllers we can track and use as a controller complex, coordinated motion within a limb, between two limbs, or even using **full-body motion**. Increasing a player's range, speed, and amount of movement clearly has the potential to help rehabilitate motor impairments.

The use of motion-controlled games like Wii Sports and Wii Fit is accurate due to their low to mid UE motion, in the case of Wii Sports and full body motion, in the case of Wii Fit. Most of the test subject presented metabolic symptoms of moderate to light intensity, meaning, that these games could also be used as an **aerobic-training tool**.

In one study, the use of the Nintendo Wii led to **significantly improved motor function** in participants within 6 months of stroke when they were compared to a recreational therapy group (tabletop games) 4 weeks post intervention.

As a general view, every time Wii games had been used with any kind of rehabilitation purpose (30 minute sessions) patients were more prolific to continue treatment and wanted to have a home copy of the game, specially between participants over **65 years**.

Most of the studies show somewhat better results when using video games as a rehabilitation tool, but also, all of them agree more study is needed in order to give a decisive statement, mostly because some patients postulate games make the rehabilitation program took longer.

Difficulty represents one of the most important factors of the rehab games. A challenge to low may be enough for beginners, but soon won't produce enough satisfaction in the users, therefore games should turn progressively more difficult. Some psychologist have postulated that there is a sensation they called "**positive failure**" produced when a player falls just short of success, and that, makes patients be more motivated to re-try exercises for better results. The joy of positive failure is associated also with the state of “flow,” defined in positive psychology literature as the satisfying feeling of heightened functioning. Positive failures also fit with experimental work on dopamine.

At the very worst and as a need-to-say alternative, the task of gathering of information regarding, mobility, reaction times, attention, memory, verbal reasoning and space reasoning among others can be done by rehabilitation games. This amount of structured information could be of great use in a data analysis system if combined with clustering and classification techniques and therefore develop more **personalised rehab programs** according to patient estimated needs.

An example of rehabilitation game is "Rehability" [7]. The advertisement of this platform have imprinted the words "motivation", "monitoring" and "remotely". Rehability is offered as an alternative to medical exercises for patients that survived stroke, multiple sclerosis or Parkinson’s disease, but also promotes "Active Ageing", a version dedicated to the well being of older people through physical end mental stimulation among **social participation**.

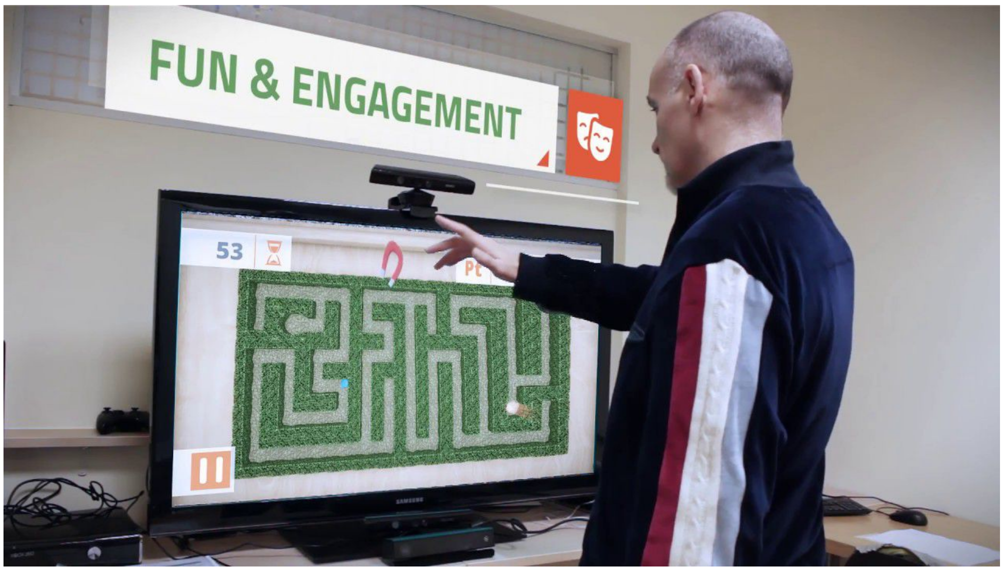


Figure 3.12: Rehability advertisement.

3.3 Development tools for interactive applications.

We talked about video games and their applications to improve everyday life, but now is time to speak about how to develop this software.

[7] http://www.rehability.me

3.3.1 Overview.

All computer systems are interactive in a way or another so, at first the term 'Interactive system' may prove to be a little bit sketchy, despite, when we speak about interactive systems we are speaking about computer systems characterised by a **heavy** or **user-driven** interaction experience (i.e., when we time '-c' as a flag for gcc we are carrying on an interaction, but that doesn't mean gcc is an interactive system itself.).

Interaction paradigms

Traditional computing now looks like it is an obsolete thing, the need for the user to stay in front of a machine trying to squeeze productivity out of it seems quite old-fashioned. It is too visible too demanding and it controls our fate. Most of the users found this paradigm to be unnecessary complex and frustrating, mostly because of (usually) a great number of features available.

Virtual Reality (VR) can be a confusing term since virtual means 'not real'. In essence, Virtual Reality, is a interaction paradigm in which the user is submerged in a simulated 3d world. In order to achieve this, the most common technique is through a head-mounted display. In this paradigm the user interaction occurs within the computer.

Figure 3.13: Oculus Rift, the first commercial VR headset

AR, on the other hand, is an interaction paradigm that is based on the perception of the real world altered and enhanced by the computer. Most people believe AR is just adding some computer graphics to the real world, but in reality, any kind of computer-generated perception is considered AR (this includes visual, auditory, haptic, somatosensory and olfactory clues).

Pervasive/Ubiquitous computing is a modern paradigm born thanks to the explosion of the Internet of things (IOT) technologies. Pervasive computing references a paradigm in which microprocessors embedded in every day life objects offer some kind of extra

Figure 3.14: Pokemon Go. World most famous use of AR in video games.

functionality in an implicit way. This interaction paradigm is specially important to **mHealth** [8] systems.

Android and Java as the de-facto standards.

Nowadays we can't speak about any kind of interactive application without speaking about Java, mostly because of Android. Android is an OS, developed by Android Inc. and later bought by Google based on **Linux kernel** and originally intended for mobile and tablet devices. Nowadays Android has spread a lot and can be found in most of the IOT devices, mostly thanks to its great **portability**.

Despite being developed with phones and tablets in mind, Android is found in a **great variety of different devices**, such as: smart-watches, smart-tv's, desktop consoles and portable consoles [9], laptops, cameras, projectors, Global Positioning System (GPS), and most of the embedded systems in every day life devices, such as the ones at cars, microwaves, e-readers, smart refrigerators . . .

The way of managing this portability across all the variety of CPU architectures, is by means of **virtualization**. Java was born as a programming language that was both, compiled and interpreted, soon, the interpreter grew so much, it was tagged as a virtual machine. That was interesting to Android because if there is a Java virtual machine (JVM) for an architecture all java should be compatible with it. In this way Android achieves portability, by means of java virtual environments.

[8] More at: https://www.researchgate.net/profile/Andreas_Pitsillides/publication/3305729_m-Health_e-Emergency_Systems_Current_Status_and_Future_Directions_Wireless_corner/links/0912f50fefe94e84ac000000/m-Health-e-Emergency-Systems-Current-Status-and-Future-Directions-Wireless-corner.pdf

[9] Such as Nvidia Shield

Despite java being the preferred language for Android development there are alternatives to it. **Kotlin** is a popular alternative to java for Android development since it is fully compatible with JVM environments, but also compiles to web assembly and native code. **Python** community has created a lot of plugins for python development in Android, some of them even compatible with Android studio. At last, but not least, an Android Native development kid (NDK) exists and is the one used in game development since it allows you code applications using **C++**.

3.3.2 Game engines.

Since we are speaking about gamification and, therefore, games, we should dedicate a section to speak about how games are developed.

Historically the common approach was to build the game "from the metal", but as son as 3d graphics were possible games experimented an increase in their development times therefore, re-use code was a must. The result were game engines, "programs used to make games", and they have the role of joining the work of artistic an technical profiles under the same set of tools. Also, they have changed a lot through the years, becoming, in some cases, massive tools.

Overview.

One of the most common mistakes when speaking about game engines is to mistake the concept **game/map editor** with **game engine**. Game engine references the **software infrastructure** that makes the development of a game possible, on the other hand, a map editor or a game editor is the software used to designed levels, scenes, maps and campaigns.

C++ is the de-facto standard for game development, mostly because of its **speed**, but also due to its **portability**, that means, the only language every gaming platform is for sure compatible with, at least nowadays, is C++. Naturally, most of the libraries and engines are going to use C or C++ at their core unless we go back a long way in time when assembly was the only real choice.

There are many ways to approach the development of a game and in the following pages we are going to speak about the most popular game engines and the and the design decisions behind them.

Graphical API, Multimedia Libraries and System calls.

In this section there are collected the main developing tools that either are not completely meant for game development or they are so close-to-the metal and generic they can be used for anything related to multimedia.

The most close-to-the-metal approach of developing a game is using architecture specific system calls. In this way all-time classics (such as **Pac-Man**) were made, is an extremely inconvenient approach coming from an age of unrefined tools and no one should develop like that ever again.

The access to the framebuffer [10] is forbidden or discouraged in modern Operating system so a realistic way to develop close to the metal today should be by the use of graphical API. This approach began to be relevant when in 1997 was released "GLQuake" (a version of Quake, a MS-DOS game compatible with today's hardware thanks to the use of **OpenGL**). There are plenty of alternatives in this area, being the most popular ones **OpenGL**, **DirectX** and **Vulkan**.

Additionally the most famous **multimedia library** is called "SFML" and can be used as a slightly more refined way of displaying graphics and playing sounds.

Is also important to mention the **physics libraries** of "Nvidia PhysX" and "Havok" since most of the high-end game engines based their physics component in either of those.

We are not used to hear a game engine is C++, Vulkan and a custom audio plugin, but back in the days, the games were develop 'from scratch' and whatever was left is what was later called the engine.

There is also important to mentioned, most of the late 90's shooters were developed by means of altering the source code of existing games, this is the case of **'Deus Ex'** and **'Half life'**, who are altered versions of 'Unreal' and 'Quake' but history remembers that Deux Ex runs in Unreal Engine and Half life runs GoldSrc (an enhanced version of Quake engine (also known as id Tech 1)) despite Unreal and Quake being games developed without a former engine.

OGRE.

Object-oriented graphics rendering engine (OGRE) is a free, open source, scene oriented, graphics rendering engine. The Ogre3d class library abstracts all the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other intuitive classes.

[10]The memory chunk that contains the bitmap displayed by the video driver.

(a) Unreal by Epic Megagames (1998).

(b) Quake by id Software (1996).

Figure 3.15: Late 90's bleeding edge of computer graphics.

OGRE recently had a massive update and a face wash, now you can choose between two different versions of the engine. OGRE1, has a wider range of compatible languages (**C#, Java and Python**), and an, also wider, range of target platforms (**Android, iOS, WebGL, Windows UWP**). OGRE2, on the other side is developed with the idea of performance in mind, is ready to display up to orders of 10 000 per frame and it is only compatible with C++.

Cocos2d.

Cocos2d is similar to OGRE in that both are rendering libraries. The difference is that Ogre3d aims to be a cross-platform solution for both, discrete and heavy-loaded graphics game whereas Cocos2d main goal is to provide an easy way of developing discrete graphic mobile games. However, Cocos2d can also build for HTML5, Windows, Mac OS X and Xbox 360.

(a) Torchlight (2009). Example of a game built with Ogre3d.

(b) Clash Of Lords 2 (2013), Cocos2d powered game.

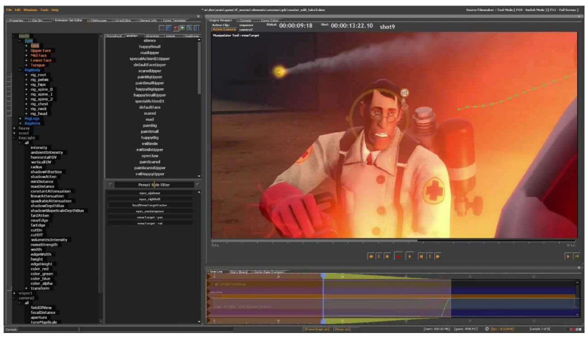
Source Engine.

The history of Source Engine begins with GoldenSrc, the engine behind the game 'Half life'. The first Source game was 'Counter Strike: Source', a port from a Half life mod, released November 1st 2004. This engine became famous because of 2 reasons. The first of those being the game 'Half life 2' and its real time physics powered by **Havok** and also for its wide **mod support**.

Despite the first game being released in November, a **Source SDK** was release in June of the same year, but the Valve editor, **"Hammer"** was not released until the November 5th same year.

Source engine is proprietary software and need to be licensed in order to release commercial games, but there is a free version for students and nonprofit modders, since the SDK allows the development of mods for existing games. Furthermore, the Valve development suite kept on growing with the addition of **Source FilmMaker**, a tool for creating cinematics and **Steam**, a digital distribution platform for video games.

This is the main engine used by **GoG** (Good old Games) [11] engineers to remaster old games and also the one chosen by Treyarch/Respawn entertainment for the **Call of Duty** and **Titanfall** sagas.


(a) Source FilmMaker interface.

(b) Half Life 2 real time physics (2004).

Figure 3.17: Source engine samples.

The interface of the source engine consists on the hammer level editor that uses a source SDK. This source SDK may be customizable via C++. Despite being one of the last-gen favourite engines nowadays looks a bit old-fashioned, that's why users are waiting for a release of a Source 2 SDK since it was announced in 2015 and the engine had its first game released in 2017.

id Tech.

We have spoken about how GLQuake was relevant regarding hardware accelerated games, and the truth is, John Carmack, the mind behind it is one of the most influential personalities

[11]GoG is digital distribution platform for video games and movies that focused on bring back old games with no DRM.

in modern video game development. After the launch of the '**Commander Keen**' trilogy (last date 1991) id Software was finally consolidated, the idea of developing a 3d game was present by that time, but most of the games that used 3d technology felt unresponsive and slow.

Soon, Carmack came with the idea of a **'pseudo 3d raycasting engine'**. This was an innovative render technique based on the idea of emulating 3d environments using just 2 dimensions. The idea was that the renderer should emit one ray per column of pixels in a 2d map and draw walls, columns and ceilings via a 1d depth buffer.

This render technique was revolutionary and old-time classics like DOOM (1993) or Wolfenstein3D (1992) run this engine, but Quake, from 1996, the arguably the first **real time** 3d game of all time did not. The naming of the first id engines is quite muddy, but in this document we will take as id Tech 1 the Quake engine and as id Tech 2 the Quake 2 engine.

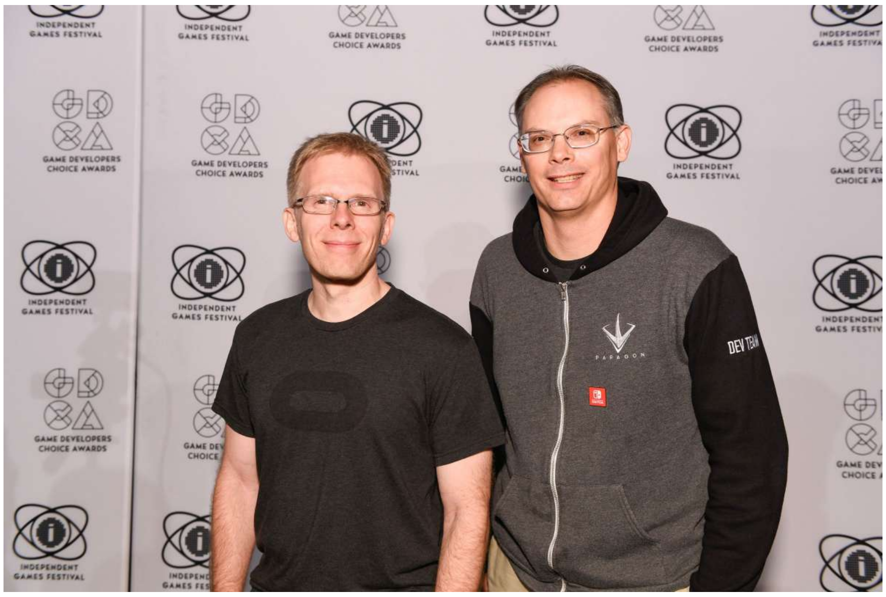


Figure 3.18: John Carmack and Tim Sweeney. Tech leads of Id Tech and Unreal Engine.

From them on, the id Tech engine continued to evolve, however it has always been synonymous with good engineering. The greatest failure of the engine was in 2010 with a game called 'Rage'. Rage was a game that used a new technique called 'Megatexture', this megatextures are supported by nowadays engines, but back in the day gave a lot to talk about.

id Tech engines up to id Tech 4 (included) have GPL license and can be found at the id Tech github [12]. This engines are just what's left of a build-from-the-ground game, and have

[12] `https://github.com/id-Software`

neither level editor nor any kind of additional software tool.

Unreal Engine.

Developed and published by Epic Games, Inc., Unreal Engine 4, UE4 for short, is a suit of integrated tools constituting a framework for game development [13]. UE4 is one of the most extended game engines in the market, despite being mainly focused on photorealism and 'next-gen graphics', UE4 is such a powerful tool it can be used for almost anything thanks to the tandem of *C++* and *Blueprint*.

Figure 3.19: Sample of Unreal Engine 4 photorealistic rendering.

Unreal Engine 4 is built over C++, but Epic considers it to be a difficult language, therefore they provided some special tools, not only to allow live component integration in the engine, but to make the task of coding in C++ a bit easier.

Macros are the very first of these tools. Despite adding difficulty at first, C++ macros are a way to expose the information of a piece of code in a graphical way int the UE4 editor. The three most common macros are **UCLASS**, **UFUNCTION** and **UPROPERTY**. This allows the engine to treat add-on components as native ones.

The C++ of UE4 starts moving away from the ISO standard with the inclusion of **UObjects**. In order to make use of UE4's perks you should start by creating a class which inherits from UObject. This class may need to accomplish certain requirements to be compiled [14].

[13]That's a long way just not to say Game Engine

[14]Requirement such as name the class with a prefix based on the type. More info: `https://answers.unrealengine.com/questions/85202/c-naming-conventions-letters-in-front-of-names.html`

For an experienced C++ programmer the usage of UObjects could be a difficult because they add constraints to the ISO objects. The first of those constraints is that you can only instantiate objects dynamically, this will be done by using the template **NewObject<T>** and **all of them will be pointers**.

Register a class as part of UE4 may be a bit laborious, but has its perks, for example, every instantiated UObject will be registered in **Unreal Engine 4 Garbage Collector**. The existence of a garbage collector combined with C++ pointer semantics make C++ coding in Unreal Engine closer to programming languages like Java or C#.

Alongside C++, Unreal Engine 4, also provides a visual scripting language called **Blueprint**. Blueprint is born from the concept of using a node-based interface to create gameplay elements from within Unreal Editor, it is also based on the idea of Object Oriented (OO) as many scripting languages are.

The Blueprint library can grant designers the use of virtually the full range of concepts and tools only available to programmers. In addition, this Blueprint library can be extended by means of the already named C++ macros.

Same as Source, UE4 needs to be licensed in order to develop commercial applications. The main engine consists on a level editor and it is compatible with several IDE's such as Visual studio, Visual studio code and Xcode for code editing.

Unity3d.

Unity is the community favourite game engine. Originally announced in 2005 as a Apple-only game engine, Unity has become, potentially, the greatest cross-platform game engine in terms of its user base.

Unity is a component-based game engine, is designed with the idea of modularity in mind. The engine was developed in C++ and some source code can be found at their repository page, but the scripting languages the engine offers are C#, "UnityScript" (a version of JavaScript for Unity) and a niche language called Boo.

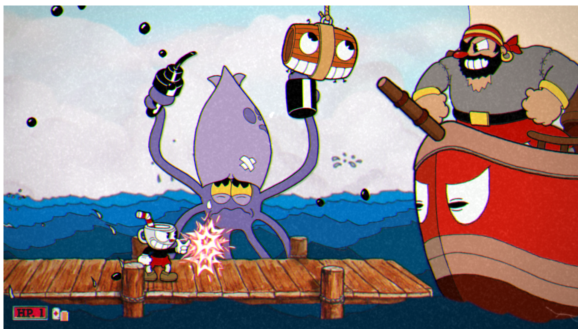
This software is in fact so popular it can be installed through the Visual Studio Installer utility, even though, the Unity Hub still the better option. Once we run Unity we face the level editor, everything spawned in this game editor is a **GameObject**. Gameobjects can be given behavior through code scripts.

Unity is supposed to be supported by Windows and macOS, but there is a branch that is compatible with Linux. One of the best characteristics of Unity is its **impressive target collection**, can build applications for iOS, Android, Windows, Linux and MacOS desktops, WebGL, tvOS, UWP, Vuforia, Facebook, and LuminOS.

To conclude, a Unity license is not necessary unless you want to release commercial games. Originally there are only 3 license types, Personal, Pro and Plus. The idea is that 'Pro' is the one to go for commercial applications, and Personal and Plus are for personal usage, being the last one a kind of "premium" version because it adds access to tutorials and extra content at the Unity store.

(a) Hearthstone (2014), made with unity.

(b) Subnautica (2018), made with unity.


(c) Cuphead (2017), made with unity.

(d) Ori and the blind forest (2015), made with unity.

Figure 3.20: Unity engine samples.

Chapter 4

Methodology

METHODOLOGIES regarding software development will be discussed through this chapter. This is an important topic since in the last few years a lot of new methodologies had became popular.

4.1 Development Methodology.

In the "Objectives" section we spoke about an scalable architecture and with an **iterative and incremental approach**. This idea of **continuous integration** is popular among **agile methodologies**.

During the following paragraphs will define methodologies that have most influenced the methodology followed during the development. This methodologies are part of the agile methodologies, a modern approach to development methodologies, this shal also be defined.

4.1.1 Agile Manifesto.

If we trace back in time we can find development models like "horseshoe" and "cascade"/"waterfall". This methodologies follow a structured, sequential, strict and rigid approach, the development of a software product is divided in several ordered stages and each one of those can't begin until the previous one has been finished.

The more traditional approaches proved to be inefficient due to a lack of communication and flexibility. This way, the main representatives from "**Extreme Programming**", "**SCRUM**", "DSDM", "Adaptive Software Development", "Crystal", "FeatureDriven Development", "Pragmatic Programming", and others sympathetic to the need for an alternative to documentation driven, heavyweight software development processes met on February 11-13, 2001 and the result was the "Agile Manifesto" [3][1].

[1]For free at: `https://agilemanifesto.org`

In this manifesto, the authors write down the **12 rules** they consider developers should use to guide their work. For the most part they advocate techniques that allow the customer to react before the cost of altering the product is too high, especially by means of **short-term constant deliveries**.

4.1.2 Scrum.

Scrum is, by far, the **most popular** agile process framework[2]. A Scrum team commonly consists on a group of 7 to 8 members, one of them being the "Scrum master". This scrum master is the one in charge of carrying on the communication with the **Product Owner** and make sure that the development environment of the **Development Team** is optimal.

This methodology is also based on the idea of "Sprints". A Sprint is a time box of, at most, one month, in which a potential **product increment** is done. This Sprints starts one after the other and during a sprint the goals can not be changed and the quality goals can not decrease.

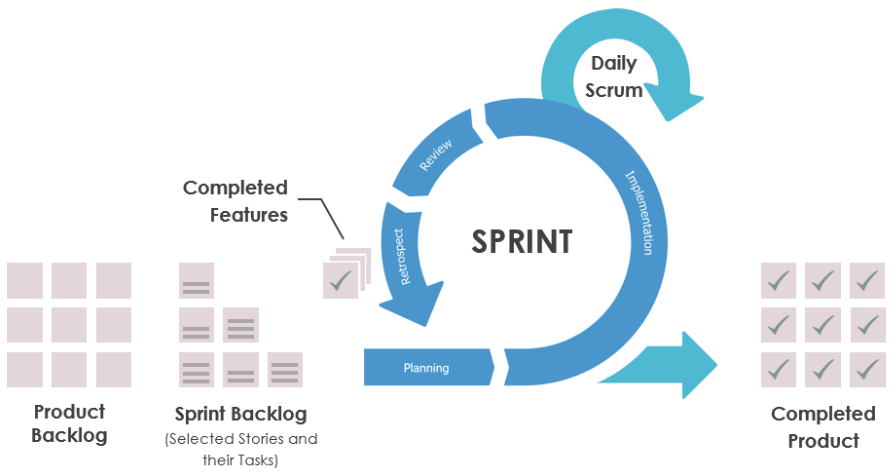


Figure 4.1: Diagram of a scrum sprint.

During a Sprint Scrum participants should collect **Scrum Artifacts** and put them in common once it is finished. The 3 main Scrum artifacts are: the **product backlog**, the **sprint backlog** and **product increment**. It is also encouraged (and specified) that the development team carries out a "**daily scrum**", a 15 minutes daily meeting to plant what they will do for the next 24 hours.

4.1.3 Extreme Programming.

Extreme programming or XP is the other major alternative to Scrum (if we ignore Kanban). In the same way as the rest of the agile methodologies, XP tries to **avoid the large workloads** to be delivered on distant dates.

[2]Guide at: `https://www.scrumguides.org/scrum-guide.html`

In contrast to scrum, XP makes the Product Owner, formerly, the customer to choose the next feature the product should have. Extreme Programming also trust in meetings between inexperienced and veteran programmers in order to learn better techniques.

There are four activities contemplated in an XP environment, coding, testing, listening, and designing. Each of those is an essential link in the chain of development. **Code** represents the core product of software engineering and is achieved by **test** driven development, a good **design** is a must for a good code and **listening** is important for programmers in order to understand the business logic behind what they are doing.

The main difference between XP and Scrum is that Extreme Programming prescribe engineering practices whereas Scrum is a more generic framework. The prescribed engineering practices of XP makes this paradigm more interesting for **educational purposes**.

4.2 Methodologies in game development.

After seeing to interesting approaches to software manufacturing it is time to speak about how they do in game development.

Development teams in game industry are multidisciplinary and have plenty of **roles**. Game development teams are usually divided in **sub-teams** according to their profile, there is usually a tech team, an art team, a design team and a marketing team. Fragmenting these teams allows them to do what they know most, but, since they come from **very different backgrounds** communication problems are going to take place.

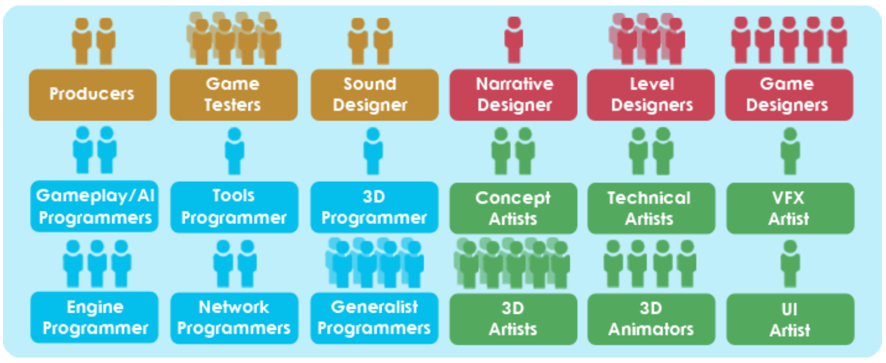


Figure 4.2: Example of a game development team.

Fortunately agile developments work for any kind of business process. The downside of an agile approach is that most of them reject the idea of a "Alpha" or "Beta" release, very common practice in the video game industry, due to the already mention idea of **continuous integration**.

Some companies have changed their development processes in order to settle these new modern techniques. Activision Blizzard is a good example of continuous integration in video games. Most of the games are released as soon as they are consider playable by the quality team and **incremented via continuous updates**. This practice creates rejection within a part of the community accustomed to the old way of development.

4.3 Chosen methodology.

Unfortunately, there is no development team to work with so none of the *default* methodologies are suited for this work. We have created a methodology based on the previously explained ones.

The plan was to meet the directors every other week. During this meetings the directors will act as **quality consultants** offering better design choices. The goal was to accomplish every requirement before the next biweekly meeting. Exceptions may occur if a milestone turns out to be **longer than expected** or if there is any kind of technical issue.

4.4 Project resources.

4.4.1 Hardware resources.

Computers.

The devices uses during the development of the project were a Medion Erazer p6689 and a custom desktop computer, the specifications are as follows:

	Laptop	**Computer**
CPU:	Intel® Core™ i5-8250U	Intel® Core™ i5-9600k @ 5Ghz
GPU:	Nvidia® GTX™ 1050 4gb	Nvidia® GTX™ 1080
RAM:	8 Gb DDR4 @ 2400 Mhz	16 Gb DDR4 @ 2400 Mhz
Operating system:	Ubuntu 16.04	Windows 10
Display:	1920x1080p IPS screen	2 x 2560x1440p screens

Table 4.1: Specifications of the hardware used in the project.

Other devices.

The only external device used apart from the computers was the TVico .TVico's hardware specifications are as follows, keep in mind everything related to image makes reference to the integrated camera:

- **Size** - 172x63x56mm
- **Weight** - 0.8 kg
- **Depth Image Size** - 640*480 (VGA) @ 30FPS RGB

- **Image Size** - 1280*720 @ 30FPS (UVC Support)
- **Field of view** - 60^o horiz x 49.5 vert. (73^o diagonal)
- **Microphones** - 2
- **Built-in OS** - Android
- **CPU** - Quad-core Cortex A17 Up to 1.8GHz (ARM)
- **GPU** - Mali-T7 600MHz
- **RAM** - 2GB DDR3
- **Video Output** - HDMI 2.0 (Audio+Video)
- **Other Connectivity** - USB 2.0 + Micro USB + 802.11 2.4GHz Wi-Fi + Bluetooth

4.4.2 Software resources.

This is the list, with a small description of every piece of software used to develop and deploy UnRehab.

Operating systems:

- **Ubuntu 16.04:** The initial approach to the project was done with OpenPose. In order to build OpenPose applications Ubuntu was installed.
- **Windows 10:** Windows was used in order to run Unity3D. On a general basis Graphics Processing Unit (GPU) drivers are more optimised for games in Windows 10, therefore, the use of Windows 10 is highly encouraged for game development.
- **Android 5.1:** Was the target platform for the application.

Programming languages:

- **C++:** This language was used during the development of OpenPose demos, with Unreal Engine 4 and during the development of the DTW library.
- **C#:** Unity3d uses 3 different languages for scripting, C#, UnityScript and Boo. The one selected during the project was C#.

Development Tools.

- **Visual Studio Code:** A free, strangely fast solution for an editor.
- **Visual Studio Enterprise:** Visual studio is fully integrated to both, Unreal Engine 4 and Unity3d and is the to-go solution for an game engine IDE.
- **CMake:** A C/C++ meta build tool. CMake doesn't build the code, but generates build projects for the construction tools of your operating system.
- **GCC and Clang:** Both were used during the first stages. Usually Clang is used for debug since errors are clearer and GCC is used for the final release.

- **Subversion:** Was used as the software versioning and revision control system for UnRehab.
- **Git, Git desktop and Github:** Like subversion but for the final release.
- **Unity3d:** A game engine we have spoken about. UnRehab is built in Unity.
- **Blender:** Is a free and open-source 3D computer graphics software toolset and was used for creating some models.
- **Unreal Engine 4:** The first approach to the development was done with this engine.
- LaTeXand Overleaf: LaTeXis the markup language used to write down this document and Overleaf the IDE and webpage used for it.
- **`https://sequencediagram.org`:** This webpage was used to generate the sequence diagrams of chapter 5.

Chapter 5

Application Architecture.

IN this chapter we will speak about the architecture design choices and hardware deployment of the application. Its being hard because Unity3d is not as object oriented as it could be. Therefore the architecture is based on the idea of **agents**, agent interaction and event-based programming.

5.1 Hardware deployment.

One of the objectives of the application was to get a small hardware deployment. One of the reasons to manage a petty infrastructure is to achieve **portability**, but also to try to keep it as **affordable** as possible.

In order to achieve this goal we will be using a kinect-like android box named TVico. This device is ought to be used as a side-television computer and have some features that are very attractive for this project such as:

- Depth sensor
- Nuitrack SDK compatibility
- Full body tracking
- Basic Gesture recognition

TVico uses a library called Nuitrack which we've already talked about. There are two ways of using TVico and Nuitrack.

First of those is as an **standalone** computer, we will use this configuration during the development of **UnRehab**. In this mode the TVico will behave as a standalone device, the only extra hardware needed is any kind of HDMI compatible display.

The second way of using the TVico is like a **wireless camera**. In this manner an application execute external in an external device could connect to the TVico and use the information gathered by it. In order to achieve this 3Divi offers an app called "TVico.apk" that can be

Figure 5.1: TVico

run as a **daemon**. On the other device at the time of initialising Nuitrack the developer can specify a Nuitrack **host** and a password and all the data regarding the skeletons will be transferred via IP protocol.

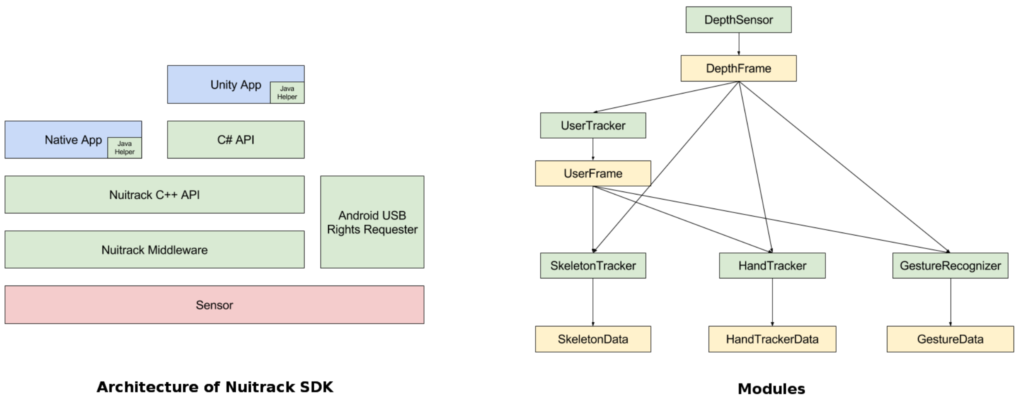


Figure 5.2: Nuitack architecture

This last way of executing Nuitrack can be very interesting, sadly it won't be used through the development of the project.

5.2 Modular design and Scalability.

A desirable feature of every software system is the ability of fast integration. As set before, **Unity3d** is going to be used in order to managed a ".apk" file that could be deployed and executed directly in the hardware.

Unity is an scene-oriented game engine. Unity **scenes**, denoted by the extension ".unity", are the engine solution to the concept of "level". Unity also allows a random level load, we have solved most of the scalability problems since **scenes can be developed independently**.

The idea of the levels, or in this case scenes, is to **layout correctly** the game or spawnable objects. In this case each of the mini games will have its own scene an there will be a nexus level to link them all. After completing the challenge a mini game purposes the player **will be directed to the main menu** to choose another scene.

In the first release there will be a total of 5 different scenes, but a sixth one will be in the project. We will speak more about this scenes in the following paragraphs.

5.3 Design Choices.

5.3.1 The avatar.

The first design choice of the project was to use an in-game avatar in order to represent the patient in the game. Most of the unity models are done joint by joint, this joints can be mapped using C# scripts, for this case we will use **indirect mapping**. Indirect mapping means that avatar will stand still in a fixed point and joints will rotate according to the user.

```csharp
using UnityEngine;

[System.Serializable]
class ModelJoint
{
    public Transform bone = null;
    public nuitrack.JointType jointType = nuitrack.JointType.None;
    public nuitrack.JointType parentJointType = nuitrack.JointType.None;

    [HideInInspector] public Quaternion baseRotOffset;
    [HideInInspector] public Transform parentBone;
    [HideInInspector] public float baseDistanceToParent;
}
```

Listing 5.1: Joint mapping information class in C#

Once we have a class to store the information regarding joints and user rotations we can create a class to manage the translation of the avatar rotation.

```csharp
using UnityEngine;

public class JointMapping : MonoBehaviour
{
    [Header("Rigged model")]
    [SerializeField]
    ModelJoint[] modelJoints;

    [SerializeField] private bool freeMovement = false;
    [SerializeField] private bool bodyMovement = false;

    protected void Start()
    {
        foreach (var modelJoint in modelJoints)
            modelJoint.baseRotOffset = modelJoint.bone.rotation;
    }

    protected void Update()
    {
        if (CurrentUserTracker.CurrentSkeleton != null)
        {
            MoveJoints(CurrentUserTracker.CurrentSkeleton);
            if (freeMovement) MoveBody(CurrentUserTracker.CurrentSkeleton);
            if (bodyMovement) MoveTorso(CurrentUserTracker.CurrentSkeleton);
        }
    }

    private void MoveTorso(nuitrack.Skeleton skeleton)
    {
        Vector3 torsoPos = (0.001f * skeleton.GetJoint(nuitrack.JointType.Torso).ToVector3());
        transform.position = torsoPos;
    }

    private void MoveBody(nuitrack.Skeleton skeleton)
    {
        foreach (ModelJoint modelJoint in modelJoints)
            modelJoint.bone.position = (0.001f * skeleton.GetJoint(modelJoint.jointType).ToVector3());
    }

    private void MoveJoints(nuitrack.Skeleton skeleton)
    {
        foreach (var modelJoint in modelJoints)
            modelJoint.bone.rotation =
                Quaternion.Inverse(CalibrationInfo.SensorOrientation)
                * (skeleton.GetJoint(modelJoint.jointType).ToQuaternionMirrored())
                * modelJoint.baseRotOffset;
    }
}
```

Listing 5.2: Avatar mapping script.

The set of joints the user can rotate in the avatar was originally of 11, but later reduced to 9. Of the 19 joints that Nuitrack tracks we stayed with: the torso, the left and right section of the hips, both shoulders, both collars, and both elbows, the 2 discarded joints were the knees.

On the other hand, the layout of the levels resembles modern interior design. The reason of doing this is to make the patient feel like home. The furniture used is part of an free asset pack of realistic furniture and props. **Horror vacui**, "fear to the emptiness", is a common phobia nowadays and vacuous spaces are commonly used as a means of disturbing the user.

5.3.2 Main menu and examples of bad code.

As mentioned before, the idea is too use a scene in order to load the rest of scenes. This main scenes locks an avatar between four capsules, each one of a different colour.

In the final version, each capsule redirects to a scene if the user touch its for too long. An interesting, yet, discarded approach would be to use the capsules as **arrow keys** in order to navigate a menu.

The **blue capsule** at the top left spot will load a mini game about cleaning a window. This game allows the user to control two hands and should **remove all the dirt** of a framed glass by means of sliding one of the hands over it.

The **red capsule**, located at the top right corner loads a game about collecting a ball and dropping it into a trash bin. The ball can spawn at several points selected by the user and the game will end once the user scores enough points.

The **green capsule**, the one placed at the bottom left part repeats the **tutorial**, a level that explains the basics of the program.

The **yellow capsule**, situated at the bottom right area, loads a **gesture recognition** mini game. In this mini game the user will have to perform gestures in order to get points, once the user reaches out the target score the game will finish.

The time required to start loading and the level to load could be **parameterized in the editor** via the following class.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class OnTouchSceneLoad : MonoBehaviour
{
```

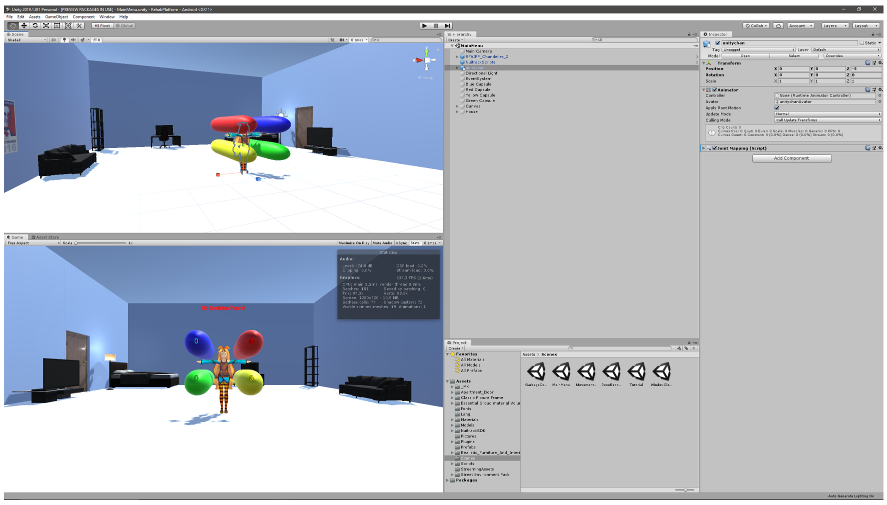


Figure 5.3: Main menu capture from the Unity editor.

```
    [SerializeField]
    public string levelToLoad;

    [SerializeField]
    public Text textBox;

    [SerializeField]
    public Text timeText;

    [SerializeField]
    public float timeToStartLevel = 5f;

    private float startTime;

    private float elapsedTime;

    private bool isTimerOn = false;

    private void Update()
    {
        if (isTimerOn)
        {
            elapsedTime = (Time.time - startTime);
            timeText.text = elapsedTime.ToString("0.00");

            if (elapsedTime >= timeToStartLevel)
            {
                textBox.text = "Loading level: " + levelToLoad;
                SceneManager.LoadScene(levelToLoad);
            }
        }
```

```
38     }

40     private void OnTriggerEnter(Collider other)
41     {
42         var outline = textBox.GetComponent<UnityEngine.UI.Outline>();

44         if (outline != null)
45             outline.effectColor = new Color(50f, 255f, 50f);

47         textBox.color = new Color(0f, 255f, 0f);
48         textBox.text = name + " has been touched";

50         if (CurrentUserTracker.CurrentSkeleton != null)
51         {
52             // Start Timer
53             startTime = Time.time;
54             isTimerOn = true;
55         }

57     }

59     private void OnTriggerExit(Collider other)
60     {
61         // End timer
62         textBox.text = "";
63         timeText.text = "0";
64         isTimerOn = false;
65     }
66 }
```

Listing 5.3: Example on a bad use of C#.

As we can see, there are great falls in this code, flaws any novice Unity programmer will fall on. We can see a class that is dedicated to logic but in lines **2**, **10** and **14** uses UI elements explicitly. This design goes against the idea of **decoupling** since the class "OnTouchSceneLoad", is loading the next level after certain conditions and reporting date to some UI elements, that means doing **two things instead of one**. User interface classes should depend on logic classes and not the other way.

5.3.3 Score manager and advanced design patterns.

We have seen an example of bad code for a reason, and that is that the use of flexible languages, specially in closed, environments **may**[1] encourage bad code. That's why, some

[1] I don't mean that 'real developers' have to program in binary.

game engines have their own ways to implement **design patterns** and to dismiss the use use of **antipatterns**.

A good example on antipattern prevention is seen in UE4 as the way it instantiate objects in real time makes it impossible to code a **singleton**. On the other hand Unity **prefabs** are an awesome example of a **prototype** patterns made easy.

The pattern "**Observer**" is arguably the most popular design pattern in modern programming. The main reason of this popularity is that the **model-view-controller** architectural pattern is build over observer, but also observer is the best choice if we want an **event based system**.

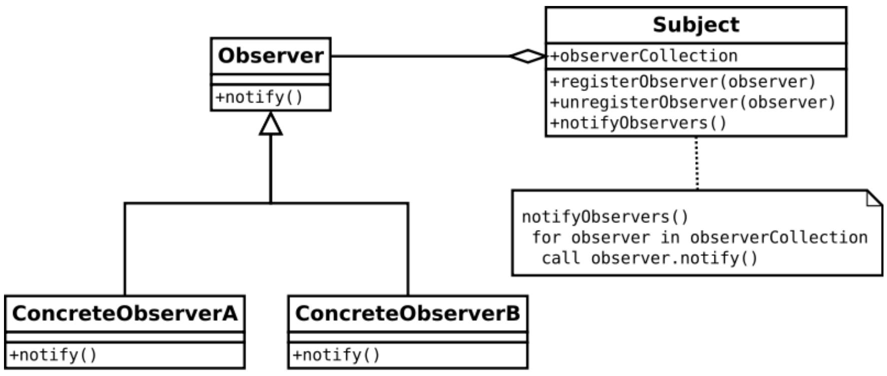


Figure 5.4: UML diagram of an observer pattern.

This pattern consist on the "subscription" of an "observer" interface to a "subject" object, allowing a relationship of 1 subject to many (from 0 to infinite) observers. This subject will notify all its subscribers when its state changes. The main downside of the pattern is that the grow of **time complexity** and **data flow** complexity as the list of observers an subjects grow, reason some communities want it to be tagged as **deprecated** [12].

Some people consider delegates a design pattern themselves despite not being listed in the original 25 patterns discriminated by the "**The gang of four**" [9]. In practice **delegates** and **multicast delegates** are consider a modern approach to the observer pattern, we can see an example on the following code:

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class ScoreManager : MonoBehaviour
{
    protected int score = 0;
8        [SerializeField]
9        protected int targetScore;

11       [SerializeField]
12       protected int delayTime = 2;

14       public delegate void GameFinished();

16       public static event GameFinished OnGameEnd;

18       protected virtual void Start()
19       {
20           OnGameEnd += EndGameOperations;
21       }

23       private void EndGameOperations()
24       {
25           Invoke("LoadMainMenu", delayTime);
26       }

28       private void LoadMainMenu()
29       {
30           SceneManager.LoadScene("Scenes/MainMenu");
31       }

33       protected void UpdateScore(int newScore)
34       {
35           score = newScore;
36           if (score >= targetScore) OnGameEnd();
37       }

39       public int GetScore()
40       {
41           return score;
42       }

44       public int GetMaxScore()
45       {
46           return targetScore;
47       }
48   }
```

Listing 5.4: Example of a class that uses delegates.

In lines **14** and **16** we can see the declaration of a delegate and an event, the first line means that functions compatible to the delegate "GameFinished" should have no arguments and must return void, the second line instantiates an encapsulated delegate (event) of type GameFinished. To put it short, both lines means that the class "ScoreManager" is can **notify the event "OnGameEnd"** and everyone can subscribe to it.

In order to subscribe to an event the user can add a function to the "execution queue" using the operator "+=" as we can see in line 20 of the following code:

```
 1 using UnityEngine;
 2 using UnityEngine.SceneManagement;

 4 public class ScoreManager : MonoBehaviour
 5 {
 6     protected int score = 0;

 8     [SerializeField]
 9     protected int targetScore;

11     [SerializeField]
12     protected int delayTime = 2;

14     public delegate void GameFinished();

16     public static event GameFinished OnGameEnd;

18     protected virtual void Start()
19     {
20         OnGameEnd += EndGameOperations;
21     }

23     private void EndGameOperations()
24     {
25         Invoke("LoadMainMenu", delayTime);
26     }

28     private void LoadMainMenu()
29     {
30         SceneManager.LoadScene("Scenes/MainMenu");
31     }

33     protected void UpdateScore(int newScore)
34     {
35         score = newScore;
36         if (score >= targetScore) OnGameEnd();
37     }

39     public int GetScore()
```

```
    {
        return score;
    }

    public int GetMaxScore()
    {
        return targetScore;
    }
}
```

Listing 5.5: Example of a class that uses delegates.

By means of declaring events as static and subscribing to them in the **Start** method we managed to build a structure as **decoupled** as possible, since none of the managers know each others, or have references to each other but they interact with each other.

There may be a problem if more than one ScoreManager is spawned into a scene, since every subscriber **should be notified once per ScoreManager**. Formerly managers should only be instantiated once so this is not a problem.

The **ScoreManager** class is a "shared class" (used across mini games). The goal of the class is to **redirect the player to the main menu** once the game is over, arguably this could be split up into two classes, one that keeps track of the user score and another to load the level when the score counter reaches the target score, but, since the task are so **trivial** and **coupled** we can break the rule of one use per class, specially since we are using only one extra namespace within unity engine.

5.3.4 User interfaces.

During the development of the project we managed to maintain the user interfaces at a minimum. The reason behind this is that too much information can make the game look more difficult than it is and therefore cause stress and rejection.

The main UI element of the whole project is the score-meter. This piece of UI shows the user the current score and the target score of the exercise. The behaviour script uses a reference to a score manager in order to extract the information, this is very convenient, since every mini game must use a class derived from score manager to load the next level or main menu. This approach uses a per-frame update of the score, since different mini games use different events and, therefore, we would need to create different behaviours from different events.

The second repeated element are the progress bars. Unity, by default has no progress bar GUI, we managed to create one by using a **slider**, another valid approach should be the use of nested and **masked images** and scale one of them horizontally.

The use of progress bars is very adequate because they give information to the player without being **too explicit**. The use of explicit information is **discouraged** since the user must not know what is happening inside the system.

For this reason, any number greater than 30 that will be this played uses a progress bar instead of explicit numbers. We can see examples of this in the **window cleaning** game, whose target score is 3 times the dirt on the screen (about 740 in total), and in the **gesture capture** levels, since the system collects information during 90 frames, during this one the progress bar first "decreases" to tell the user 'get ready' and increases once the capture starts.

The last UI element are the **chat bubbles**. The use case of the chat bubbles is mostly for decorative purposes. This bubbles can be seen at the **tutorial** and the **garbage collecting** scenes, in the first one the serve as a background for the instructions and in the second one they display some **motivational sentences**.

5.3.5 Garbage collecting game.

The garbage collecting game is based on the sport **basketball**. Ball will spawn at a 'random location' and the user must pick it and throw it to the trash can, once the ball reaches the bin a new **ball will appear** after a while. Once the user reaches a customizable-through-the-editor score the game ends.

By default the target score is set at 20 and each 'goal' (ball guided to trash) is one point. The frame the ball contacts the trash the ball will be despawned and then respawned. The new spawn of the ball is done by the SpawnManager, this occurs by default half a second after the goal occurs.

The spawn positions of the ball are defined by **game objects**, the **transform** of this game objects will be the transform of the spawned ball, the ball cannot be spawned twice at the same transform. Another possible solution could have been using

In order to attach the ball to the avatar hands we have code a script that adds the ball as a **sub-object** of the hand. First, the hand registers the hit with a collider, if the object associated to the collider has the tag "Ball" the ball will stick to the hand. An exception to

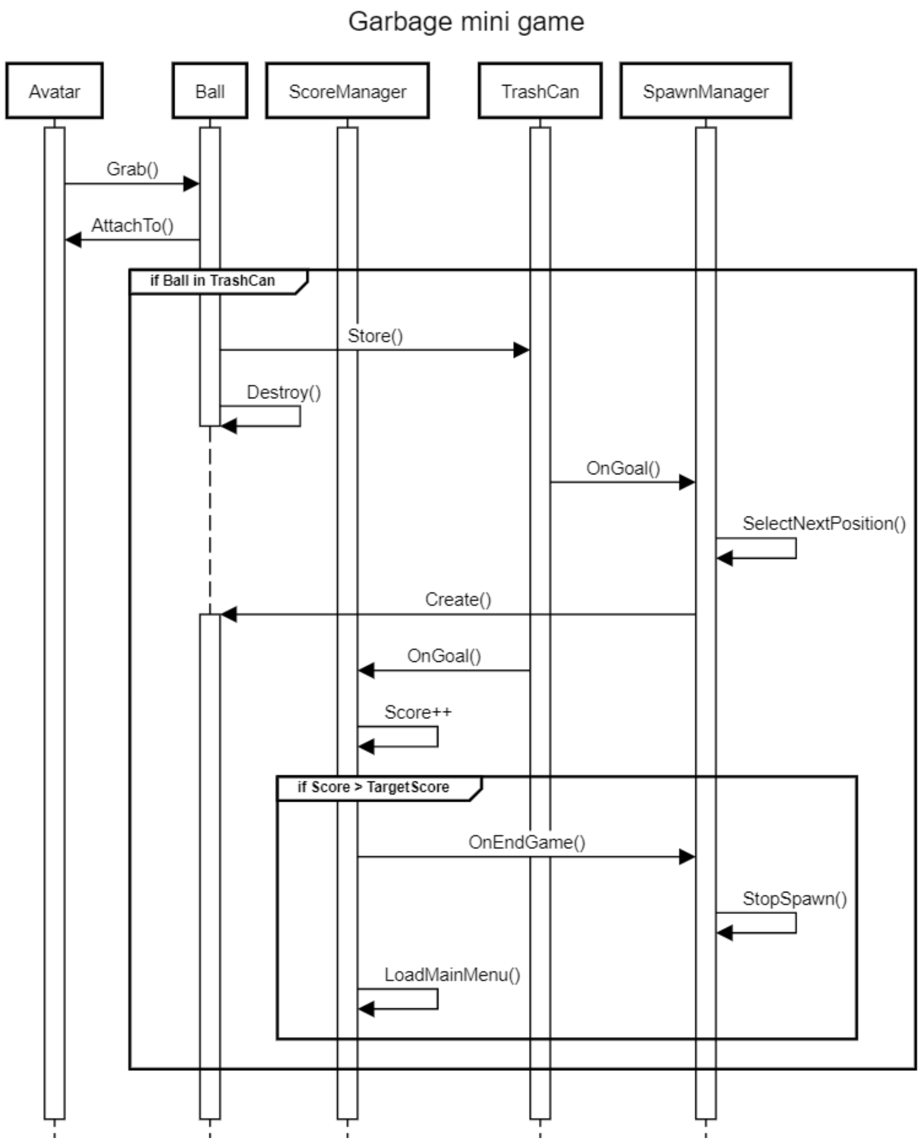


Figure 5.5: Garbage mini game expressed as a sequence diagram.

this case is if the ball is **already grabbed**, in this case, the ball should have a "parent", the hand", and therefore won't be picked up.

Two similar algorithms in this mini game are the random motivational sentence chooser an the random spawn point chooser. In order to manage not to spawn the ball in the same spot twice or to display two times the same phrase we have stored them in an array and stored the index of the last happening.

5.3.6 Window cleaning game.

The first noticeable change in this game is that the virtual avatar is not the usual character. In this game we will control **two hands**. These hands are controlled by the user's real hands.

In order to achieve hand movement we developed a new script instead of using the existing one about joint mapping. In this script we dispense with the use of the storage class "ModelJoint". The operation of this script is as follows, every frame the script queries the position of the hands and calculates the new position. This new position consists in the **starting point** of the hand composed with the relative position of the hand, because Nuitrack estimates user is located at (0, 0, 0).

```
using UnityEngine;

public class HandMapping : MonoBehaviour
{
    public enum Hands { left = 0, right = 1 };

    [SerializeField]
    Hands currentHand;

    [SerializeField]
    private Vector3 originalPosition;

    private void Update()
    {
        if (CurrentUserTracker.CurrentSkeleton != null)
        {
            Vector3 handPosition = (currentHand == Hands.left)
                ? CurrentUserTracker.CurrentSkeleton.GetJoint(nuitrack.JointType.LeftHand).ToVector3()
                : CurrentUserTracker.CurrentSkeleton.GetJoint(nuitrack.JointType.RightHand).ToVector3();

            handPosition *= 0.0025f;
            handPosition += originalPosition;
            handPosition.z = originalPosition.z;
            gameObject.transform.position = handPosition;
        }
    }

}
```

Listing 5.6: Example of a class that uses delegates.

In this case, the game manager tracks points based on the times each piece of dirt has been cleaned up. When one big dirt pile is cleaned it transforms into a medium dirt pile, whom transforms into a small one when cleaned too. Small dirt accumulations disappear once cleaned.

The event of dirt being cleaned is triggered once the hand of the user **leaves** the dirt. This is done this way to encourage constant rubbing, more like the user will do in real life. Also, a lockdown of 1 second was added, the idea behind this is **slow down** the user and prevent errors due to the **collision events** being query too often.

The original design of this game was very different. In first candidate the avatar was **facing the camera** with the window in front and the player had to touch the dirt in order to clean it. This approach ended up being too unnatural and counter intuitive.

Interface in this game offers just a progress bar and a the percentage of the carried our cleans in relation to the target cleans. The percentage number format is discretised so that the user doesn't try to calculate any numbers. The idea of displaying the raw numbers falls out inefficient. The objective of the exercise is to emulate an action, not to achieve a goal.

We believe hiding the event that increments score to the user makes sense in order to keep the **simulation** illusion while offering some game mechanic through the **progress bar**. The main downside of the previously mentioned progress bar is may mislead some people to think that is a timer.

5.3.7 Pose recognition game.

The game about pose recognition is the most interesting one. The mini game, at first, look similar to the **garbage collecting** game. The goal of this game is to perform the **correct gesture** a number of times. Like in the garbage collecting game, the target count could be customised through the **editor**).

The task of pose recognition has been done with two techniques. The first of those is with the built up pose recognition module of Nuitrack. The other method is by means of using **time series analysis** with the position or the rotation of user joints.

Nuitrack gesture recogniser.

This module offers us an acceptable amount information about users and gestures. There are two enumerations within the module, one with the possible gesture types and other that encodes the possible user **state types**. According to the user tracker manager a user can be in 3 different states: absent, in scene, and active.

The second enumeration, the gesture types correspond to the gestures Nuitrack can track. The gesture types are similar to a common input found in **android** operating systems.This gestures are:

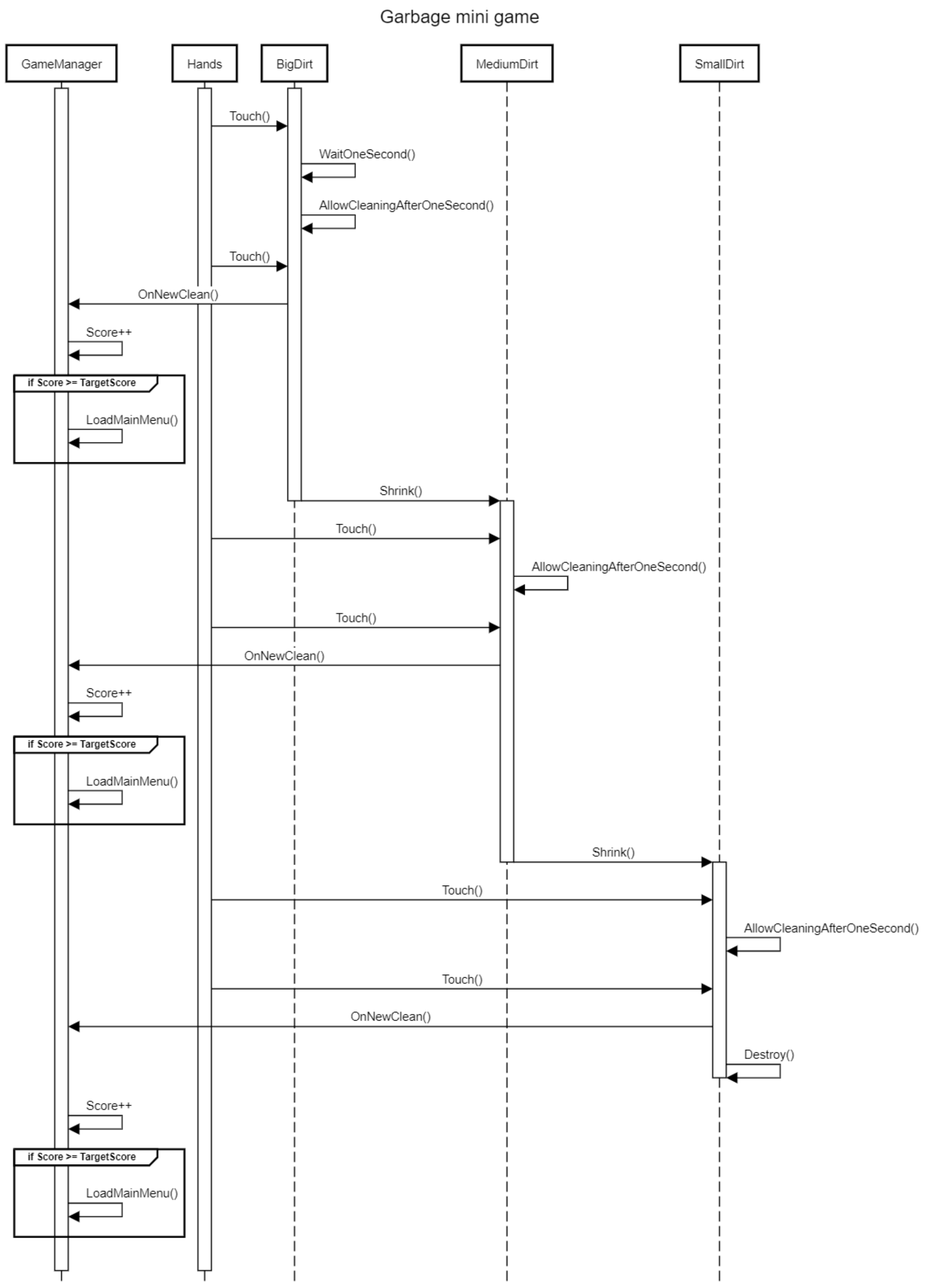


Figure 5.6: Window mini game expressed as a sequence diagram.

- Waving gesture.
- Pushing forward.

- Swiping left.
- Swiping right.
- Swiping up.
- Swiping down.

The gesture recogniser is one of the six Nuitrack modules. The main class of the gesture recogniser module is, the gesture recogniser. The usage of this class should be done through a **static class** in the .Net Nuitrack manager.

The intended way for developers to use this module is via **delegates**, the event of the delegate is called **OnNewGestureEvent**. The gesture function delegate should accept an object called "GestureData". This class stores information regarding the timestamps when the event was launched, the users and the gesture they have carried out.

For Nuitrack, a gesture is composed of a user and a gesture type. At a lower level the gestures have a **progress** indicator and a confidence **proportion**. Neither of both were used during the development of this work.

The pose manager uses three of the Nuitrack gestures, **waving, swipe left, swipe right**. Despite using the Nuitrack gesture recogniser, the system also implements **custom gesture** recognition.

Time series analysis with DTW.

The Dynamic Time Warping algorithm is used specially in **speech recognition** algorithms, but can used used to analyse any time series. An example of the DTW used for pose recognition can be found at [5]. When we say DTW we can refer to both, the algorithm itself and to the **resulting distance**. A classic DTW algorithm has 3 parts: **calculation of the cost matrix**, **pathfinding across the cost matrix** and **calculation of the total cost**.

In order to use DTW for gesture recognition we have to create a model that represents the **perfect gesture** first. After that we can query the information stored in the user's skeleton and create another time series to compare with which to compare it.

This model is based on user **joint rotations**. This model stores information regarding the rotation of user joints over a sequence of **frames**. The target framerate of the game is of 30 frames per second, and a gesture should take an estimate of 3 seconds. Rotations in Unity are expressed in **quaternions**. Quaternions are a number system that extends the complex

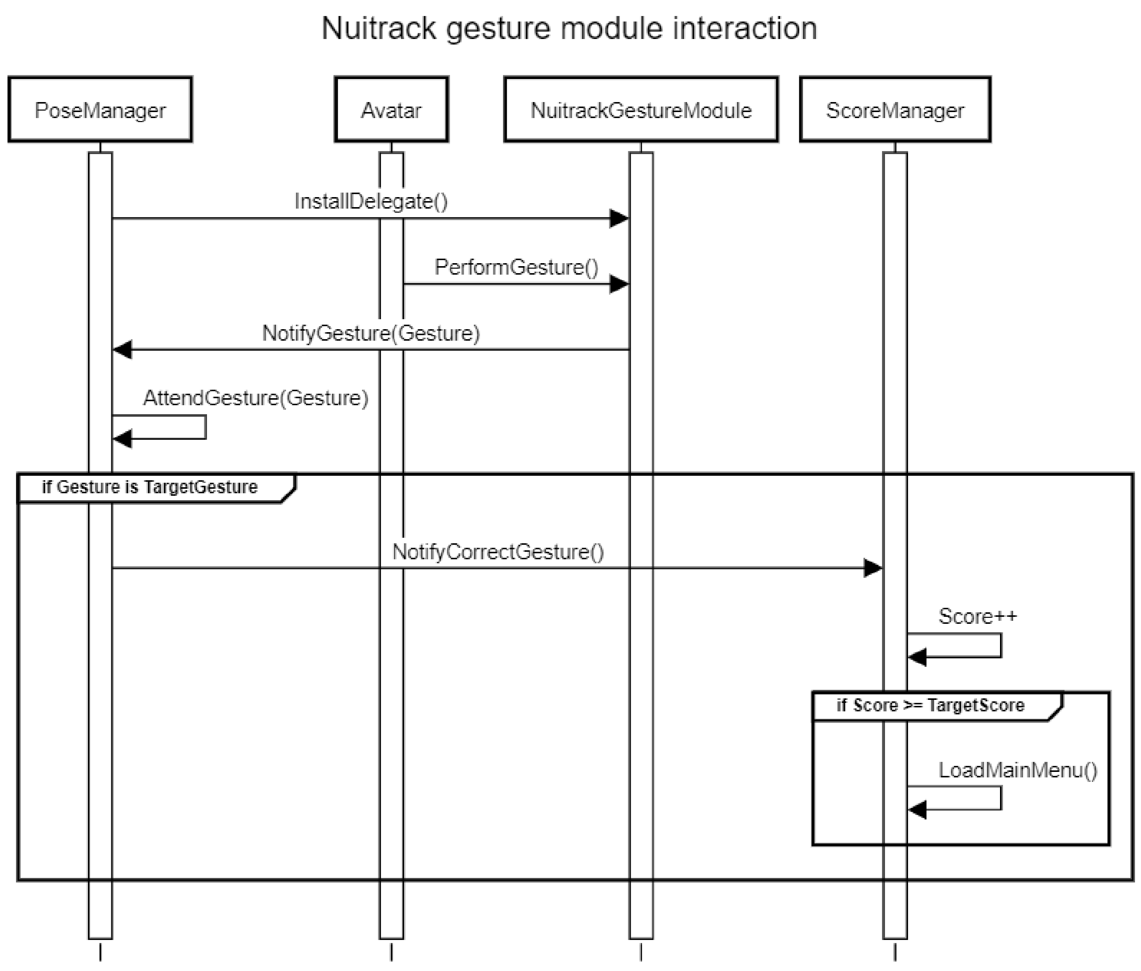


Figure 5.7: Sequence diagram representing the interaction of the game with the Nuitrack gesture module.

numbers, commonly expressed as a quadruple:

$$a + b\,\mathbf{i} + c\,\mathbf{j} + d\,\mathbf{k}$$

```
public class PoseModel
{
    static public int numberOfJoints = 9;

    static public int expectedFrameRate = 30;

    static public float numberOfSeconds = 3f;

    static public int numberOfFrames = (int)numberOfSeconds * expectedFrameRate;

    private Quaternion[,] rotationModel = new Quaternion[numberOfJoints, numberOfFrames];

    private double maxError = 200.0f;

    ...
}
```

Listing 5.7: Information of a gesture model.

The rotations are quaternions, but the default implementation of the DTW only works with **numeric** time series. In order to solve this problem we though in two different approaches.

Keep in mind that Dynamic Time Warping is a **greedy** algorithm. That meaning, it needs to calculate **all the possible costs** and after that is when the path across this costs is calculated. The path can only move on to neighbours. This path may **not be the optimal** in regards to costs, but it is not very computationally demanding.

The cost matrix is calculated by the absolute value of a substraction, and here we find the first problem. **How are quaternions substracted?**. Theoretically this substraction value can be changed by any kind of **distance** function. The absolute value of a substraction could be interpreted as a **euclidean distance** between two points.

$$d(p,q) = \sqrt{\sum_{i=0}^{n}(p_n + q_n)^2} = \sqrt{(p_0 - q_0)^2} \quad if \quad \forall x \in n \geq 1 \quad p_n = q_n = 0$$

In the end what it really takes to calculate the costs matrix is a **distance function**. The euclidean distance operation can be substituted by any other distance function, even if the inputs are not real numbers. We have spoken about the **euclidean distance** as the only possible distance, but a few examples could be the **Manhattan distance**, the **greater value** or the **euclidean squared**.

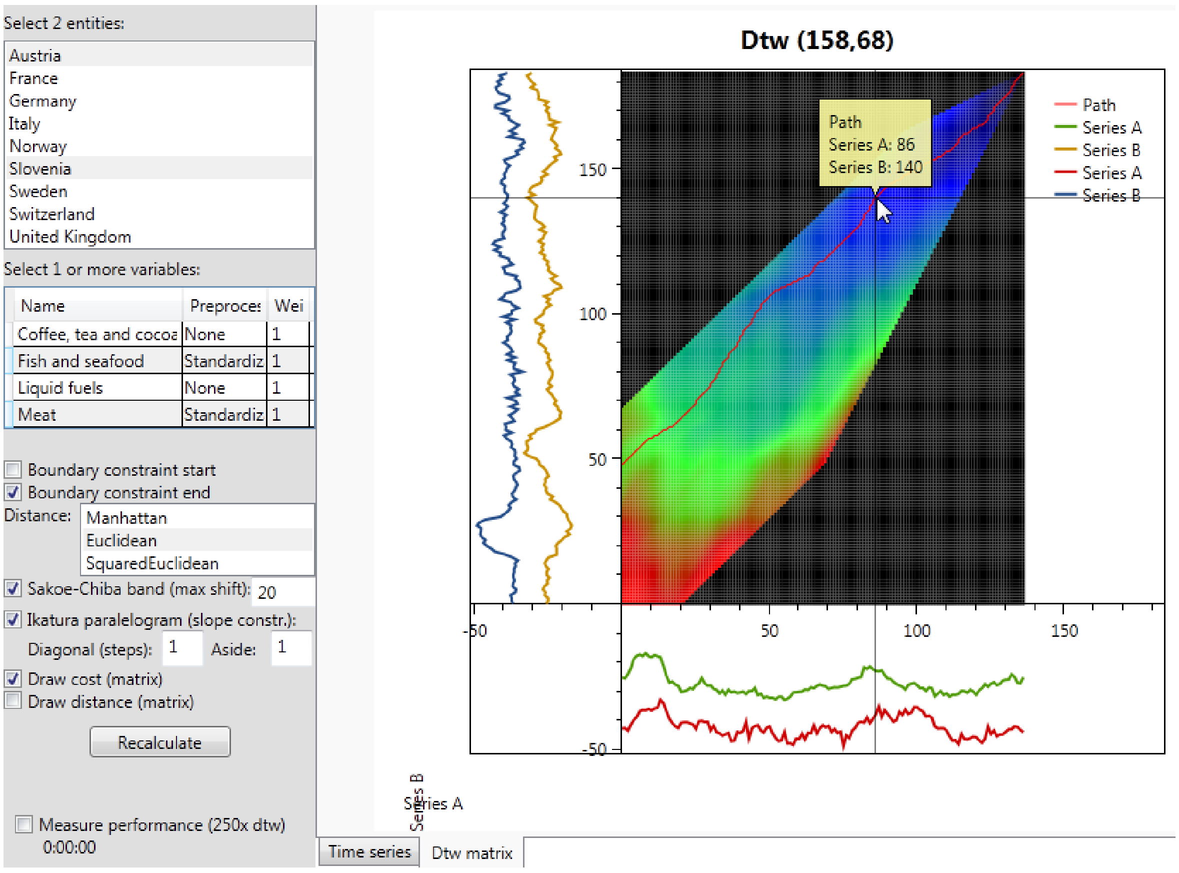
The given implementation of the euclidean distance **works** for the quaternions since it could be interpret as tupples. Another approach could be a difference of modules.

$$d(p,q) = \sum_{i=0}^{4}\sqrt{(p_n)^2} - \sum_{i=0}^{4}\sqrt{(p_n)^2}$$

The other option is to do a different DTW for each one of the components in the quaternion. The memory usage of the algorithm in this approach is theoretically quadrupled. Fortunately we don't need to keep the four cost matrix in memory so we can just accumulate the costs in a variable.

After some deliberation, we stayed with the **second option**. The computational cost of calculating the **square root** of a number is significantly higher than manipulate the memory many times. The perk of this approach is not only that is **faster**, but also there are many implementations of this version of DTW online.

The DTW code used in the final version is provided by the repository found at `https://github.com/doblak/ndtw`.



Select 2 entities:

Austria
France
Germany
Italy
Norway
Slovenia
Sweden
Switzerland
United Kingdom

Select 1 or more variables:

Name	Preproces	Wei
Coffee, tea and cocoa	None	1
Fish and seafood	Standardiz	1
Liquid fuels	None	1
Meat	Standardiz	1

☐ Boundary constraint start
☑ Boundary constraint end
Distance: Manhattan / Euclidean / SquaredEuclidean
☑ Sakoe-Chiba band (max shift): 20
☑ Ikatura paralelogram (slope constr.):
Diagonal (steps): 1 Aside: 1
☑ Draw cost (matrix)
☐ Draw distance (matrix)
Recalculate

☐ Measure performance (250x dtw)
0:00:00

Figure 5.8: Distance matrix from the ndtw repository.

Despite being sort of optimal, **DTW** complexity grows very fast according the collection sizes. One of the ideas that emerged during the development of the project was to delegate the algorithm to an **external server**.

In the end, delegating task proved not to be a good idea since that would have complicated the **deployment of the system**.

In order to decide if the recorded time series is or not the correct gesture the algorithm will accumulate the error of every DTW and compare it to an **experimental error**. This error has been calculated by trial and error.

The model of the custom movements is generated through a **custom scene**. This scene is not packed in any release since is a developer tool. The scenes consists on a trigger which starts a countdown of 3 seconds to allow the user to get ready and starts recordings for about

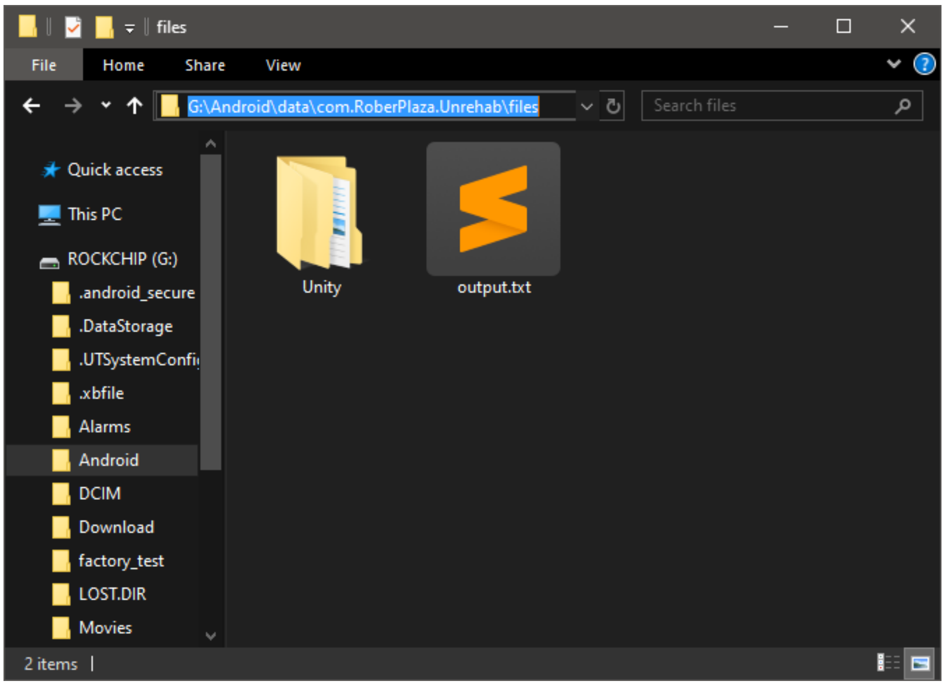


Figure 5.9: Path of the generated file.

3 seconds. The capture is made **frame by frame** so more than 3 seconds may be required if performance is low.

Another approach could have been to capture all the possible samples during 3 seconds and analysing them. We stayed with the first approach since different collection sizes are very punished by the algorithm.

The resulting document is a model using a **domain language**. This domain language uses the following regular expression:

[(-?\d.\d, -?\d.\d, -?\d.\d,-?\d.\d);+]\n+.

Thankfully the grammar is so simple there is no need to implement a language processor in order to analyse it since a few string splits are enough.

The generated models are too complex to be hardcoded, therefore are loaded at **runtime**. In order to manage this, Unity, allows the use of **streaming assets**. Most of the assets of a Unity project are combined at build or pack time, however, sometime is a smart decision or even necessary to load some assets at runtime.

Unity will store all the assets placed in the folder **"StreamingAssets"** to a particular folder of the target system. In order to facilitate the use of streaming assets, the path to the folder will always be found at "Application.streamingAssetsPath", as the absolute path changes between platforms.

In order to access the streaming assets folder in Android we will use a custom asset library called "Better Streaming Assets". Since android project are deployed using an ".apk" file, everything in the project is **compressed**.

In order to access a zipped file in Android is needed to read by **bytes** instead of strings, and is a slow and complicated approach. Better Streaming Assets solves this and allows you to read all the file as a string.

```
public class PoseModel
{
    ...

   public void LoadModel(string fileName)
    {
        string[] fileInLines = BetterStreamingAssets.ReadAllLines(fileName);
        int jointIndex = 0;
        int frameIndex = 0;



        foreach (string line in fileInLines)
        {
            frameIndex = 0;
            // [(...);(...)] -> [(...), (...)]
            string[] rotations = line.Substring(1, line.Length - 3).Split(';');

            foreach (string rotation in rotations)
            {
                // (1,2,3,4) -> [1, 2, 3, 4]
                string[] values = rotation.Substring(1, rotation.Length - 2).Split(',');

                rotationModel[jointIndex, frameIndex]
                    = new Quaternion(float.Parse(values[0]), float.Parse(values[1]),
                    float.Parse(values[2]), float.Parse(values[3]));
                frameIndex++;
            }
            jointIndex++;
        }
    }

    public void SaveModel(string fileName)
    {
        FileInfo file = new FileInfo(Application.persistentDataPath + "/" + fileName);

        if (file.Exists) file.Delete();

        StreamWriter writer = file.CreateText();
        for (int jointIndex = 0; jointIndex < numberOfJoints; jointIndex++)
        {
            writer.Write("[");
            for (int frameIndex = 0; frameIndex < numberOfFrames; frameIndex++)
            {
                Quaternion frameJointRotation = rotationModel[jointIndex, frameIndex];
                writer.Write(string.Format("({0},{1},{2},{3})", frameJointRotation.x,
                    frameJointRotation.y, frameJointRotation.z, frameJointRotation.w));

                if (frameIndex < numberOfFrames + 1)
                {
                    writer.Write(";");
                }
                writer.Flush();
            }
            writer.Write("]");
            writer.Write(writer.NewLine);
        }
    }
}
```

Listing 5.8: Methods to save and load models.

In order to **choose** the next gesture the system first decides if the next movement is going to be tracked by Nuitrack or if it is a **custom gesture**. The information of the next gesture being track by Nuitrack is needed because of the UI since the information showed by the interface is not the same. In order to inform the player about the next movement the avatar will have a **dialog bubble** with the next gesture.

If the next gesture is tracked by Nuitrack nothing will change in the UI. On the other hand, if the next gesture is a custom one a **progress bar** will be displayed. At first the progress bar decreases to allow the user to **get ready**, this will take 3 seconds. Once the bar is depleted it will start refiling for **90 frames**. The reason behind this is simple, is more natural to use time, but the system will lose data if the hardware can't push more than 30 frames per second.

The virtual avatar will stay in screen in order to provide **visual assistance** to the user. During this scene the **mirrored effect** of the camera will not be such.

Finally, the level layout expressed as a FSM will be shown.

$$\begin{tikzcd}
 & & & MM \arrow[dll, "LT"] \arrow[dl, bend left, "LWC"] \arrow[dr, bend left, "LGC"] \arrow[drr, bend left, "LPR"] & & \\
\text{start} \arrow[r] & T \arrow[urr, bend left, "FT"] \arrow[urr, bend right, "ST"'] & WC \arrow[ur, bend left, "GO"] & & GC \arrow[ul, bend left, "GO"] & PR \arrow[ull, bend left, "GO"]
\end{tikzcd}$$

Legend:

- T = Tutorial
- WC = WindowCleaning
- GC = GarbageCollecting
- PR = PoseRecognition
- FT = Finish tutorial
- ST = Skip tutorial
- LT = Load Tutorial
- LGC = Load WindowCleaning
- LT = Load Tutorial
- LWC = Load WindowCleaning
- LPR = Load PoseRecognition
- GO = Game Over

Chapter 6

Results

THE result of the development of the project comprises three different sub-projects. During this chapter we will present and discuss the three different products resulting from the effort, the work distribution of the project as a whole and some estimations of the development costs.

6.1 Work distribution.

The development of the project began on the 13th February, 2019 as an internship to investigate **Skeletal tracking techniques** at Furious Koalas$^{\text{LTD.}}$ and from then on evolved into a whole new project.

The whole project can be divided into 3 different parts. The first part was dedicated to the study and usage of **OpenPose** as well as the development with it. In the second stage We developed a **C++ implementation of DTW** and learnt about the use of **game engines**. At the last part of the project we documented everything and developed the **final game**.

6.1.1 Stage One, OpenPose.

The original idea was just to evaluate and compare **skeletal tracking** methods. The first library regarding this concern was OpenPose. We have spoken about OpenPose previously in this document. OpenPose was the initial approach and it can be good enough or even **very good** for some use cases, but not his.

Sprint one.

The time period from 13th February, 2019 to 20th February, 2019, was dedicated to the construction of the library "OpenPose". As previously mention in chapter 3, the development environment of this library is **very closed** and took nearly a complete week to setup everything related to it.

Sprint two.

From the 21st February, 2019 to 1st March, 2019, the time was spent into the development of a few **technical demos**. The purpose of this demos was to understand what is OpenPose

capable of. This demos were developed using the build tool inside OpenPose, therefore, these demos were **not standalone** programs.

Sprint three.

The last sprint related to OpenPose covered the dates from 4th March, 2019 to 18th March, 2019. During this time period was published a repository that contains a **simplified** way to use OpenPose. Another result of this sprint is the release of **standalone demos** for different utilities.

Final product.

The final release of this stage could be found at `https://bitbucket.org/furious_koalas/skeletaltrackingexamples`. This repository requires you to have OpenPose build and installed. Once it's build, the developer should be able to build some documentation via **Doxygen**.

6.1.2 Stage two, Dynamic time warping.

At this time is when we decided to present this project as a work. The original idea was the same that the final one. Since the device was about to be received we start to advance in other areas, in this case, managing a DTW implementation.

Sprint four.

The time period between 19th March, 2019 and 29th March, 2019 was dedicated to the research abut **Dynamic Time Warping**. The research was not just about the implementation, but also about optimisation, the main resources were [19] and [1].

During this sprint we started thinking about the **joint model** used in the final implementation. Another important milestone of this stage was the discovery of the **ndtw** repository containg the code use in the first release.

Sprint five.

During the weeks form 1st April, 2019 to 18th April, 2019 we developed and published an implementation of DTW for the language C++. This implementation allows the use of different type of time series and, by means of taking advantage from the **functional** capabilities of the language, it also allows the use of different **distance** functions.

This implementation was done in C++ because originally, "UnRehab" was about to be developed in **Unreal Engine 4**. We spent weeks trying to make Nuitrack work in UE4. In the end we didn't manage to make it work so we switched the game engine to **Unity**.

There was also developed an implementation of a **docker container**. This container deploys a **flask python** HTTP server that offers a DTW-processing service. This server uses a python hook for R, called "rpy2". The code can not be found because the repository was deleted when this approach was discarded.

Sprint six.

From the 23rd April, 2019 to 10th May, 2019 the time was invested into doing builds of Android in UE4. As mentioned before we didn't manage to make Nuitrack work in Unreal Engine 4. During this time many mini games were developed in order to tests Unreal's Android build tool.

During this test we imported the DTW library to an Unreal Engine 4 project with fruitful results. The games were not uploaded since they were not relevant.

Sprint seven.

The week from the 11th May, 2019 to 18th May, 2019 was spent in meetings with the team, planing the scope of the final project and filling out the paperwork to validate the internship.

Final product.

The result of this step translates into another repository with an implementation of the DTW algorithm in C++. Despite the resulting code not being very extensive, the commit history reveals a lot of different approaches to the algorithm.

All the code can be found at `https://github.com/RoberPlaza/DTW`. The final approach could have been transformed into a **header only** library, but by that time we had already realised we were not going to code the rest of the project in C++.

```
template < class T, double (*DistanceMeasure)(const T&, const T&)>
class GenericDTW
{
public:

    GenericDTW (const std::vector<T>& collX, const std::vector<T>& collY, const uint32_t& step = 0)
        : xSeries(collX), ySeries(collY), costsTable(collX.size(), collY.size()), sakoeStep(step)
        {} ;

    const Path& getPath()
    {
        if (calculated < CompletitionLevel::PathCalculated)
            calculatePath ();
        return bestPath;
    };

    const double& getCost()
    {
        if (calculated < CompletitionLevel::MaxCostCalculated)
            calculateCost();

        return minCost;
    };

    const CostTable& getCostTable()
    {
        if (calculated < CompletitionLevel::TableCalculated)
            calculateCosts();

        return costsTable;
    };

protected:

    CostTable               costsTable;

    Path                    bestPath;

    const std::vector<T>&   xSeries;

    const std::vector<T>&   ySeries;

    uint32_t                sakoeStep;

    double                  minCost     = 0.0;

    CompletitionLevel       calculated  = CompletitionLevel::NotCalculated;

    virtual void calculateCosts ()
    {
        for (uint32_t i = 0; i < xSeries.size(); i++) {
            for (uint32_t j = 0; j < ySeries.size(); j++) {
                costsTable(i, j) = DistanceMeasure(xSeries[i], ySeries[j]);
            }
        }
        calculated = CompletitionLevel::TableCalculated;
    };

    void calculatePath ()
    {
        if (calculated < CompletitionLevel::TableCalculated)
            calculateCosts ();

        uint32_t i = xSeries.size() - 1;
        uint32_t j = ySeries.size() - 1;


        bestPath.push_back(std::make_pair(i, j));

        while ( i > 0 && j > 0) {
            if (costsTable(i - 1, j - 1) <= costsTable(i - 1, j)
                &&
                costsTable(i - 1, j - 1) <= costsTable (i, j - 1))
            {
                i--;
                j--;
            }
            else
            if (costsTable(i - 1, j) <= costsTable(i - 1, j - 1)
                &&
                costsTable(i - 1, j) <= costsTable(i, j - 1))
            {
                i--;
            }
            else
            {
                j--;

            }
            bestPath.push_back(std::make_pair(i, j));
        }

        while (i > 0) {
            i--;
            bestPath.push_back (std::make_pair (i, j) );
        }

        while (j > 0)  {
            j--;
            bestPath.push_back (std::make_pair (i, j) );
        }

        calculated = CompletitionLevel::PathCalculated;
    };

    void calculateCost ()
    {
        switch (calculated) {
            case CompletitionLevel::NotCalculated:
            case CompletitionLevel::TableCalculated:
            case CompletitionLevel::PathCalculated:
                calculatePath ();
            case CompletitionLevel::MaxCostCalculated:
                for (const auto& pair : bestPath)
                    minCost += costsTable[pair];
        }
        calculated = CompletitionLevel::MaxCostCalculated;
    };
};
```

Listing 6.1: C++ DTW implementation.

6.1.3 Stage three, Development of UnRehab.

The last stage was in which the development of the game was carried out. This was done using Unity and Visual Studio. The version control system used was a private **Subversion** server and the candidates were uploaded to Github.

Sprint eight.

During the sprint that took place between the dates 20th May, 2019 to 3rd June, 2019. During this time the team got used to Unity3d and began to animate the chosen virtual avatar.

The avatar was originally "**Robot Kyle**[1]". The idea was to keep the avatar as neutral as possible. In spite of this, the final avatar ended up being "**Unity-Chan**[2]", a cartoon depiction of the engine.

Another important milestone of this stage was the script for **mirroring the camera.** Was necessary to mirror the camera for some mini games. This is so because some test done to random users showed that mirror camera was more **natural** to them.

Sprint nine.

The ninth sprint happened between 3rd June, 2019 and 17th June, 2019. During this sprint we developed the first version of the 3 mini games. This versions were unpolished and unintuitive.

[1]To be found at: `https://assetstore.unity.com/packages/3d/characters/robots/space-robot-kyle-4696`
[2]To be found at: `https://assetstore.unity.com/packages/3d/characters/unity-chan-model-18705`

During this sprint the furniture was added as level decoration. The empty levels made most of the test users felt distressed. The furniture layout was intended to make the user feel at home.

Sprint ten.

During the dates 17th June, 2019 and 1st July, 2019. This sprint brought along the pose recognition infrastructure. We have already spoken about how the pose recognition is done.

The first approach to the recognition infrastructure had two custom gestures, **hands up** and **hands down**. At this point in time the user avatar was mapped by 11 joints. Short time afterwards was when this number was reduced to 9.

The asset streaming module was also used in this iteration. The first attempt to integrate the module in the system was by means of **hardcoding it**. This was not possible due to the massive length of the model format.

Final sprint.

The last two weeks of project development took from 1st July, 2019 to 15th July, 2019. During this development cycle most of the **UI elements** were added and most of the mini games were polished.

The most remarkable milestone about this stage was the **remake** of the window cleaning game. In the first release of UnRehab the user will control **two hands** during the window cleaning game.

The last important step about UnRehab's development was the calculation of the model error. This error was calculated by means of trial and success. The final error was calculated so the user doesn't score a goal if s/he does not move and scores if s/he tries.

The public repository for the work is located at `https://github.com/RoberPlaza/RehabPlatform`.

6.2 First UnRehab candidate.

We haven't spoken about the final product in the previous section because we wanted to explained the final result in depth.

The version **v1.0** of UnRehab is the result of this project. This project packs five different scenes and three different mini games. The user can navigate the mini games through four colour capsules.

6.2.1 Tutorial scene.

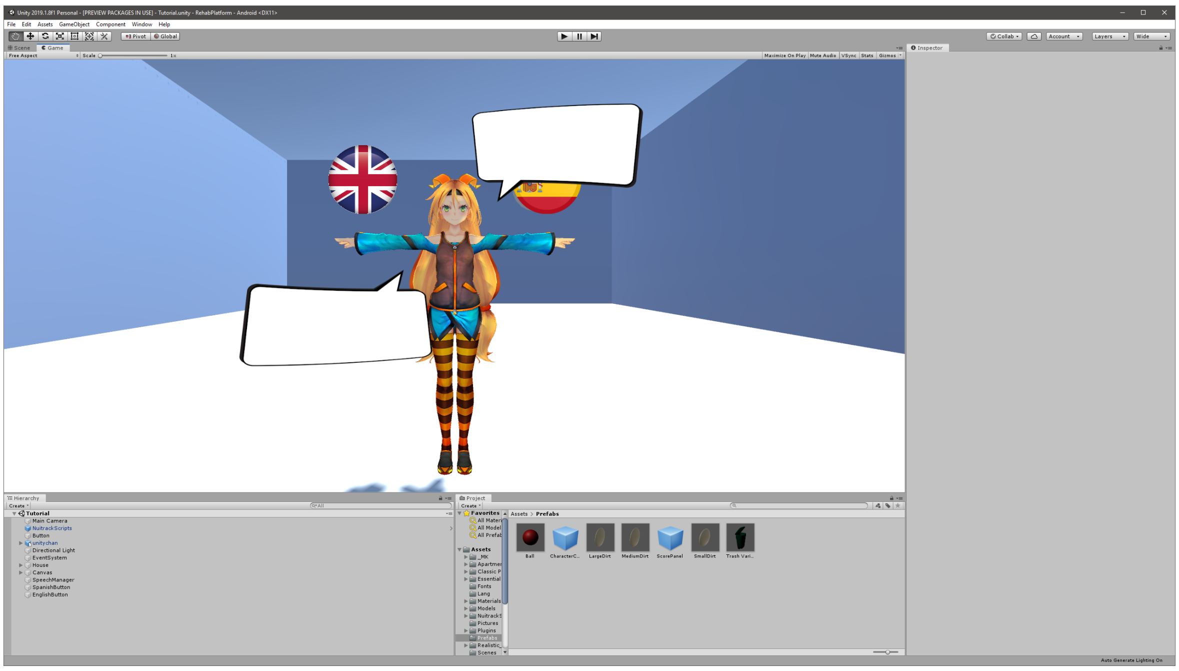


Figure 6.1: Screenshot of the tutorial.

The first scene, and the starting one is the **Tutorial**. The tutorial is an introduction to the system. Through the introduction the avatar will talk to the user using two **chat bubbles**. Each sentence will change the bubble in order to 'tell' the user there is more dialog.

The first thing users have to do is select a language. This can be done by touch one of the flag buttons in the scene. The tutorial is the only multilingual scene, during rest of the game text will be in **English**.

Usually every development environment has its own solution to multiple language support. The solution purposed by Unity is just to bulky and inconvenient. This solution goes through the use of **XML** files in order to store every text in the game in every language.

In the end this was not done this way because C# is not a very good language to manipulate any kind of markup language. These inconveniences, hand in hand with the problem of **Android asset streaming** made impossible to add consistently multiple language support.

The game will write a message every 5 seconds. After some dialog the game will ask the player to touch a level gate. Time after will be revealed that the tutorial can be skipped by means of **waving** to the camera.

6.2.2 Main menu scene.

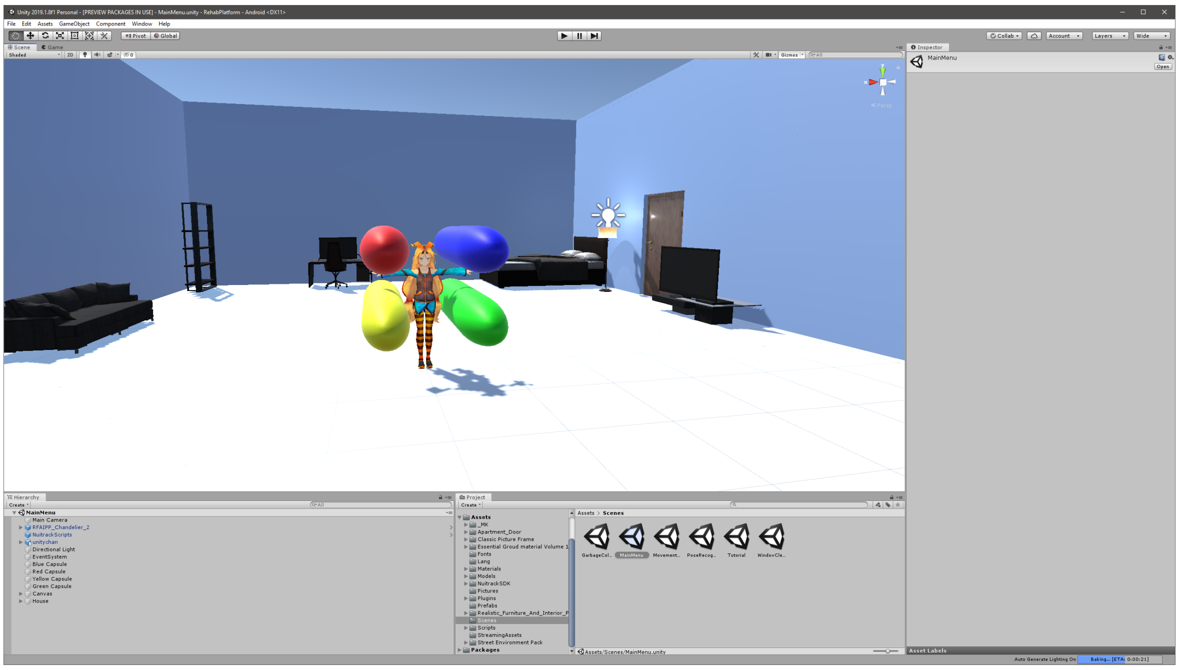


Figure 6.2: Screenshot of the main menu.

The 'level nexus', the gateway to every level is the **MainMenu** scene. The four capsules are the **level gates**. After touching them for 5 seconds a new level will start loading. The user may touch several capsules but the **first gate to reach** 5 seconds will trigger.

The level mapping related to colours is as follows:

- **Blue**: loads Window Cleaning.
- **Red**: loads Garbage Collector.
- Yellow: loads Pose Recognition.
- Green: allows to repeat the Tutorial.

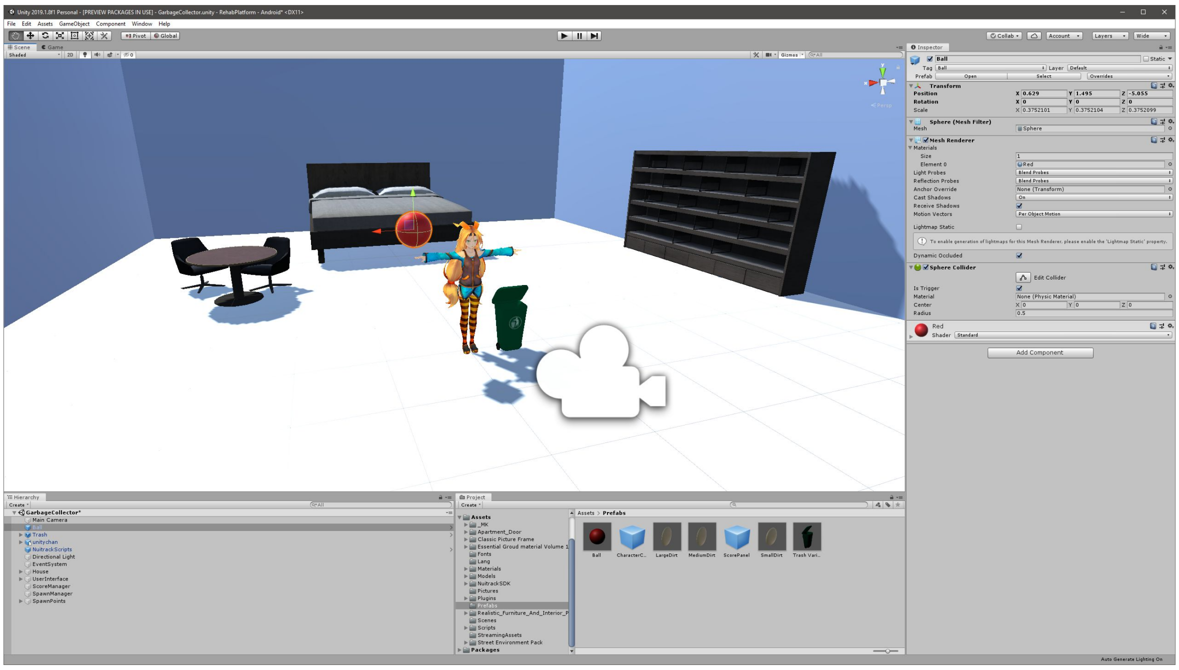
The avatar is **fixed to the ground** and the user can make the avatar move the upper body. The camera is mirrored in this mini game.

6.2.3 Garbage collecting scene.

The first scene that holds a mini game is a blend of collecting garbage and basketball. The target of the user in this mini game is to score points by means of guiding a ball to a trash bin.

The target score of the first release is **20**. There are also 5 spawn points for the ball. Within the code there is a mode that allows random spawn of the trash bin. Although only one boolean variable needs to be changed in order to play this mode, there is no possible way to run it in the final apk.

The hitbox trigger of the trash bin is extended to the infinite in order too help the player with the deposits. The spots where the ball can be spawned are distribute on a semi circle around the avatars head and in front of the face.



Ball — Static: off
Tag: Ball — Layer: Default
Prefab: Open | Select | Overrides

Transform
Position X 0.629 Y 1.495 Z -5.055
Rotation X 0 Y 0 Z 0
Scale X 0.3752101 Y 0.3752104 Z 0.3752099

Sphere (Mesh Filter)
Mesh: Sphere

Mesh Renderer
Materials
Size: 1
Element 0: Red
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Cast Shadows: On
Receive Shadows: ✓
Motion Vectors: Per Object Motion
Lightmap Static: ☐
To enable generation of lightmaps for this Mesh Renderer, please enable the 'Lightmap Static' property.
Dynamic Occluded: ✓

Sphere Collider
Edit Collider
Is Trigger: ✓
Material: None (Physic Material)
Center X 0 Y 0 Z 0
Radius: 0.5

Red
Shader: Standard

Add Component

Figure 6.3: Screenshot of the garbage collector game.

The original trash bin model was very different to the final one. The first model was a bin that resembled a **basket hoop**, was finally removed since the **low game resolution** made the model very confusing. Just like in the main menu, the avatar is placed in a point and the camera is fixed and mirrored.

6.2.4 Window cleaning scene.

The longest mini game, at least according to time required is the window cleaning mini game. We can see in the middle of the scene a dirty window. The camera position of the game allows the user to watch the window up close and create deeper immersion.

The avatar of this game are two hands both mapped to the real hands of the user. The hands are mapped 'upside down' because the camera is **vertically flipped**.

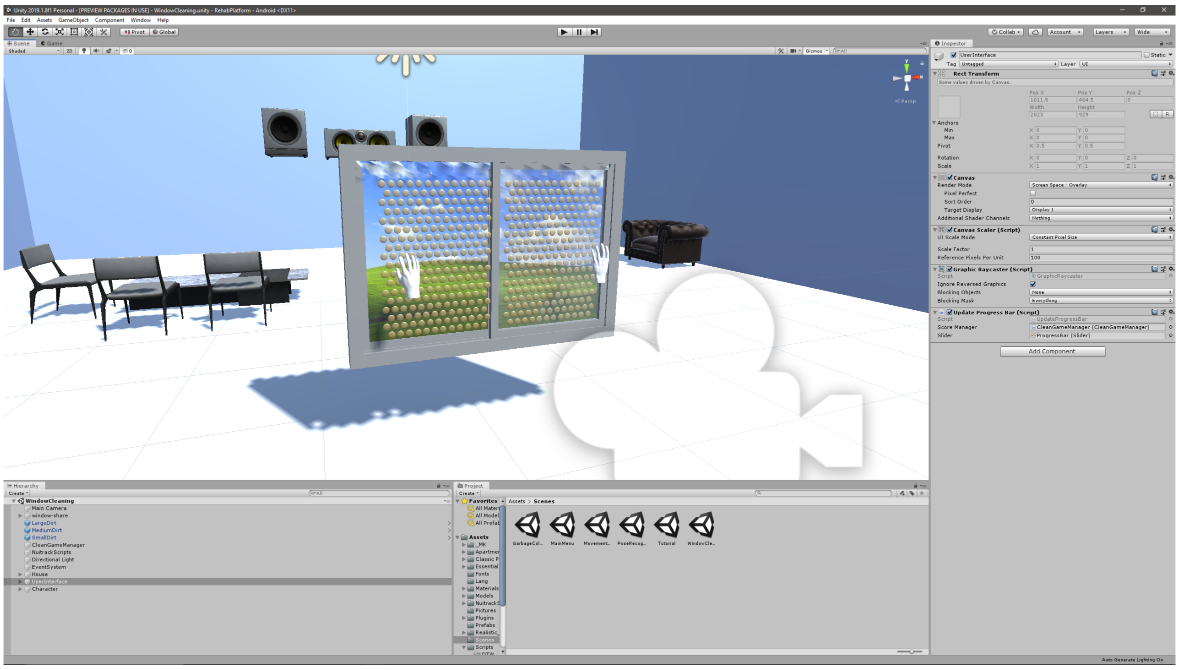


Figure 6.4: Screenshot of the window cleaning game.

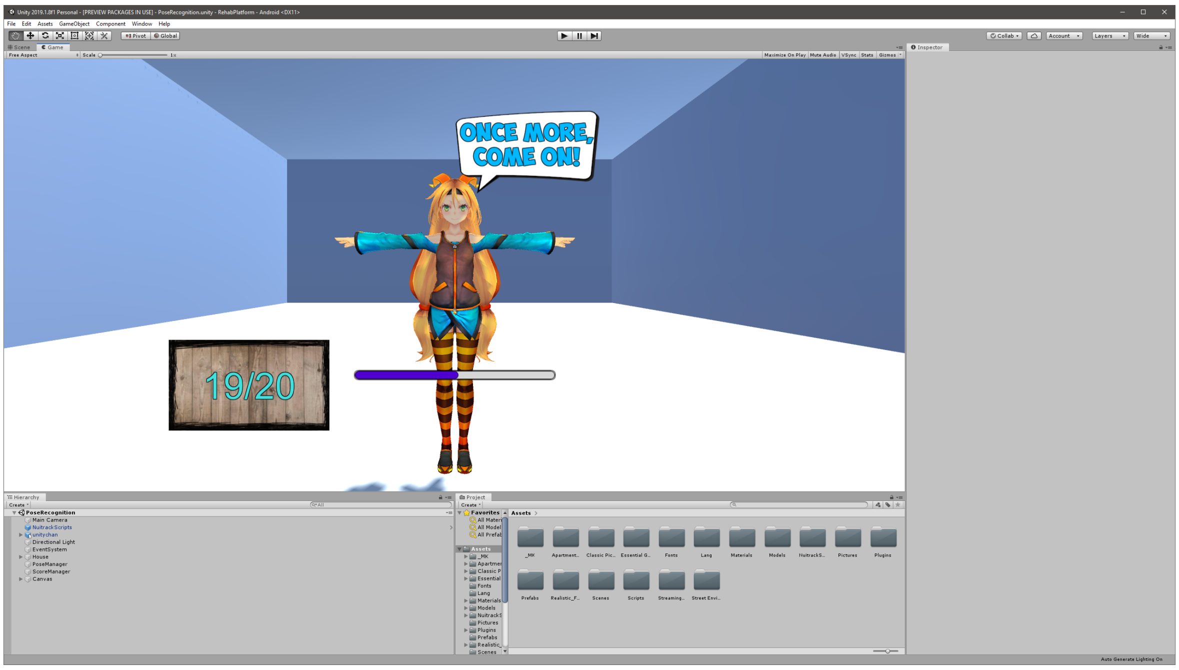


Figure 6.5: Screenshot of the pose recognition game.

6.2.5 Pose recognition scene.

The last scene packed is the gesture recognition game. The first gesture the user has to make will always be waving. The progress bar will only appear when the next pose has a time limit. This progress bar first empties itself and then starts capturing as previously mentioned. Poses that require progress bar are **worth double the points**.

The avatar is placed in this scene in order to offer information about the current pose. The camera mirroring was dispensed with in this scene.

6.2.6 Movement generator scene.

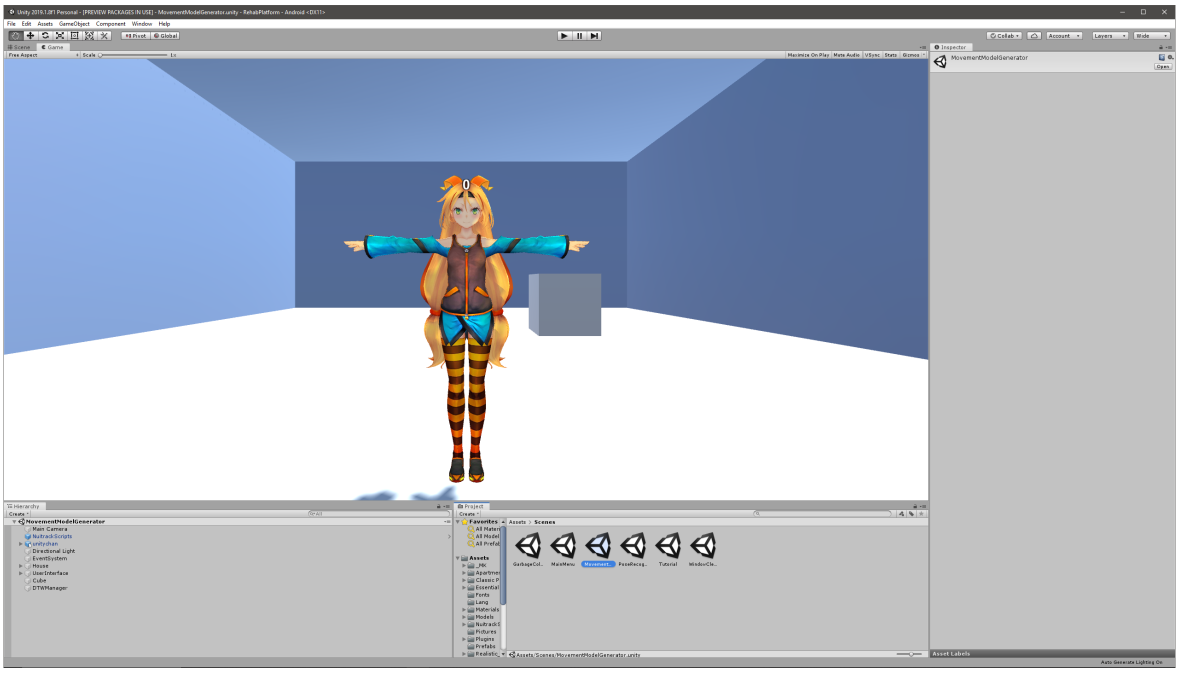


Figure 6.6: Screenshot of the movement generator level.

At last, but not least, the sixth scene is the **developer only** model generator. This scene is not packed in the final release, but, you can play it from the source project.

The little square at the left of the avatar triggers the **capture event**. After the beginning of this event you will see a progress bar similar to the one at the **pose recognition** scene. The system will give you three seconds to get ready and will capture a **gesture module** for another 3 seconds.

6.3 Resources, cost and viability.

In order to develop UnRehab there were not many resources needed. On the other hand, if a large enterprise would like to create a similar platforms would face some extra costs.

The most expensive piece of hardware for this project was the computer. Fortunately my rig is much more powerful than needed for this project, for this reason it won't be taking into account. However, in order to run Unity3d to develop games with discrete graphics a $600 desktop computer, roughly €550 would do.

The second most expensive device in the whole project was the **TVico**. This device currently retails at **$339.99**, but has periodic sales in which its price is reduced to **$279.99**. These figures are translated into €305 and €250 respectively. This device was borrowed during the development but will be taken into account for the final costs calculation.

A license of Nuitrack was included with the TVico. It won't be taken into account but it has a value of **$99.99** one time charge. This license allows commercial usage.

In order to test the TVico the addition of an HDMI **monitor** is a must. Some inexpensive HDMI monitors can be found for around **$120** or **€110**.

Visual Studio was used to developed UnRehab. The community version of visual studio is free but doesn't allow commercial usage. Visual studio licenses go from **$45 monthly** to **$800 yearly** after an initial payment of $1200.

Fortunately **Visual Studio code** is a free multi-platform solution and part of the project was done with this vs code. Most of the developers can use this option instead of the expensive one. This license won't be taken into account.

In order to publish games done with Unity3d is needed to pay a fee of **$125** monthly. A while ago android license was not included in the original $125, luckily nowadays everything is included within the **Unity pro** license.

An extra cost of $50 or €45 will be added as a deposit for extra **game assets**. Assets like models, animations or code are a cheap and easy solution for many problems, specially since the **Unity store** is very well integrated into the engine.

The idea of this extra $50 is not to use a **generic model**. The used avatar is very well animated and very well finished, but a robot, artist dummy or patient will make the project look more serious.

The most expensive asset of this project can easily be the **developer/s**. The first of the 3 stages was the less relevant to the work, so we are not going to count it. The project took from 19th March 2019 to 15th July, that means **4 months of work** (if we round it up).

According to the webpage **"Glassdoor" [6]**, the average game developer at Barcelona has a payroll of approximately **€36776** yearly. Given that Barcelona is a more expensive city and we are junior engineers it would be unrealistic to apply for such a salary.

We consider a good figure for jobs like these would be around **€28000 gross** yearly. This means 4 payments of €2000 and $\frac{4}{6}$ of an extra payment (Considering 14 payments of €2000) for a value of **€8333.33** or **$9285**.

The costs of the project add up to the figure of **$10,430 or €9,373**. See figure 6.1 for further details.

	Price in dollars	**Price in euros**
Unity-capable computer:	600	550
TVIco and Nuitrack:	0	0
HDMI monitor:	120	110
Visual studio code:	0	0
3 months of Unity Pro:	375	335
Custom avatar:	50	45
Unity Developer for 4 months:	9285	8333
Total:	$10,430	€9,373

Table 6.1: Cost breakdown.

Chapter 7

Concluding Remarks

In this chapter we will describe the conclusions after the development project. This chapter is divided into 3 different sections regarding important aspects of the final product. First, we will have an overview about what objectives were accomplished, then we will discuss the difficulties of the project and, at the end, we will comment on some improvements and future work for the system.

7.1 Accomplished objectives.

The objectives of the system were specified at chapter 3. It is something wonderful to say that all of them were accomplished.

The original objective of UnRehab was to create a scalable architecture of rehabilitation exercises that motivates and encourages the user through gamification and has a small hardware deployment.

- **Reject the use of traditional controllers.** This sub-objective was achieved just by means of using the TVico. At first it was difficult to figure out a layout that allows the player to take full control of the game just by moving the body. We based the game controls on the ideas of games developed by 3DiVi, the company that created and delivers TVico. This games tend to use the capture of the camera and add render objects on top. This does not resemble the idea of using a virtual avatar, but the games had touchable triggers in order to interact with the game.
- **Possibility to include new games.** The solution is sub-objective was the election of a game engine as a development tool. We have already spoken about the concept of a video game level, in this case scene. Through the development of new scenes, new games can be added. A new mini game will require either a new capsule of a different colour or to rebuilt the main menu. The layout of the Main Menu scene is just not too good, but it can easily be changed.
- **Small, portable hardware infrastructure.** In the end we managed to packed the final application in to just one file. The solution to this sub-objective was simple, but through this document we have spoken about a possible server to calculate the

DTW. The deploy of the system is so small TVico doesn't even need internet access. Nevertheless, a future version of the system may need to have internet connection because the current version doesn't gather data about the patients and that could be interesting.

- **Medium-to-low activity collection of exercises.** The mini games developed through the process were not very intense, so we can mark this sub-objective as accomplished. Most games allow the user just to 'move around' at his/her own rhythm, the only mini game that doesn't follow this rule is the one about pose recognition, doesn't appear in every stage of the game and when it does is optional and very rewarding in terms of score. Despite of most of the games being very static it can be done faster with harder movement.
- **Intuitive, user-friendly UI.** We will like 'tick' this sub-objective as accomplished but UI usability is very subjective. In the end the UI ended out very lessen. The mayor percentage of GUI's display information regarding current score and target score. All the remaining elements are, for most of the times, irrelevant in terms of game play.
- **Movement-based control system.** This sub-objective is related to the first one. All the interaction in the game in carried out by means of the user moving. The most characteristic movement is the use of the hands at the window cleaning mini game. Since the capsules of the main menu are four this could be changed in order to be resemblant to the arrow keys.

7.2 Future work.

Even though we're proud of the result the system could be incremented and be added a lot of features. The following list enumerates and explain some possible enhancements for the system:

- **Remake of the main menu:** We have spoken previously about the need to modify the layout of the level. A list of levels should be added to the UI and the possibility to select and load levels should be accessed via the capsules.
- **Creation of a custom avatar:** During the costs analysis we mention the purchase of a custom model. The used character is serviceable and functional but is also copyrighted and anything-but-related to topics we discuss in this work. A gender-neutral mannequin or dummy would be perfect.
- **Addition of music:** Sound is always the forgotten part of video games. Sound was not added during development due to the lack of speakers in the available screens. A few sound clues and some soft music could easily rise the game quality.
- **Install a data collection and analysis module:** A big addition to the system could be the addition of a machine learning module. The game has information regarding

joint position, average movement, time taken to complete the games... This data can be fed to a data mining system and farm some interesting information to offer patients a better treatment and/or environment.

7.3 Difficulties faced and lessons learnt.

Through the development of the project we have faced a lot of hardships and learned a lot of things along the way. Has been a grateful experience and there was never a challenge without reward. The most important problems and lessons are the following:

- **Unreal Engine 4 Android build fails:** We were trying to make an Android build work for almost 2 whole weeks. This was the major take down of the project because in the end was going to be both a TFG and a TFC for the CEDV. This inconvenience was not the only one, with a new engine comes a new language and a new approach to solve problems. Also, after using the monstrous tool Unreal Engine 4 is everything feels short. Lesson learnt from this is that a timely retreat is a victory.
- **OpenPose construction:** The construction of OpenPose took almost a week. Was a very difficult step due to being the first one of the path. This was the first real deadlock of the project. Lesson learnt from this experience was that just because you red the manual doesn't mean you know how to work.
- **Creation of the gesture models:** Was not really a hard section, but we dragged him a long time until finally facing it. We thought about is so much that when the time arrived implementation was successful at first. This teaches us how important is to thing about problems before trying to solve them.

APPENDICES

Appendix A

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

2. VERBATIM COPYING

3. COPYING IN QUANTITY

Reference

[1] Pablo León Alcaide. *AG-COSTANE: Algoritmos genéticos para el cálculo del centroide de un grupo de series temporales mediante una distancia no Euclídea*. 'tfg'of esi-uclm. July 2017.

[2] National Stroke Association. *National Stroke Association What is a Stroke?* 2019. URL: `https://www.stroke.org` (visited on 07/20/2019).

[3] Kent Beck et al. "Manifesto for agile software development". In: (2001).

[4] Chris Biggin. *Introducing the 2018 Gamer Segmentation Syndicated Report*. 2018. URL: `https://www.eedar.com/post/introducing-the-2018-player-segmentation-syndicated-report` (visited on 07/18/2018).

[5] Sait Celebi et al. "Gesture recognition using skeleton data with weighted dynamic time warping." In: *VISAPP (1)*. 2013, pp. 620–625.

[6] glassdoor. *Game Developer Salaries in Barcelona, Spain Area*. 2018. URL: `https://www.glassdoor.com/Salaries/barcelona-game-developer-salary-SRCH_IL.0,9_IM1015_KO10,24.htm` (visited on 03/14/2018).

[7] Alexei Gritai, Yaser Sheikh, and Mubarak Shah. "On the use of anthropometry in the invariant analysis of human actions". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 2. IEEE. 2004, pp. 923–926.

[8] Adrián Ollero Jiménez. *Allegra: Gamifcation-based Tool to Practice Melodic Dictation*. 'tfg'of esi-uclm. July 2019.

[9] Ralph Johnson and John Vlissides. "Design patterns". In: *Elements of Reusable Object-Oriented Software Addison-Wesley, Reading* (1995).

[10] Choong Hee Kim et al. "Automated Camera-Based Estimation of Rehabilitation Criteria Following ACL Reconstruction". In: *arXiv preprint arXiv:1810.11087* (2018).

[11] Keith Lohse et al. "Video games and rehabilitation: using design principles to enhance engagement in physical therapy". In: *Journal of Neurologic Physical Therapy* 37.4 (2013), pp. 166–175.

[12] Ingo Maier, Tiark Rompf, and Martin Odersky. *Deprecating the observer pattern*. Tech. rep. 2010.

[13] Diego Molero Marín. *HoloMusic XP: Gamification-based System for Teaching Music and Piano using Mixed Reality*. 'tfg'of esi-uclm. July 2018.

[14] newzooo. *Introducing Newzoo's Gamer Segmentation^{TM}: The Key to Understanding, Quantifying, and Reaching the New Era of Game Enthusiasts*. 2019. URL: https://newzoo.com/news/introducing-newzoos-gamer-segmentation-the-key-to-understanding-quantifying-and-reaching-game-enthusiasts-across-the-world (visited on 04/24/2019).

[15] World Stroke Organization. *https://www.world-stroke.org/*. 2019. URL: https://www.world-stroke.org/ (visited on 07/17/2019).

[16] George Papandreou et al. "Towards accurate multi-person pose estimation in the wild". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4903–4911.

[17] Deva Ramanan and Xiangxin Zhu. "Face detection, pose estimation, and landmark localization in the wild". In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Citeseer. 2012, pp. 2879–2886.

[18] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[19] Hiroaki Sakoe and Seibi Chiba. "Acoustics, Speech and Signal Processing". In: *IEEE Transactions on* 26 (1978), pp. 43–49.

[20] Jordan Sirani. *Top 10 Best Selling Video Games of All Time*. 2019. URL: https://www.ign.com/articles/2019/04/19/top-10-best-selling-video-games-of-all-time (visited on 05/17/2019).

[21] National Stroke Association UK. *National Stroke Association UK What is a Stroke?* 2019. URL: https://www.stroke.org.uk (visited on 07/20/2019).

Este documento fue editado y tipografiado con LaTeX empleando
la clase **esi-tfg** (versión 0.20181017) que se puede encontrar en:
`https://bitbucket.org/arco_group/esi-tfg`

[respeta esta atribución al autor]