Development of a Biofeedback System for Motor Rehabilitation Support

Sara Margarida Almeida Alberto

Thesis to obtain the Master of Science Degree in

Biomedical Engineering

Supervisor: Professor Miguel Pedro Tavares da Silva

Co-Supervisor: Terapeuta Alexandra Maria Gomes Leão de Morais e Castro

Examination Committee

Chairperson: Professor Fernando Manuel Fernandes Simões

Supervisor: Professor Miguel Pedro Tavares da Silva

Member of the Committee: Professor Carlos Miguel Fernandes Quental

Member of the Committee: Professor Pedro Paulo Valente Gentil Soares Branco

May 2018

Acknowledgments

I would like to thank my supervisor at IST, Professor Miguel Tavares da Silva, for introducing me to the world of biomechanics and always offering support and advice and guiding me when I felt overwhelmed or lost. I would also like to thank the Professor for providing the equipment necessary to make this thesis possible. I would also like to show my gratitude to my co-supervisor, Alexandra Maria Gomes Leão de Morais e Castro, for the opportunity to test my work and also for the enthusiasm, energy, suggestions and availability shown throughout the tests.

I would also like to thank all the volunteers who participated in the tests, since this work only has meaning due to their participation.

To my dear friends Afonso Fernandes and Miguel Revez, who were by my side for the past six years and became like family, I am grateful for all the ways you helped me through the difficult moments and cheered with me on the happy ones. A huge thank you to André Carreira, for the superhuman patience and for making me slow down and relax when I truly needed it.

Last but not least, the biggest thank you to my mother for always clearing all my doubts and to my father for trusting me to make the right choices for myself. I would not be who I am without everything you taught me.

Resumo

Os avanços tecnológicos em aquisição de movimento, jogos e Realidade Virtual têm levado ao aparecimento de aplicações nas quais é possível interagir com ambientes virtuais em tempo real através de movimento, recebendo *feedback*. A utilização destes sistemas para a criação de jogos sérios para reabilitação motora é uma ideia ainda em desenvolvimento. O propósito desta tese é a criação em Unity de uma aplicação que permita aos utilizadores realizar exercícios terapêuticos no contexto de um jogo no qual interagem com um ambiente virtual através de um sistema de sensores inerciais (F.A.B.™ System), recebendo *biofeedback* visual em relação a postura e movimento. A aplicação permite registar utilizadores numa base de dados e a aquisição e visualização de dados relevantes. Foi criado um jogo para apoio a reabilitação motora de pacientes entre os sete e os treze anos de idade que sofreram uma luxação do cotovelo. De modo a ajustar os parâmetros do jogo à população em estudo, foi criado um grupo de controlo de onze sujeitos que jogaram o jogo e providenciaram o seu parecer qualitativo. A aplicação foi depois testada num sujeito com a patologia em seis sessões de terapia. Os resultados obtidos demonstram interesse na aplicação e um aumento de motivação do utilizador, mas não foi possível avaliar a influência na eficiência da terapia devido a constrangimentos temporais e a um número de utilizadores reduzido. Estes resultados encorajam o desenvolvimento e avaliação a longo prazo de aplicações de *biofeedback* com base em jogos para utilização em reabilitação motora.

Palavras-chave: sensores inerciais; Unity; F.A.B. System; *biofeedback*; jogos sérios; luxação do cotovelo.

Abstract

The advances in human motion tracking systems, gaming technologies and Virtual Reality have led to the creation of applications where it is possible to interact with virtual environments in real time through the performance of motions, receiving feedback. The use of these systems to create serious games for physical rehabilitation is an idea that is still in development. This thesis proposes the creation in Unity of an application that would allow patients to perform therapy exercises by playing a game in which they interact with a virtual environment through the use of an inertial sensor system (F.A.B. System), receiving visual biofeedback regarding posture and movement. The application registers users in a database and after each session it is possible to save and display relevant data. A game was created with the aim of aiding in the physical rehabilitation of patients aged seven to thirteen years old who suffered an elbow dislocation. In order to adjust the game parameters to the target population, a control group of eleven subjects was created. Subjects played the game and provided qualitative feedback. The application was then tested on one subject suffering from the determined pathology, who played the game in six rehabilitation sessions. The results obtained show interest in the application and an increase in patient engagement, although it was not possible to assess influence on therapy efficiency, due to time constraints and small sample size. These findings encourage the further development and long-term assessment of game based biofeedback applications in motor rehabilitation contexts.

Keywords: inertial sensors; Unity; F.A.B. System; biofeedback; serious games; elbow dislocation.

Table of Contents

List of Figures

List of Tables

List of Symbols

Conventions

a, A, α	Scalar
$\boldsymbol{a}$	Column vector
$\boldsymbol{A}$	Square matrix
$\mathring{q}$	Quaternion

Superscripts

$\boldsymbol{R}^T$	Transpose of a matrix
$\boldsymbol{R}^{-1}, \mathring{q}^{-1}$	Inverse matrix or quaternion
$\overline{\mathring{q}}$	Adjoint of a quaternion
$\|\mathring{q}\|$	Norm of a quaternion

Latin Symbols

$\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$	Basis vectors of a generic reference frame
$\boldsymbol{R}$	Rotation matrix
$\boldsymbol{R}_q(\mathring{q})$	Rotation matrix in terms of quaternion components
$\boldsymbol{R}_S^A$	Rotation matrix which relates rotated reference frame S in relation to reference frame A
$\boldsymbol{R}_A, \boldsymbol{R}_S$	Rotation matrices representing object orientations
$\boldsymbol{R}_z(\psi)\ \boldsymbol{R}_y(\theta)\ \boldsymbol{R}_x(\phi)$	Rotation matrices about the reference frame axis
x, y, z	Cartesian coordinates
$\boldsymbol{Q}(\mathring{q})$	Quaternion matrix
$\mathring{q}_R(\boldsymbol{R})$	Matrix to quaternion mapping
u	Unity distance units

Greek Symbols

ψ, θ, ϕ	Angles of rotation

Abbreviations

2-D	Two-dimensional
3-D	Three-dimensional
AI	Artificial Intelligence
API	Application programming interface
AR	Augmented reality
CVA	Cerebrovascular accident
IMU	Inertial Measurement Unit
IR	Infra-red
LCL	Lateral collateral ligament
LED	Light emitting diode
MCL	Medial collateral ligament
ROM	Range of motion
SDK	Software development kit
UDP	User datagram protocol
UI	User Interface
VR	Virtual reality

1 Introduction

1.1 Motivation

Physical rehabilitation is the most common complementary therapeutic act in Portugal, corresponding to 72.1 % of all 23.7 million complementary therapeutic acts performed in 2015 (Instituto Nacional de Estatística, I.P., 2017). It is essential for the full recovery of injuries but it is often painful and requires the patient to perform repetitive acts that might bring discomfort or pain, reducing patient engagement. In fact, lack of patient interest and motivation are some of the main factors that might lead to unsuccessful physical rehabilitation (Maclean *et al*, 2000). Besides, conventional rehabilitation techniques depend on the constant supervision and instruction by a therapist and provide little to no reliable quantitative information about patient progress.

Therefore, in order to achieve more successful physical rehabilitation, it is important to make the act more motivating for the patients, making them more interested in pushing themselves to perform the required movements. Gathering qualitative and quantitative data that can be used by therapists and doctors to assess the evolution of the patient's state and adjust the exercises accordingly could also lead to an increase in efficiency.

Another aspect that might improve the quality of the physical rehabilitation acts is providing the patient with real time information regarding the quality of the performed movements, i.e. whether or not the movement is being performed correctly, reducing the need for therapist supervision. One way to achieve this and increase patient engagement at the same time is by providing biofeedback to the patients, particularly in the form of serious therapeutic games. Serious games are games in which the aim is not just to entertain the player but also to reach another goal. Their use in the context of therapy has been increasing recently due to the improvements in game technology and relatively cheap motion tracking technology. So far, studies have found that the use of therapeutic games to provide biofeedback to patients and help in the correction of movements increases patient motivation and satisfaction, although whether or not this approach is more effective than traditional physical therapy approaches is still not clear (Hsu *et al.*, 2010; Matallaoui *et al*., 2017).

Serious games created for therapeutic purposes must be adequate to the pathology being treated and a specific target population. Elbow dislocations are the most common paediatric joint dislocations (Rasool, 2004), particularly in children who practice sport (Hickey and Loebenberg, 2006; Halstead, 2017) and usually require the patient to go through motor rehabilitation in order to fully recover elbow flexion and extension. Therefore, it would be relevant to create a therapeutic game that addressed the movements required in physical rehabilitation of children who have suffered this injury.

Hence, in this work a biofeedback therapeutic game application will be created in order to improve the motivation therapeutic outcomes of patients who suffered an elbow dislocation, guiding them to a correct performance of the required movements by using motion tracking systems to allow interaction with a virtual environment. The application will acquire motion data along with other parameters that can be used to assess the patient's progress accurately, while reducing the need for therapist supervision.

1.2 Scope and Objectives

The main objective of this thesis is to use a motion capture system and a game engine to create an application for motor rehabilitation in which patient movement can be acquired and processed in real time, providing visual feedback of the performed movement, making the patient more aware of his own performance. A game was created in order to give the patient a clear goal and an immersive and entertaining environment besides the visual feedback, hopefully increasing patient motivation and satisfaction, generating better rehabilitation results. The created software also aims to register quantitative data from each therapy session, storing it and making it available for a later analysis, providing to the therapist accurate information about patient progression.

For this purpose, the motion capture system chosen was an inertial sensor system, F.A.B. System, and the chosen game engine was Unity. An age range (7 to 13 years old) and a pathology (elbow dislocation) were chosen. That way it was possible to focus the game creation on a game that would make the patient perform the motions usually prescribed for the physical rehabilitation of that specific pathology and to create an environment appealing to that specific set of the population. The validity of the application as an aid for rehabilitation and as a way to increase patient motivation is assessed by using the application on patients that meet the criteria established in terms of pathology and age.

1.3 State of the Art

Serious games are games with a purpose other than entertainment, aiming to increase user engagement in a certain task by using the appealing nature of games. One area where serious games have been highly developed over the past years is exercise, with the creation of exergames, i.e. games that aim to make people perform physical exercise in the context of a virtual environment. Soon there was an interest in applying these games to physical rehabilitation, and several studies on this approach have already covered topics from brain function rehabilitation to isometric muscle strengthening, exercising for elderly and balance training (Tanaka *et al.*, 2012). Howcroft *et al*. (2012) studied the use of active video games for rehabilitation in children with cerebral palsy using a marker-based visual tracking motion capture system,

concluding that the games encouraged motor learning and could be strategically used to achieve therapeutic goals. However, a number of studies report limited effects of exergaming. One of the main problems identified is the novelty effect: there is an initial abrupt increase in interest in the system when it is implemented, but once the novelty wears off, user retention might be low (Matallaoui *et al*., 2017). Hsu *et al.* (2010) analysed the use of Wii bowling in upper limb rehabilitation, concluding that the only significant improvement was an increase in patient motivation when compared to conventional exercises, with no increase in rehabilitation efficiency.

Exergames depend on biofeedback, the exposure of biological information to the user in real time, to interact with users. This concept becomes even more relevant when these games are applied to rehabilitation. Betker *et al*. (2006) found that biofeedback and videogame based exercises improved balance control in several neurological disorders and increased the patient's attention span and engagement. Merians *et al*. (2002) reported on the use of VR environments and haptic, visual and auditory biofeedback to improve hand function following a cerebrovascular accident (CVA), showing that the participants showed improved hand functions after a two-week training program. Patients in a study of the impact of biofeedback and virtual environments in control of a CVA patient's hemiparetic arm by Broeren *et al.* (2004) showed increased motivation along with better dexterity, grip force and motor control. Piron *et al.* (2005; 2007) conducted larger studies that showed biofeedback could aid in upper limb function rehabilitation in CVA patients, showing the same level of success as conventional therapy but with increased patient satisfaction. Doyle *et al*. (2011) reports an increase in movement precision when patients perform exercises interacting with a visual game that provided visual feedback, as opposed to performing the same exercise with limited or no feedback. A study on healthy adult postural stability by Fitzgerald *et al*. (2010), with a control group and a group receiving biofeedback from an exergaming system showed that despite the similar progressions in terms of postural stability, the biofeedback group showed more satisfaction and interest. These findings suggest that biofeedback and exergames may be used to improve exercise techniques in the context of rehabilitation.

One technique to provide visual biomechanical biofeedback is through motion tracking systems, providing the patient with feedback of their own movement and using it to interact with the game environment. Human motion tracking can be done through visual systems (marker based or markerless) or non-visual tracking systems. Currently, visual marker based systems are the most accurate (Zhou and Hu, 2008; Ceseracciu *et al*., 2014) but they are expensive, not portable, difficult to set up and can have several issues due to the use of markers, which can also be uncomfortable for the user (Zhou and Hu, 2008). Therefore, for exergames and therapeutic exergames, the most common approach is the use of visual markerless systems or non-visual tracking systems (based on inertial or magnetic sensors).

There are several commercially available options for motion tracking: the Kinect, launched by Microsoft, the Nintendo Wii and PlayStation's Move system. Chang *et al.*, (2011) created a Kinect-based rehabilitation application which was applied on two subjects with motor disabilities who reacted enthusiastically and performed more correct movements than when using conventional rehabilitation

methods. Webster and Celik (2014), however, criticize the qualitative nature of most Kinect-based elderly care and stroke rehabilitation results obtained so far. Wollersheim *et al*. (2010) studied the application of Wii games in exercise for the elderly and reports no increase in physical activity, possibly due to strategies developed by the users to advance in the game without performing the movements properly, since the Wii only detects movement through the controller, allowing the user to adopt motion strategies engaging only the hand. These issues would not be a problem using systems like Move or Kinect, since these include cameras for motion tracking. Either way, commercially available exergame systems are not properly designed for rehabilitation (Anderson *et al*., 2010).

However, this type of technology shows promising results which encourage the improvement of practical movement acquisition systems and therapeutic games, in order to achieve applications that effectively improve the efficiency of motor therapy, along with patient satisfaction.

1.4 Contributions

The main contributions of this thesis are:

- The creation of a Unity application which provides real time visual biofeedback and a specific goal for the patient using inertial sensors as a portable, comfortable motion capture system.
- Use of inertial sensors as a way to retrieve relevant quantitative patient data to assess progress.
- The construction of a database capable of supporting several users on which to store data from several physical therapy sessions and allow access to said data at any moment.
- Test of the created application on a control group in order to assess adequate game parameters depending on patient's characteristics;
- Test of the created application on a work group to evaluate the validity of the application as a therapeutic aid for elbow dislocations;

1.5 Structure and Organization

This thesis is structured in six chapters:

Chapter 1: the motivation, objectives and main contributes of this work are described, along with the state of the art in serious games, biofeedback and human motion tracking technologies.

Chapter 2: presents basic knowledge about serious games and biofeedback, along with existing applications in health.

Chapter 3: presents knowledge about human motion tracking technologies and game engines, along with the motives for the choices made in those aspects.

Chapter 4: describes some mathematical and computational concepts required for the association of the chosen motion tracking system and game engine, the animation of an avatar for patient interaction and other aspects of software creation. The aspects considered in the game's design are also explained.

Chapter 5: provides a description of the pathology which was selected as the target of the game's therapeutic aspect, along with the control and work groups in which the software was tested. This chapter contains the treated data from the use of the software and a discussion of its meaning.

Chapter 6: focuses on the conclusions that can be drawn from the work that has been done and presents suggestions on ways to improve this system in the future.

2 Serious Games and Biofeedback

2.1 Gamification, serious games and exergames

Salen and Zimmerman (2003) define game as "(...) a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome". The use of games or game elements in non-game contexts has been increasing in the past years, as approaches such as gamification and serious games gain relevance in areas from exercise to health or education, ecological behaviour or work productivity (Matallaoui *et al.*, 2017; Groh, 2012). Groh (2012) makes a distinction between serious games and gamification, clarifying that while a serious game is a full-fledged game created for a non-entertainment purpose, gamification refers only to the use of typical game elements, such as leaderboards and achievement badges, in new contexts. While in both cases the goal is to improve user engagement in a task, in the case of gamification the task remains unchanged and the aim is to improve user motivation to complete it through feedback, contrasting with the serious games approach, where besides an increase in motivation, the aim is to provide a new way to perform the task. In this case, feedback is provided not just through rewards but also through the way the task is performed. Marsh (2011) proposes a formal definition for serious games as follows: "Serious games are digital games, simulations, virtual environments and mixed reality/media that provide opportunities to engage in activities through responsive narrative/story, gameplay or encounters to inform, influence, for well-being, and/or experience to convey meaning. The quality or success of serious games is characterized by the degree to which their purpose has been fulfilled. Serious games are identified along a continuum from games for purpose at one end, through to experiential environments with minimal or no gaming characteristics for experience at the other end."

One field that has seen serious games prominently implemented is health and exercise, through the creation of exergames, defined by Tanaka *et al* (2012) as digital games that promote physical activity through the use of platforms designed to capture body movement or reactions, turning physical exercise into a more captivating experience. This concept dates back to the 1980's, when Nintendo first launched PowerPad, a mat that recorded footsteps, and PowerGlove (Sinclair *et al.*, 2007) but the first truly successful application was *Dance Dance Revolution* by Konami in 1998 (Tanaka *et al*., 2012). In 2002, the development of more affordable and reliable motion sensors and game technology led to the creation of Sony EyeToy, an attachment for Sony PlayStation consisting in a camera able to monitor the player's kinematic information. The image processing was done in two dimensions, making this a very limited system (Whittinghill, 2014; Tanaka *et al., 2012*). The Nintendo Wii was released in 2006 and improved in 2010, relying now on a controller composed of a triaxial accelerometer and a triaxial gyro sensor, capable of detecting changes in the user's hand's movement speed, acceleration and direction, along with the hand's pose, making this a very robust system (Agmon *et al.*, 2011; Tanaka *et al.*, 2012). Nintendo Wii has

been advertised as a way to make children more physically active and has achieved considerable market success, proving the consumer's interest in exergaming. There are, however, two more motion capture based exergaming commercial systems available, with relatively less success: Sony PlayStation Move and Microsoft Kinect.

As entertainment-driven exergames evolved, the efficiency and efficacy of these systems and these types of games on rehabilitation began being studied, using mostly the Kinect or Wii and Wii Fit systems, along with a new Nintendo peripheral released in 2007, the Wii Balance Board (Tanaka *et al*., 2012).

However, not all serious games can be considered therapeutic. Mader *et al*. (2012) presents a model to analyse therapeutic games through the relations between its three main aspects: the player, the game and the therapy.

- Therapy: Through the design process of a therapeutic game, it is not possible to assess the actual benefits of the therapeutic function. However, it is possible to assess previous scientific evidence of similar game features having therapeutic effects and define parameters to be evaluated post-development, namely expected short and long-term therapeutic value and scientific proof of efficiency, i.e. demonstration and discussion of the effects.
- Player: Game design depends on understanding the target player in terms of relevant parameters, such as age, gender, particular conditions and abilities.
- Game: After considering the player and the therapy, the game must be designed considering several aspects, namely the input and output systems, the goals of the game (one must consider if the goals exist and are appealing; analyse if the goals are meant to be short, mid or long-term), the means of player feedback, the meaning of the game's score, the difficulty level and how it can be adjusted, in-game variability, usability, expected positive side-effects and previously reported serious uses for the game or games with similar features.

After considering these aspects of therapeutic game design, it is necessary to validate the design coherency by evaluating the relations between the mentioned aspects.

- Player/Therapy: Does the player have the condition that the therapy can help improve?
- Game/Therapy: Which game features are therapeutic or motivational? Where and how will the game be used?
- Player/Game: Was the game designed in a way that makes it safe and enjoyable for the patient? Is the patient able to play it?

This model can be used to give a general idea of whether a game can be considered therapeutic or not and can be a guideline in the creation of therapeutic games.

2.2 Biofeedback

Biofeedback can be defined as the technique of exposing biological information to patients in real time that would otherwise be unknown to them, reason why it is sometimes referred to as extrinsic feedback, as opposed to intrinsic or sensory feedback, which is the information accessible to the patient through intrinsic sensory receptors (Giggins *et al.*, 2013). Biofeedback usually involves measurements of biomedical variables using monitorization instruments. The data obtained is then transmitted to the patient in real time through either direct feedback, a direct exposition of a variable's value (such as heart rate), or transformed feedback, where the measured values are used to control an auditory, visual or tactile signal (Giggins *et al.*, 2013; McKee, 2008).

Sweetser and Wyeth (2005) defined a model through which a game should be designed to ensure user satisfaction, called *Game Flow*. According to this model, users must feel like they can control their actions in the game to create immersion. In the context of physical rehabilitation, the game must accurately measure the user's movement and utilize that information directly in the way the game is controlled. Providing patients and clinicians biofeedback may contribute to a better control of the monitored physical process, making it possible to increase accuracy during functional tasks while providing the patient with an opportunity to be more engaged in their own rehabilitation process and reducing the need for monitorization by healthcare professionals (Giggins *et al.*, 2013; McKee, 2008). The more commonly measured biomedical variables can be divided into two categories: biomechanical (movement, posture control and forces) or physiological (neuromuscular, cardiovascular or respiratory biofeedback) immersive (Giggins *et al.*, 2013). The focus of this work is on biomechanical biofeedback, which involves measurements of movement, postural control or forces produced by the body. These variables can be measured through several devices, like force plates, electrogoniometers, camera-based systems and inertial sensors, among others. In this thesis, an inertial sensor system was used to provide postural and movement biofeedback, as described in more detail further ahead.

Laver *et al*. (2011) defines Virtual Reality (VR) as an advanced form of human-machine interface which allows the user to be immersed in a computer generated responsive environment. Recent developments in gaming technologies and VR allowed for the creation of therapeutic exergames in which the patient performs rehabilitation exercises safely in a more immersive, interactive and enjoyable environment, receiving haptic, visual or auditory feedback (Giggins *et al.*, 2013).

The fast-paced improvement in technology and the growing interest in VR applications make this type of approach to rehabilitation a growing sector, particularly with the opportunity to create home-based VR therapeutic exergames that would allow patients to complement therapy sessions at home without needing supervision. Furthermore, the possibility to store data from the exergames to be analysed by therapists and doctors promises to provide relevant quantitative data to assess patient progression and to create more personalized exercise prescriptions.

2.3 Applications

Jintronix Inc. (Montreal, Canada) created a VR Kinect-based upper limb rehabilitation system at a relatively low price, composed by three unilateral and two bilateral activities, each divided in ten difficulty levels (Archambault *et al*., 2015). The system can be used in either a clinical environment or at the patient's home and promotes a series of motions such as reaching, transporting and dropping virtual objects by moving the upper limb through a defined trajectory. Parameters like the number and position of the objects, required movement speed and the presence or absence of obstacles can be adjusted in order to personalize the game to the patient's specific needs and abilities in terms of movement range, speed and precision (Norouzi-Gheidari *et al.,* 2013).

Figure 1. Examples of Jintronix software (http://www.jintronix.com/).

BTS Nirvana[1] is a virtual rehabilitative environment developed by BTS Bioengineering Corp (Brooklyn, NY, USA) for the rehabilitation of patients affected by neuromotor and cognitive disorders (De Luca *et al*., 2017). The system is composed by markerless infrared sensors, the BTS Nirvana software for motion analysis, a Webcam and a workstation touchscreen (De Luca *et al*., 2017; Carlomagno *et al*., 2011). The system is connected to a projector or a screen and the patient chooses exercises and scenarios to interact with, receiving audio and visual stimuli. The difficulty level, speed and range of motion of each exercise can be adapted in order to fit the patient's needs. This system is recommended for patients with movement and attentive dysfunctions, particularly caused by stroke (De Luca *et al*., 2017). The patient's progress is measured during the exercise and the system generates significant indexes which can be accessed in real time or through reports. The system can be managed through several operating systems and devices, such as PC, tablets and smartphones.

[1] http://www.btsbioengineering.com/products/nirvana/ (2017) [accessed 2018 March 17].

Figure 2. Examples of BTS Nirvana environments (http://www.btsbioengineering.com/products/nirvana/).

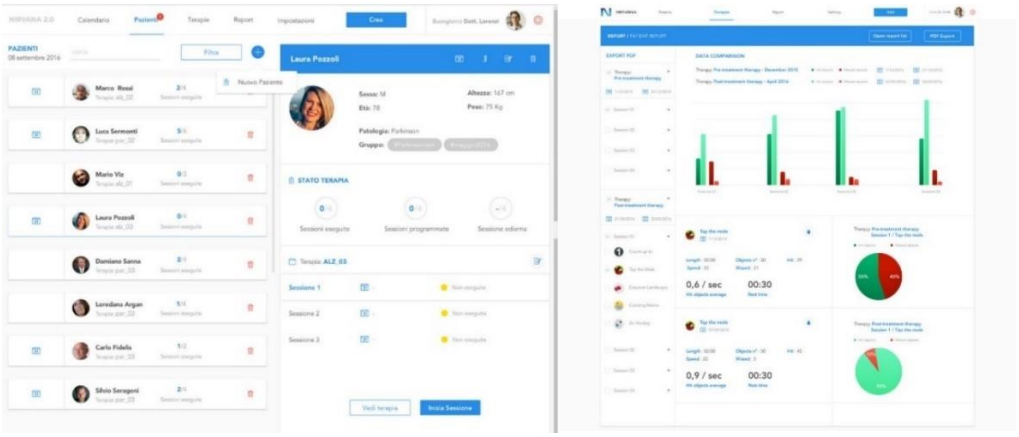

Figure 3. Examples of BTS Nirvana patient reports (http://www.btsbioengineering.com/products/nirvana/).

Reflexion Health Inc. (San Diego, CA, USA) created the tele-rehabilitation application called Virtual Exercise Rehabilitation Assistant (VERA)[2]. This application guides the patient through a series of exercises using the Kinect to track the movement of 20 joints (Komatireddy *et al*., 2014). This application can be used at home without a therapist's supervision and provides corrective feedback in real-time, while storing information regarding the number of exercises performed and their quality.[2]

[2] http://reflexionhealth.com/ (2017) [accessed 2018 March 17]

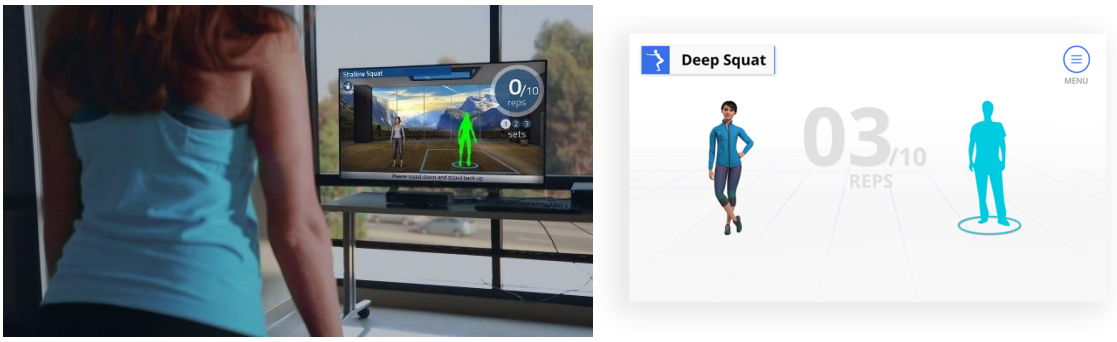

Figure 4. Example of VERA software (http://reflexionhealth.com/).

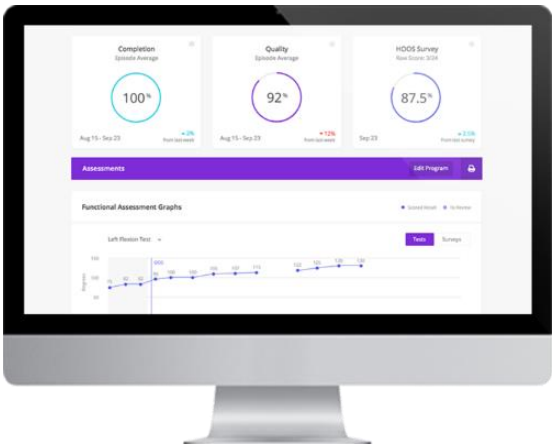

Figure 5. Example of VERA report (http://reflexionhealth.com/).

KineLabs (Kowloon, Hong Kong, China) developed a Kinect-based application that consists of three games meant to improve trunk balance and upper and lower limb mobility, aimed at elderly stroke patients with the goal of improving or maintaining their independence. This application can be used at home, in hospitals or in assisted living facilities and allows for storage of data related to joint angles and motion control quality to be analysed by therapists (Tong, 2014; Tong *et al.*, 2012). This system is free for the public, being that the only costs are the costs of the Kinect sensor and PC or laptop (Tong *et al.*, 2012).

Toyra[3] (Indra Sistemas, S.A., Madrid, Spain) is a real-time virtual reality application based on a motion capture system with the aim of improving upper limb rehabilitation while providing health professionals objective measurements, being tested in several research centres. Another aim of Toyra is to increase patient motivation with the goal of increasing effectiveness of therapy and reducing average hospital stay times. Toyra offers several options in terms of motion capture systems for different cases: wired or wireless inertial sensors, the Kinect sensor, Sony BUZZ wireless remote control and an adapted motorised wheelchair joystick. In terms of software, Toyra has two versions of their interactive therapy stations, one for hospital environment and one for home application.

[3] http://www.toyra.org/ [accessed 2018 March 17]

Figure 6. Example of Toyra software (http://www.toyra.org/).

SWORD Health[4] (Porto, Portugal) created SWORD Phoenix, a virtual therapist technology which uses AI and inertial sensor based motion tracking to analyse patient performance and provide feedback. This system aims to maximize engagement and clinical outcomes, while retrieving objective data for later analysis and rehabilitation adaptation. The goal is to allow patients to perform exercises at home and reduce the need for immediate supervision and provide data that makes it possible to define more patient-specific therapies.

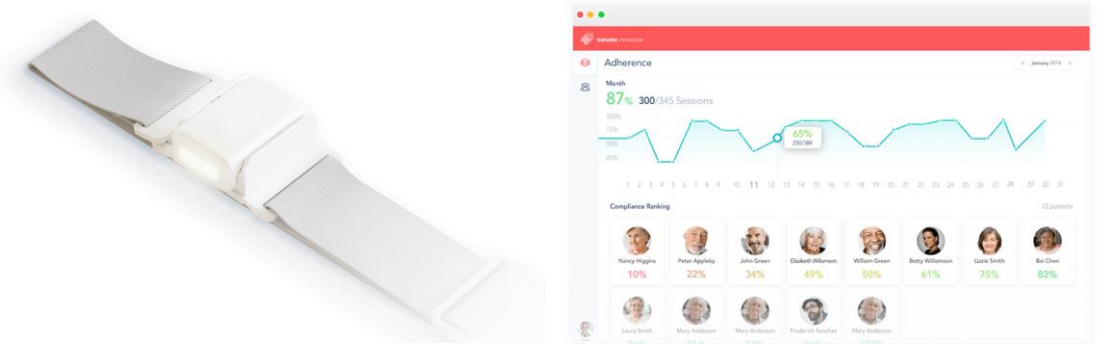

Figure 7. SWORD Phoenix sensors (left) and software (right) (https://www.swordhealth.com/).

The previously mentioned systems show that there is a growing interest in the commercial use for biofeedback and serious games in physical rehabilitation. The expected increased patient motivation, along with the quantitative data that would allow for therapies that are patient-specific and more efficient create a great interest in this type of applications. There are still a lot of areas to explore, namely regarding serious games, which are only present in two of these systems, the type of movement acquisition, since most of these systems (including the two that included serious games) rely on the Kinect, and the type of pathologies and population for which the systems were created.

[4] https://www.swordhealth.com/ [accessed 2018 May 02]

3 Biofeedback Interfaces

3.1 Movement acquisition systems

Human movement acquisition systems were created in order to generate data that represents the changes in posture of the human body or its segments in real time, using motion tracking technology (Zhou and Hu, 2008). A review study by Zhou and Hu (2008) covers the main movement acquisition methods, dividing them in three main categories: visual-tracking (with or without markers), non-visual tracking and robot-aided tracking. Robot-aided tracking systems will not be covered in this thesis since their main applications involve stimulation of muscles, movement assistance or weight support (Roy *et al*., 2009; Hesse and Uhlenbrock, 2000; Perry *et al*, 2007) which are not in the scope of this work.

3.1.1 Visual tracking

Visual tracking systems usually make use of several optical sensors and may require the user to wear markers placed on specific places on the body (marker-based) or not (marker-free). Marker-based visual tracking systems are considered the standard in human motion acquisition and analysis due to the high position estimation accuracy, having errors below 1 mm (Zhou and Hu, 2008; Ceseracciu *et al*., 2014). Moleslund *et* al. (2006) did a survey about advances in computer vision for human motion tracking, concluding that visual systems present the more attractive solution. In 1973, Johansson performed a study that became a milestone in human motion tracking, in which he attached small reflective markers to the joints of human subjects, who were then monitored throughout several trajectories. The basic theory behind this study is still present in current systems. These systems can be used to estimate anatomical segment orientations and angles, along with joint centres and have become powerful tools for orthopaedic diagnosis and for improvement of athletic performance, among other applications (Davis *et al*., 1991; Charlton *et al*., 2004). The markers used in these systems can be considered passive, active or hybrid: passive if they do not generate light and merely reflect incident light, and active if they produce light to be detected by an optical system (Zhou and Hu, 2008). The position of each marker is estimated by combining the information from the 2-D images captured by each camera (Menache, 2000; Zhou and Hu, 2008).

However, despite the accuracy of these systems and their use in several types of applications, they present some disadvantages which prevent them from being used more broadly. Besides the high cost of the systems, they are complex to setup, require a long time of patient preparation and require specific environments in order for the data to have high quality, making them unsuitable to track patient motion in day-to-day situations or in even slightly chaotic environments (Best and Begg, 2006; Bonnechère *et al.*,

2014). The markers can be a source of discomfort and movement restriction for the patient (Corazza *et al*., 2006) and their use requires the identification of bony landmarks, meaning that an incorrect identification can lead to errors calculating rotations occurring in a single plane (Della Croce *et al., 2005*). Marker occlusion and errors caused by light sources may also create error in the estimation of marker position (Zhou and Hu, 2008). It is also necessary to consider that there might be artefacts caused by soft tissues, i.e. marker dislocation due to skin deformation, leading to relative movement between markers or even causing them to fall (Fuller *et al.,* 1997; Leardini *et al.*, 2005).

Marker-less visual tracking systems are the least invasive form of motion tracking since they only require optical sensors to estimate human body movement and were created with the aim of avoiding the flaws of the marker-based systems. These techniques, however, require intensive computation to be able to track objects in 3-D with relatively small errors and it is worth noting that cameras used in such systems are not conventional cameras, since those usually have a sampling rate of no more than 60 frames per second (Zhou and Hu, 2008). The fact that there is no need for markers or sensors makes these systems very appealing for motion tracking in day-to-day activities (Moeslund *et al*., 2006). However, these systems tend not to be very robust at estimating joint orientation from joint position, leading to an inaccurate representation of motion.

Both marker-less and marker-based visual systems can have problems due to occlusion, either of the markers or of the body segments themselves.

3.1.2 Non-visual tracking

The main advantage of non-visual tracking systems is the fact that occlusion will not be an issue, since there is no dependence on optical sensors (Zhou and Hu, 2008). The most relevant of these systems are based on either inertial or magnetic sensors.

Inertial sensor systems, composed of accelerometers and/or gyroscopes, have been used for navigation and augmented reality modelling and represent an efficient and relatively cheap way to track full body movement (Zhou and Hu, 2008; Hansson *et al*., 2001). These sensors, particularly wireless ones, are easier to wear than markers, although they may still be cumbersome to wear in day-to-day life (Li and Buckle, 1999). Despite the fact that these systems are usually very sensitive and have a large capture area, there could be issues with offset fluctuation, signal noise and integration drift, leading to a gradual increase in position and orientation errors (Zhou and Hu, 2008). Besides, it is not possible to estimate the position of sensors with respect to each other (Schepers *et al*., 2010).

Magnetic sensor systems have been used to track movement in virtual reality due to their size, high sampling rate and lack of problems with occlusion. However, there are some latency issues, along with jitter when around ferrous materials or electronic devices and measurement noise (Zhou and Hu, 2008).

Attempts at solving these problems have been and are being made, mainly through the use of Kalman filters (Zhou and Hu, 2008).

A combination of gyroscopes, accelerometers and magnetometers can be used to reduce drift by continuous correction of orientation (Roetendberg *et al*., 2013). Besides, this combination allows for estimation of relative body positions (Schepers *et al*., 2009).

Other types of sensors, such as mechanical or acoustic sensors, can be used for non-visual motion tracking. However, their use is not as common due to issues of lack of comfort and inability to account for all joint rotations in the case of mechanical sensors (Menache, 2000), or the need for large devices, the need for a line of sight between emitters and receivers and latency in acoustic systems (Zhou and Hu, 2008). For these reasons, these types of systems were not relevant for this thesis.

3.1.3 Comparison

The advantages and disadvantages of the main movement acquisition systems are summarized in Table 1.

Table 1. Summary of characteristics of the main motion tracking systems.

System		Accuracy	Portability	Computation	Cost	Comfort	Drawbacks
Non-visual tracking	Inertial	High	High	Efficient	Medium	Medium	Drifts
	Magnetic	Medium	High	Efficient	Medium	Medium	Ferromagnetic materials
Visual tracking	Marker-based	High	Low	Inefficient	High	Low	Occlusion, marker movement
	Marker-free	Medium	High	Inefficient	Low	High	Occlusion

From this table, it is possible to conclude that, for the purposes of applications such as the one being created in this thesis, the more adequate motion tracking systems would be either marker-free visual tracking systems or non-visual tracking systems, due to their high portability. Considering the necessity for efficient computation, cost-efficiency, portability and high accuracy in order to obtain relevant motion data for medical analysis, the best type of motion tracking system would be a non-visual tracking system, consisting of a combination of magnetic and inertial sensors.

3.1.4 Commercially available motion tracking systems

Commercial exergaming systems depend on inexpensive, comfortable movement acquisition, and most current therapeutic applications of exergaming use these same systems, which are either inertial sensors (non-visual tracking), cameras (marker-free visual tracking) or a combination of both.

The most popular commercially available non-visual tracking system is the Nintendo Wii MotionPlus, consisting of a triaxial accelerometer operating at 100 Hz (ADXL330) and gyro sensors that allow for a more accurate capture of complex motion, along with an IR camera (Tanaka *et al*., 2012; Lee, 2008). The Nintendo Wii system also includes the Wii sensor bar, in order to obtain information on the 3-D position of the controller (Tanaka *et al*., 2012; Lee, 2008). The sensor bar has two infra-red (IR) Light Emitting Diodes (LED) clusters, one on each side, which are detected by the controller's IR camera. However, position estimation is difficult due to the limitation to the controller position and posture that it requires (Tanaka *et al*., 2012).

This system has a relatively high temporal resolution for hand motion detection and has opened several SDK to the public for development of applications using this controller (Tanaka *et al*., 2012; Lee, 2008). Some drawbacks of this system in the context of this work are that since the user is holding the remote, it is possible to perform wrist movements that would replace movements of other joints, the fact that the applications created are restricted to upper limbs, and the limited amount of motion data that can be obtained. It is not possible, for example, to obtain accurate values of angles between body segments, and body posture information is not accurate.

Figure 8. Nintendo Wii MotionPlus controller (left) and sensor bar (right) (https://store.nintendo.com/)

Another inertial sensor based commercial system is the PlayStation Move Motion Controller. This controller contains a triaxial accelerometer, a triaxial gyro sensor and a geomagnetic sensor, allowing for rotation tracking and cumulative error correction (Tanaka *et al*., 2012). This system also includes a high-speed camera, The PlayStation Move Camera, so as to estimate positions. This camera detects an illuminated sphere attached to the controller and determines the distance between the two by analysing the sphere size (Tanaka *et al*., 2012). The 3-D coordinates are calculated with high resolution based sub-pixel image processing. While recognition of the controller's position and orientation is more robust on

the PlayStation Move than the Wii, upper body motion estimation is worse and there are no available SDK (Tanaka *et al*., 2012).

Since hand motion information is extracted similarly, some of the Nintendo Wii's drawbacks apply here too: movements can be done with the wrist, avoiding movement of other joints, and the applications created with this system are limited to upper limb motion. Considering that this system includes a camera, it would be easier to obtain more motion data, such as angles and posture information.

Figure 9. Sony PlayStation Move Motion Controller (left) and Move Camera (right) (https://www.playstation.com/)

Another popular commercially available motion tracking system is the Kinect One, launched by Microsoft. This sensor contains a colour camera and a depth sensor, which includes an IR emitter and an IR sensor. The depth image is obtained through a time-of-flight method based on the time IR light takes from being emitted until being sensed by the IR sensor after being reflected off the target (Skalski and Machura, 2015). This system allows for the prediction of the position of several human joints from a single depth image. Compared to other commercially available systems, position and orientation estimation with the Kinect is accurate and could be used in a clinical setting (Kurillo *et al*., 2013a; Kurillo *et al*., 2013b). Besides, most of the drawbacks from the other systems are not an issue in this case, and SDK's are available (Tanaka *et al*., 2012). However, this system is subject to occlusion and some studies have found that proportional biases in some parameters may reduce the accuracy of this system (Clark *et al*., 2012).

Figure 10. Microsoft Kinect One (https://www.microsoft.com/)

3.1.5 The F.A.B.™ System

The F.A.B. ™ System[5] (Functional Assessment of Biomechanics ™) developed by Biosyn Systems Inc (Surrey, BC, Canada) is an off-the-shelf portable full body recording system consisting of 17 inertial motion sensors, each consisting of an orthogonal triad of accelerometers, magnetometers and gyroscopes which allow real time detection of angular displacements, the F.A.B Recorder software and a Real Time Receiver, which must be connected to a computer through an USB port in order to use the software. Each sensor measures its acceleration and orientation in 3-D space. The accuracy of motion capture is typically better than 1° for vertical plane angles and 3° for horizontal plane angles. The accelerometers and gyroscopes are sampled at 100 Hz while the magnetometers are sampled at 25 Hz, limiting the temporal resolution to 25 Hz. The data is transmitted to the receiver at a packet rate of 25 Hz (Mack *et al*., 2010). This system is rated to collect acceleration up to 5 g and angular velocity up to 1200 degrees per second (Mack *et al*., 2010).

However, this system presents some drawbacks, namely the fact that motion capture accuracy might vary with motion speed and that the sensors might shift in position, leading to inaccurate movement representations. Besides, as all non-visual tracking systems, this system may lose accuracy due to drift or poor transmission, among other issues (Rahimi *et al*., 2011). This system does not provide information regarding the position of the sensors.

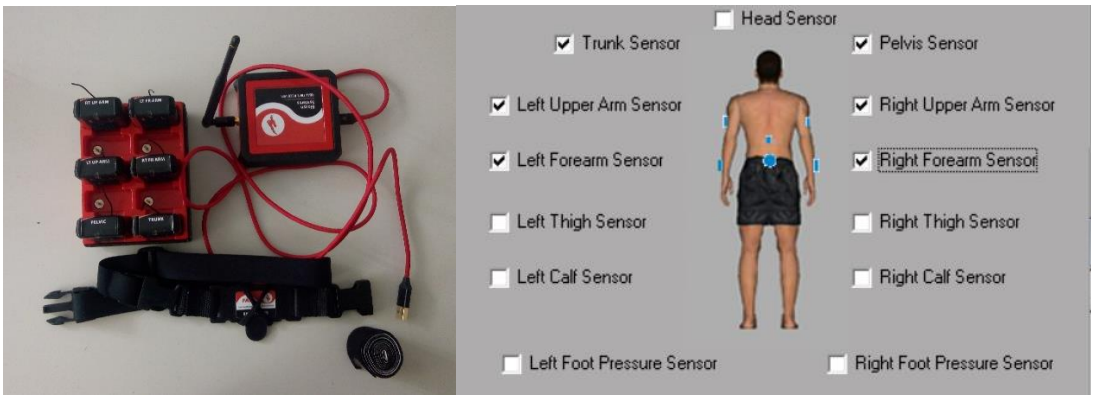

Figure 11. Components of the F.A.B. ™ System (left), representation of the position of the sensors on the body (right)

Each sensor in the F.A.B. ™ System is identified with the name of the body segment it represents and must be placed on the correct segment. In this context, only the pelvis, trunk, right and left arm and right and left forearm sensors were included. The sensors have parent/child relationships with each other, being

[5] http://www.biosynsystems.net/f-a-b-system/ [accessed 2018 May 4]

that no child sensor can be used without its parent, as illustrated in Figure 12. The rotations of each sensor are relative to its parent, i.e. at the end of calibration, the three Tait-Bryan angles of a sensor will be set at zero regardless of their position and orientation relative to each other. The only angle that is not set to zero is the pelvis sensor's yaw, corresponding to the vertical axis, which determines the orientation of the body.

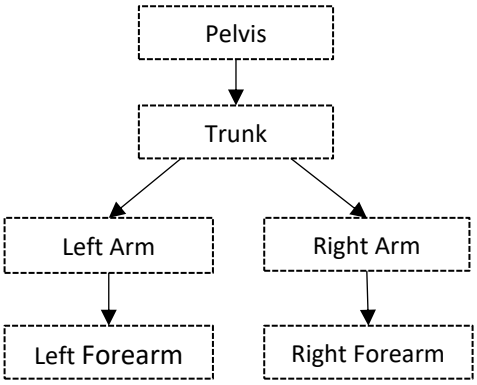

Figure 12. Representation of the parent/child relationships between inertial sensors in the F.A.B. ™ System. Each arrow points from the parent to the child.

Compared to the Kinect, the F.A.B. ™ System allows for a more accurate estimation of joint orientation and body posture and is not subject to occlusion, making it more fit for a clinical application. Besides, when compared to the Wii or Move systems, F.A.B. ™ system requires several sensors tracking different body parts, meaning that applications can be created for more than just upper limbs and that accuracy is much higher when estimating body posture. None of the F.A.B. ™ sensors have to be hand held, making the system more comfortable and preventing substitution movements with the wrists.

3.3 Game Engines

Game Engines are middlewares created to facilitate the development of applications by offering an environment that improves efficiency through the gathering of several types of engines necessary in a game's creation. Hardware abstraction is also done by the game engine, allowing the application to be run in different platforms (Antonín, 2017; Macedo *et al.*, 2015).

A game engine can be divided in several components responsible for different functions in the final application:

- Rendering: refers to the exhibition of a 3-D object in the application through communication with a graphics API (Application Programming Interface) like OpenGL or DirectX, avoiding the need to access them directly. Rendering takes up most frame update time, up to 90% (Antonín, 2017).

- Animation: responsible for animating the game objects, i.e. the change in predefined parameters through time, either changes in position/rotation, shape distortion or rigging (Antonín, 2017). Rigging is the creation of a skeleton-like structure within an object's mesh. Segments of the skeleton are connected to parts of the mesh, allowing the object to be animated through several methods, including motion capture.
- Scripting: responsible for the description of game objects' behavior through the attachment of scripts describing how the object will interact with the player or other objects.
- Audio: most games count on music and sounds to create a more realistic and interesting atmosphere. It becomes necessary to have an engine to play and stop sound according to game events. Advanced engines allow developers to simulate echo in 3-D environments or play spatial sound according to the player's position (Antonín, 2017).
- Physics: a physics engine, like Havok or PhysX, is necessary to ensure that a game object's behavior corresponds to the behavior of a similar object in reality. Each object has a collider, which corresponds to its borders and allows interaction with other objects, and a physical material, which determines surface and mass properties and might simulate soft bodies whose borders change according to outer forces. The physics engine calculates movement vectors and collisions between objects based on their properties.
- Artificial Intelligence (AI): most games involve non player characters and therefore require the existence of an AI, responsible for controlling the movement and behavior of these characters. The game engine does not directly program the AI but it offers a set of tools that allow the developers to create one with the desired behaviors (Antonín, 2017).

When an application is created in a game engine, each game object must be updated and/or renderized in each frame, through a loop called Game Loop. Each game loop corresponds to one frame and comprises three steps: data entry, game object update and game object visualization (Macedo *et al.*, 2015). During the data entry phase, the state of input device variables is updated, registering the user's input. In the following phase, all game objects are iterated and their respective update methods are called in order to obtain the full status of the application in a certain frame. The last step refers to the action of the graphics engine, rendering the objects in the scene (Macedo *et al.*, 2015).

Currently, the three most popular game engines are Unity, which was the most used game engine in 2016 (Antonín, 2017) followed by Unreal Engine and Cry Engine. These are the game engines compared in this thesis.

3.3.1 Cry Engine

The Cry Engine was created by Crytek (Frankfurt, Germany), focused on the creation of PC and console applications. It uses Lua scripts and C++ as programming languages, and it contains state of the art lighting,

rendering, animation and design engines, supporting highly realistic simulations (Antonín, 2017; Trenholme and Smith, 2008). It has high scalability and it has been used by big game companies like Ubisoft (Rennes, France) (Macedo *et al.*, 2015). However, it has a steep learning curve, supports the smallest number of platforms of the three engines considered and does not offer complete documentation, making it the least suitable for beginner developers. Besides, the online developed community is small and it has high hardware demands (Trenholme and Smith, 2008).

3.3.2 Unreal Engine 4

Unreal Engine was developed by Epic Games (Cary, NC, USA) and it is considered the lead in terms of realistic sceneries. It supports development in all the main platforms (Windows, MacOS, Linux, XBox, Playstation, Steam OS, Android, iOS, etc) although it is not the most suitable engine for mobile development (Macedo *et al.*, 2015; Petridis *et al*., 2010). In terms of programming languages, Unreal offers the opportunity to use C++ or the Blueprint system, a robust visual scripting language. While it is easy to learn, it has a high abstraction level, leading to a high computational overhead, making this language up to ten times slower than C++ (Antonín, 2017; Macedo *et al.*, 2015). Writing the scripts in C++ can be complicated. While the framework is open source, giving the developers a bigger control over the system, it is complex and compilation times are high (Antonín, 2017). Unreal provides documentation and there is an online developer community, but the documentation is not as complete and the community is not as active as Unity's.

3.3.3 Unity 3D Engine

Unity is a 2-D, 3-D, VR and AR (Augmented Reality) development platform created in Denmark and was released for the first time at the Apple conference in 2005, targeting OS X development. Nowadays it supports development for 27 platforms (Antonín, 2017; Craighead *et al*., 2007). As programming languages, it uses C#, JavaScript and Boo, allowing for fast efficient scripting (Antonín, 2017).

Unity's main advantage, apart from the extensive multiplatform distribution, is its learning curve since it is easy to use, making game development quick. The software runs even on computers that are not very robust. Besides providing full documentation, Unity relies on an active online developer community where several developers exchange information (Antonín, 2017; Craighead *et al*., 2007) and an Asset Store where several developers offer or sell resources.

Graphics-wise, Unity is inferior to Unreal Engine or Cry Engine, being the engine with least scalability, more suitable for the creation of small or mobile projects (Antonín, 2017; Macedo *et al.*, 2015).

3.3.4 Comparison

An overview of the most important aspects of the previously mentioned game engines is presented on Table 2.

Table 2 - Comparison between the main current game engines

	Unity	**Unreal Engine**	**Cry Engine**
Programming Language	C#, JavaScript, Boo	C++, Blueprint	Lua Scripting, C++
Learning Curve	Easy	Medium	High
Documentation	Complete	Incomplete	Incomplete
Online Community	Big, Active	Medium	Small
Scalability	Medium	High	High
Supported Platforms	All relevant platforms	All relevant platform	Does not support development for MacOS

Considering the requirements of this project, the most relevant factors when choosing a game engine were ease of use, documentation, online community and programming language, which is why Unity was the chosen development platform. Since the project is not exceedingly demanding graphically, Unity's inferior performance in this aspect is not an issue.

3.3 F.A.B. System and Unity

Using inertial sensors to provide biofeedback and acquire data in a Unity created game environment allowed for the relatively fast development of a visually appealing, easy to use application that interacts with the user, providing information about the performed movements. The fact that the application was developed in this game engine means that it can be easily deployed for several operating systems and that its creation was made significantly easier by using items from the asset store. The inertial sensor system is portable, comfortable and easy to apply and has a high level of accuracy, being adequate for an application to be used in a therapeutic context, so as to provide accurate relevant information to healthcare professionals.

4 Software Development

In terms of software, the objective of this thesis was to develop an application that allowed several patients to create accounts which would be stored in a database, select a therapeutic game that is adequate to the therapy of their pathology and the injured articulation, play the game receiving visual feedback in real time by animating an avatar in real time through the use of inertial sensors, allowing for immediate visual feedback and then store and view information regarding each session in the database. This process involved defining a structure for the application, establishing a connection between the sensors and Unity, animating an avatar in real time with information provided by the sensors, designing a game that could have therapeutic value and that was adequate to the population in study and managing the database information in order to store it and visualize it in the application correctly. The structure of the application is presented in Figure 13. Figure 14 represents the steps from acquiring user motion with the inertial sensor system until the movement information reaches the game environment, providing visual feedback.

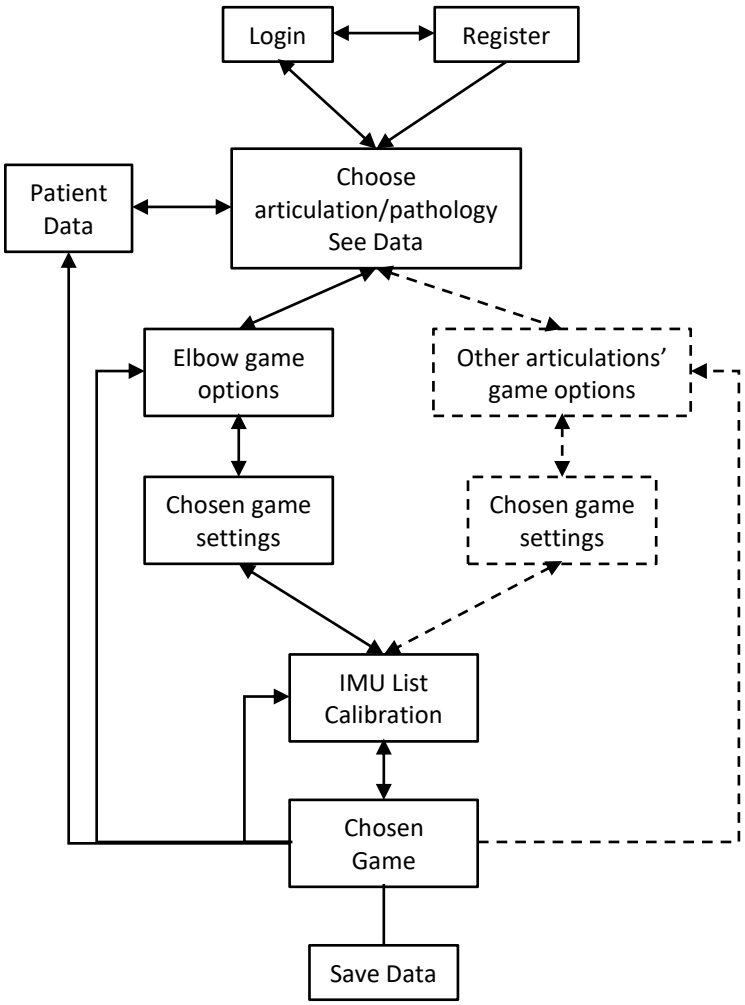

Figure 13. Schematic representation of the structure of the application. Dashed arrows and boxes represent parts of the software that were not created, although the software is ready to access them. Two way arrows mean that it is possible to go back and forth between two parts.

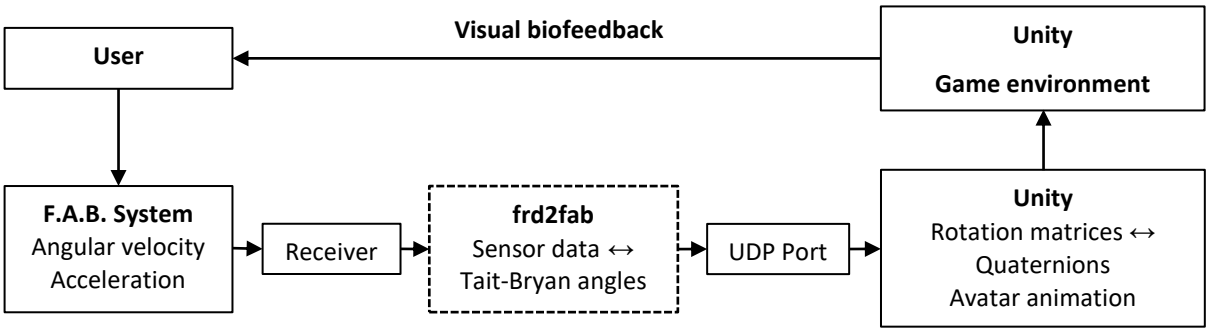

Figure 14. Schematic representation of the steps taken to achieve biofeedback.

4.1 Rotation Matrices, Euler Angles and Quaternions

In order to animate an avatar using the patient's motions, it was necessary to manipulate the orientation of a rigid body in relation to another rigid body; thus some algebraic concepts are needed in order to perform the mathematical computations. Orientations can be represented on the 3-D vector space in several ways, being that the ones relevant for this thesis are rotation matrices, Euler angles and quaternions. In this case, it is worth noting that the inertial sensor motion capture system in use transmits sensor orientation as Euler angles and the chosen game engine, Unity, represents game object rotations as normalized quaternions, i.e. unit quaternions. Although inertial sensors measure angular velocity and acceleration, it is worth noting that the relation between angular velocity and Euler angles, described by Diebel (2006) was not approached in this thesis since the data received from the sensor system on the UDP port was already presented as Euler angles.

4.1.1 Rotation Matrices and Euler Angles

A rotation matrix is used to transform vectors and coordinates from one coordinate frame to another and vice versa, i.e. it is a matrix that, upon multiplication with a vector rotates the vector while preserving its length. Considering $SO(3)$ the special orthogonal group of all 3×3 rotation matrice, if $R \, \epsilon \, SO(3)$, then

$$\det \boldsymbol{R} = \pm 1 \tag{1}$$

$$\boldsymbol{R}^{-1} = \boldsymbol{R}^{T} \tag{2}$$

Rotation matrices for which $\det \boldsymbol{R} = -1$ are called improper and are not rigid-body transformations, therefore in this context only proper rotations $(\det \boldsymbol{R} = 1)$ are considered. The elements of any matrix can be referenced as:

$$\boldsymbol{M} = [\boldsymbol{m}_1 \quad \boldsymbol{m}_2 \quad \boldsymbol{m}_3] = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_3 \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \tag{3}$$

Euler's theorem states that "any two independent orthonormal coordinate frames may be related by a minimum sequence of rotations (less than four) about coordinate axes, where no two successive rotations may be about the same axis" (Kuipers, 2000). Each rotation about a single coordinate axis is a coordinate rotation. Consider the 3-D reference frame presented in Figure 15, which corresponds to both Unity's and F.A.B. Recorder's reference frame. Rotations of ϕ, θ and ψ about the x, y and z axis, respectively, can be given by:

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\ \phi & -sin\ \phi \\ 0 & sin\ \phi & cos\ \phi \end{bmatrix} \tag{4}$$

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & cos\ \theta \end{bmatrix} \quad (5)$$

$$\boldsymbol{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & cos\ \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6)$$

The angles ϕ, θ and ψ are called the Euler angles. The application of three sequential rotation matrices to a point will yield the same result as the application of the matrix that results from their multiplication, given that they are multiplied in the same order, since matrix multiplication is not commutative (Diebel, 2006). Of the 27 possible sequences of the three angles, only 12 satisfy the constraint that no two successive rotations may be about the same axis. These twelve can be separated into two groups: the classic Euler angles, representing rotations about one axis, then another and finally on the first axis which was moved to a new rotation, and Tait-Bryan or Cardan angles, where the three rotations are on three different axis (Winter, 2009).

The sequence used in this context is $z - x - y$. The rotation matrix is then given by:

$$\boldsymbol{R} = \boldsymbol{R}_y(\theta)\,\boldsymbol{R}_x(\phi)\boldsymbol{R}_z(\psi) = \quad (7)$$

$$= \begin{bmatrix} \sin\psi\cos\theta sin\ \phi + \cos\theta\cos\psi & sin\ \phi\ \sin\theta\cos\psi - cos\ \theta\sin\psi & cos\ \phi\sin\theta \\ \cos\phi\sin\psi & cos\ \phi\ \cos\psi & -sin\ \phi \\ \sin\psi\cos\theta sin\ \phi - \sin\theta\cos\psi & sin\ \phi\cos\theta\cos\psi + \sin\psi\sin\theta & cos\ \phi\ \cos\theta \end{bmatrix}$$

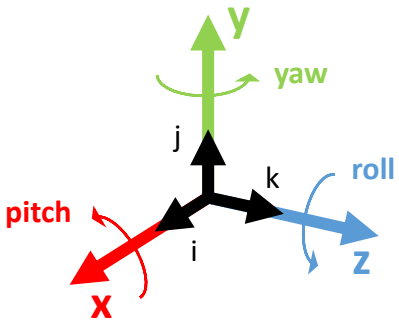

Figure 15. Original reference frame.

One problem that will be encountered when dealing with Euler angles is the gimbal lock problem. This problem is a singularity that occurs when $\phi = \pm 90°$, making the roll and yaw axis coincide and losing one degree of freedom. Indeed, when $\phi = 90°$, the rotation matrix becomes:

$$\boldsymbol{R} = \begin{bmatrix} cos\,(\theta - \psi) & \sin(\theta - \psi) & 0 \\ 0 & 0 & -1 \\ -\sin(\theta - \psi) & cos\,(\theta - \psi) & 0 \end{bmatrix} \quad (8)$$

In this situation the system depends on the difference between the two angles, having infinite solutions.

4.1.2 Quaternions

Another way to avoid gimbal lock is to use unit quaternions to represent an object's attitude. Quaternions can be defined as a vector space over reals. A quaternion $\mathring{q} \epsilon \mathbb{H}$ may be represented as a vector:

$$\mathring{q} = [q_0 \quad q_1 \quad q_2 \quad q_3]^T \quad (9)$$

Where q_0, q_1, q_2 and q_3 and scalars called the components of the quaternion. Alternatively, a quaternion may be represented defining a scalar part, the real number q_0, associated with a vector part $\boldsymbol{q} \epsilon \mathbb{R}^3$, defined as:

$$\boldsymbol{q} = \boldsymbol{i} q_1 + \boldsymbol{j} q_2 + \boldsymbol{k} q_3 \quad (10)$$

Where $\boldsymbol{i}, \boldsymbol{j}$ and $\boldsymbol{k}$ are standard orthonormal basis in $\mathbb{R}^3$. The quaternion is then given by the sum:

$$\mathring{q} = q_0 + \boldsymbol{q} \quad (11)$$

The adjoint, norm and inverse of a quaternion can be given by:

$$\overline{\mathring{q}} = \begin{bmatrix} q_0 \\ -\boldsymbol{q}_{1:3} \end{bmatrix} \quad (12)$$

$$\|\mathring{q}\| = \sqrt{{q_0}^2 + {q_1}^2 + {q_2}^2 + {q_3}^2} \quad (13)$$

$$\mathring{q}^{-1} = \frac{\overline{\mathring{q}}}{\|\mathring{q}\|} \quad (14)$$

Quaternion multiplication is not commutative, and can be defined by:

$$\mathring{q} \cdot \mathring{p} = \mathring{q}_m(\mathring{q}, \mathring{p}) = \boldsymbol{Q}(\mathring{q})\mathring{p} = \overline{\boldsymbol{Q}}\,(\mathring{p})\mathring{q} \quad (15)$$

In equation 15, $\boldsymbol{Q} : \mathbb{H} \rightarrow \mathbb{R}^{4\times4}$ is the quaternion matrix function, defined by:

$$\boldsymbol{Q}(\mathring{q}) = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \quad (16)$$

All quaternions in Unity are unit quaternions, i.e. $\|\mathring{q}\| = 1$. It is necessary to go from a rotation matrix to a quaternion and vice versa, in order to be able to perform operations with the rotation matrices and animate Unity objects, whose orientation is given by a quaternion. Considering a vector $\boldsymbol{z} \epsilon \mathbb{R}^3$ in global coordinates and $\boldsymbol{z}' \epsilon \mathbb{R}^3$ which is the same vector in body-fixed coordinates, then:

$$\begin{bmatrix} 0 \\ \boldsymbol{z}' \end{bmatrix} = \mathring{q} \begin{bmatrix} 0 \\ \boldsymbol{z} \end{bmatrix} \mathring{q}^{-1} = \mathring{q} \begin{bmatrix} 0 \\ \boldsymbol{z} \end{bmatrix} \bar{\mathring{q}} = \overline{\boldsymbol{Q}}\,(\mathring{q})^T \boldsymbol{Q}(\mathring{q}) \begin{bmatrix} 0 \\ \boldsymbol{z} \end{bmatrix} = \begin{bmatrix} 0 & \boldsymbol{0}^T \\ \boldsymbol{0} & \boldsymbol{R}_q(\mathring{q}) \end{bmatrix} \begin{bmatrix} 0 \\ \boldsymbol{z} \end{bmatrix} \tag{17}$$

Where $\boldsymbol{R}_q(\mathring{q})$ is the rotation matrix that can be given, in terms of the components of a unit quaternion, as:

$$\boldsymbol{R}_q(\mathring{q}) = \begin{bmatrix} {q_0}^2 + {q_1}^2 - {q_2}^2 - {q_3}^2 & 2q_1q_2 - 2q_3q_0 & 2q_1q_3 + 2q_2q_0 \\ 2q_1q_2 + 2q_3q_0 & {q_0}^2 - {q_1}^2 + {q_2}^2 - {q_3}^2 & 2q_2q_3 - 2q_1q_0 \\ 2q_1q_3 - 2q_2q_0 & 2q_2q_3 - 2q_1q_0 & {q_0}^2 - {q_1}^2 - {q_2}^2 + {q_3}^2 \end{bmatrix} \tag{18}$$

Therefore:

$$\boldsymbol{z}' = \boldsymbol{R}_q(\mathring{q})\boldsymbol{z} \tag{19}$$

From equation 18, it follows that:

$$4{q_0}^2 = 1 + r_{q11}(\mathring{q}) + r_{q22}(\mathring{q}) + r_{q33}(\mathring{q}) \tag{20}$$

$$4{q_1}^2 = 1 + r_{q11}(\mathring{q}) - r_{q22}(\mathring{q}) - r_{q33}(\mathring{q}) \tag{21}$$

$$4{q_2}^2 = 1 - r_{q11}(\mathring{q}) + r_{q22}(\mathring{q}) - r_{q33}(\mathring{q}) \tag{22}$$

$$4{q_3}^2 = 1 - r_{q11}(\mathring{q}) - r_{q22}(\mathring{q}) + r_{q33}(\mathring{q}) \tag{23}$$

$$4q_2q_3 = r_{q23}(\mathring{q}) + r_{q32}(\mathring{q}) \tag{24}$$

$$4q_1q_3 = r_{q31}(\mathring{q}) + r_{q13}(\mathring{q}) \tag{25}$$

$$4q_1q_2 = r_{q12}(\mathring{q}) + r_{q21}(\mathring{q}) \tag{26}$$

$$4q_0q_1 = r_{q23}(\mathring{q}) - r_{q32}(\mathring{q}) \tag{27}$$

$$4q_0q_2 = r_{q31}(\mathring{q}) - r_{q13}(\mathring{q}) \tag{28}$$

$$4q_0q_3 = r_{q12}(\mathring{q}) - r_{q21}(\mathring{q}) \tag{29}$$

It is then possible to arrive at four different mappings from a rotation matrix to a quaternion, $\mathring{q}_R^{\,i}: SO(3) \rightarrow \mathbb{H}$ for $i \in \{0,1,2,3\}$. Considering $\boldsymbol{R}$ the rotation matrix, then:

$$\mathring{q}_R{}^0(\boldsymbol{R}) = \frac{1}{2}\begin{bmatrix} (1 + r_{11} + r_{22} + r_{33})^{\frac{1}{2}} \\ (r_{23} - r_{32})/(1 + r_{11} + r_{22} + r_{33})^{\frac{1}{2}} \\ (r_{31} - r_{13})/(1 + r_{11} + r_{22} + r_{33})^{\frac{1}{2}} \\ (r_{12} - r_{21})/(1 + r_{11} + r_{22} + r_{33})^{\frac{1}{2}} \end{bmatrix} \quad (30)$$

$$\mathring{q}_R{}^1(\boldsymbol{R}) = \frac{1}{2}\begin{bmatrix} (r_{23} - r_{32})/(1 + r_{11} - r_{22} - r_{33})^{\frac{1}{2}} \\ (1 + r_{11} - r_{22} - r_{33})^{\frac{1}{2}} \\ (r_{12} + r_{21})/(1 + r_{11} - r_{22} - r_{33})^{\frac{1}{2}} \\ (r_{31} + r_{13})/(1 + r_{11} - r_{22} - r_{33})^{\frac{1}{2}} \end{bmatrix} \quad (31)$$

$$\mathring{q}_R{}^2(\boldsymbol{R}) = \frac{1}{2}\begin{bmatrix} (r_{31} - r_{13})/(1 - r_{11} + r_{22} - r_{33})^{\frac{1}{2}} \\ (r_{12} + r_{21})/(1 - r_{11} + r_{22} - r_{33})^{\frac{1}{2}} \\ (1 - r_{11} + r_{22} - r_{33})^{\frac{1}{2}} \\ (r_{23} + r_{32})/(1 - r_{11} + r_{22} - r_{33})^{\frac{1}{2}} \end{bmatrix} \quad (32)$$

$$\mathring{q}_R{}^3(\boldsymbol{R}) = \frac{1}{2}\begin{bmatrix} (r_{12} - r_{21})/(1 - r_{11} - r_{22} + r_{33})^{\frac{1}{2}} \\ (r_{31} + r_{13})/(1 - r_{11} - r_{22} + r_{33})^{\frac{1}{2}} \\ (r_{23} + r_{32})/(1 - r_{11} - r_{22} + r_{33})^{\frac{1}{2}} \\ (1 - r_{11} - r_{22} + r_{33})^{\frac{1}{2}} \end{bmatrix} \quad (33)$$

To avoid producing complex results, a composite function $\boldsymbol{q}_R: SO(3) \rightarrow \mathbb{H}$ can be defined in order to select the adequate mapping:

$$\mathring{q}_R(\boldsymbol{R}) \coloneqq \begin{cases} \mathring{q}_R{}^0(\boldsymbol{R})\ if\ r_{22} > -r_{33}, r_{11} > -r_{22}, r_{11} > -r_{33} \\ \mathring{q}_R{}^1(\boldsymbol{R})\ if\ r_{22} < -r_{33}, r_{11} > r_{22}, r_{11} > r_{33} \\ \mathring{q}_R{}^2(\boldsymbol{R})\ if\ r_{22} > r_{33}, r_{11} < r_{22}, r_{11} < -r_{33} \\ \mathring{q}_R{}^3(\boldsymbol{R})\ if\ r_{22} < r_{33}, r_{11} < -r_{22}, r_{11} < r_{33} \end{cases} \quad (34)$$

4.2 Software Development Kit (SDK)

In this section, the mathematical computations and computational methods implemented to access the data acquired by the inertial sensors system on Unity are presented.

In the use of the created software, movement acquisition is done using a wireless inertial sensor system (F.A.B.™ System). The system consists of IMU (inertial measurement units), which are placed over the skin or clothes using elastic bands, and a receiver (F.A.B Belt Clip receiver) responsible for receiving sensor data in real time and transmitting it to the computer to which it is connected. The computer will run a

program (frd2fab.exe) responsible for processing the sensor data into packets which are then sent to an established UDP (User Datagram Protocol) port. The UDP packets include, among other data, the Tait-Bryan angles of each sensor, represented as 16-bit signed numbers corresponding to the value of the angle in degrees multiplied by 80 (the full structure of the UDP packets is presented in Appendix A). Therefore, the numbers obtained range from -28800 to +28800, corresponding to an amplitude between -360.00° and +360.00°. In order to use these data in the Unity application, it was necessary to create a script to access the UDP port and read, in real time, the byte array composed of the information sent by the frd2fab program. In the script, the relevant Tait-Bryan angles are obtained by concatenating the 2 bytes that make up each one. Each value is then divided by 80 and limited to the interval [0°, 180°[U [-180°, 0°[in order to correspond to the value in degrees and for each sensor a three-dimensional vector is created where each component is the value of one of its Tait-Bryan angles. Rotations on Unity's y axis correspond to yaw, x rotations correspond to pitch and z rotations correspond to roll.

Applying equation 7, it is possible to get each inertial sensor's rotation matrix (and therefore, each body segment orientation) in Unity's coordinate axis.

4.3 Avatar Animator

The orientation of each segment of a rigged avatar is stored in Unity as a quaternion and for each inertial sensor there is now a matrix corresponding to its orientation in Unity's global coordinate axis. It is now necessary to apply the sensor rotations to each avatar segment. Consider $\boldsymbol{R}_S \in \mathbb{R}^3$ the matrix that corresponds to a sensor's orientation and $\mathring{q}_A \in \mathbb{H}$ the quaternion storing the orientation of the respective avatar segment. By applying equation 18 to the respective quaternion $\mathring{q}_A$, it is possible to get a matrix $\boldsymbol{R}_A \in \mathbb{R}^3$ corresponding to the orientation of the avatar segment. The relation between $\boldsymbol{R}_S$ and $\boldsymbol{R}_A$ can be given by:

$$\boldsymbol{R}_A = \boldsymbol{R}_S^{\boldsymbol{A}}\, \boldsymbol{R}_S \tag{35}$$

Where $\boldsymbol{R}_S^{\boldsymbol{A}} \in \mathbb{R}^3$ is the rotation matrix that transforms the orientation from global Unity coordinates to the avatar coordinates. This matrix must be calculated when the patient enters the game in the required position, as:

$$\boldsymbol{R}_S^{\boldsymbol{A}} = \boldsymbol{R}_A\, \boldsymbol{R}_S^{-1} \tag{36}$$

This matrix remains fixed, and during the game equation 35 is applied in order to obtain the avatar segment's orientation as a matrix. In order to actually rotate each segment accordingly, equation 34 must be applied to get the corresponding quaternion. The process is summarized in figure 16:

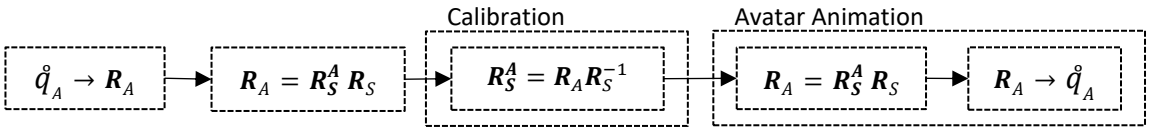

Figure 16. Schematic representation of the process of avatar animation.

A rigged avatar in Unity is composed of several game objects set in a hierarchy, where there are parent/child relationships. Due to the similar parent/child relationships of the F.A.B. System sensors, the rotations calculated for each avatar segment correspond to that segment's local rotation, i.e. the rotation relative to the parent object.

Figure 17. Example of avatar animation

4.4 Limitations

Despite the fact that the movement of the avatar was mostly accurate, smooth and with little delay, it was noted that if the patient moved their arms abruptly, the corresponding arm on the application would abruptly assume a wrong orientation and then return to the proper one. Although patient movement can

be tracked properly even with small errors while placing the sensors, acquiring an angle that corresponds to reality requires the sensors not to move from their place during the session and the calibration position to be carefully set.

The sensor system is also only capable of tracking body segment rotations and not positions, limiting the type of applications and games that can be done using the avatar's movement.

4.5 Game Design Specifications

In this section, several aspects which were taken into consideration during the development of the game present in the created software are discussed. The game was created for physical rehabilitation of children aged 7 to 13 years old who suffered elbow dislocations and no other health issues. It consists of a scenery where several characters appear and launch spherical projectiles of two different colours. The game objects used to represent the characters and the scenery were obtained through the asset store, being part of an asset called Voxel Scifi Environment.

The avatar was placed at a distance of 17u (Unity distance units) of the characters and has two disks of two different colours attached to its hands. After a period of adjustment where the patient will see the full body of the avatar and experiment with the movement, the game view changes and the actual game begins, where the patient will be moving the two disks and must catch the spheres with the correspondingly coloured disk a predetermined number of times in order to destroy the character that launched it. This means the player will be performing movements similar to those they would perform in conventional occupational therapy sessions in a more motivating and challenging way, while receiving visual feedback. As previously described, Mader *et al*. (2012) defined a therapeutic game design model, based on the relation between three aspects: the therapy, the player and the game. In this section, the created game is analysed in terms of the parameters described in this model.

4.5.1 Therapy

- Expected short-term therapeutic value: improvement of the elbow joint extension and flexion angle;
- Expected long-term therapeutic value: definitive recovery;
- Scientific proof of efficiency: to be evaluated;
- Scientific references: while this particular condition has not been evaluated, several sources report on the improvement of upper limb function using therapeutic games. (Betker et al., 2006; Merians et al., 2002; Broeren et al., 2004; Doyle et al., 2011; Fitzgerald et al., 2010 Fitzgerald et al., 2010)

4.5.2 Player

- Age range: 7 to 13 years old;
- Gender: Irrelevant;
- Particular conditions: elbow dislocation;
- Abilities: no other health conditions, good vision or corrective equipment;

4.5.3 Game

- Input system: motion capture using inertial sensors;
- Output system: visual information of game scenery and characters on a screen;
- Goals: eliminating all characters shown on screen;
- Feedback: the patient's arm movement controls the movement of two disks shown on screen; the spheres caught with the disks disintegrate in their own colour if caught with the right disk or in red if caught with the wrong one.
- Score: the score shown in-game reflects the number of spheres successfully caught with each disk. Other information is gathered during the game session that can later be displayed to give the patient an overview of their progression over time: the time spent on each level and the number of successful catches required to destroy a character, the maximum elbow flexion and extension angle achieved, the evolution of the elbow flexion and extension angle through the time of the game and the chosen projectile speed and time between projectiles being launched.
- Difficulty: the difficulty level can be adapted through several parameters. The game is divided in three levels that require progressively more elbow extension and have progressively more characters to destroy. The projectile speed and time between launches can be altered, along

with the number of required successful catches to destroy a character (in order to influence the time spent on each level), the percentage of projectiles of the colour of the disk corresponding to the injured arm (in order to make the patient perform the task with that arm more or less often) and the maximum torso lateral flexion angle allowed. This last parameter controls the way the patient is allowed to reach for the spheres with the disk, allowing or preventing torso movements that are usually adopted by people with this type of injury as a way to avoid extending the elbow. In-game, when torso lateral flexion exceeds the determined angle, all actions in the game will stop until the patient goes back to the proper position, discouraging this behaviour. This restriction is an important aspect of the game, since substitution movements are one of the main concerns of an occupational therapist when correcting patient movements and this system would reduce the need of constant supervision from the therapist while performing these movements. All the difficulty parameters can be changed at any moment.

- Variability: while the task to be performed is always the same, each level of the game has progressively more characters to destroy and requires a higher amplitude of extension of the elbow.
- Usability: the only ability necessary to play the game is bilateral upper limb function along with being able to distinguish colours and good vision or the use of proper corrective equipment. Therefore, children whose only health condition is a dislocated elbow will be able to play this game safely.
- Expected positive side-effects: it is expected that the patient will become more motivated towards the task, and the will to complete the game will make the patient less aware of any pain he might have, thus making the experience more pleasant and possibly making the patient extend and retract the arm further than they normally would. The outcomes in terms of flexion and extension angle are expected to be at least similar to those obtained through conventional therapy.
- Reported serious uses: features like VR-based biofeedback have been used for therapeutic games with some success in the past, with reports of outcomes comparable to those of conventional therapy and increased patient motivation and satisfaction.

Figure 18. Game definitions window, where the following parameters can be altered: initial level, the arm being evaluated, game speed, minimum time between projectiles, torso lateral flexion restriction angle, percentage of projectiles to be caught with the side being evaluated and the number of required successful catches in order to destroy a character.

Figure 19. Game environment and avatar during the adaptation period.

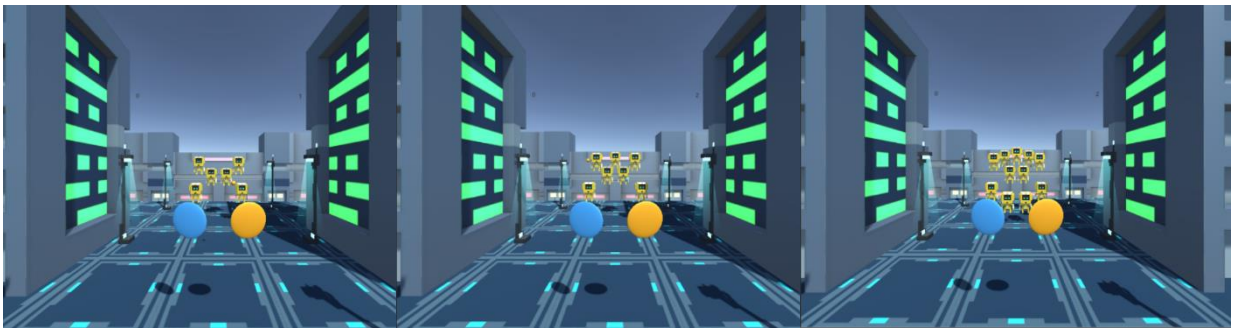

Figure 20. Game environment on levels 1 (left), 2 (centre) and 3 (right)

4.5.4 Relations between Aspects

- Player/Therapy: the people who would play this game would have suffered a dislocation of the elbow, which is a condition that can be improved by upper limb occupational therapy, namely movements like arm extension and flexion.
- Game/Therapy: the game is meant to be played in a clinical setting, with a therapist, during occupational therapy sessions. The visual aspects of the game, along with the character's destruction and the distinct effects between a successful and unsuccessful catch are meant to motivate the patient to advance in the game. The required arm movements along with the torso movement restrictions make up the therapeutic aspect of the game, encouraging the patient to not only perform the movements but also perform them without any substitute movements that would allow them to reach farther without further extension of the elbow. Furthermore, throughout the use of the game, movement data is being collected and stored along with other information, in order to provide the patient and/or the therapist quantitative information regarding performance, from the parameters used in each session to the time needed to complete each level, the maximum extension and flexion angles and the angles' progression throughout the session.
- Player/Game: the game has average physical and cognitive requirements and its use has no known adverse effects. The scenery and mechanics were designed taking into account the target demographic both in terms of visual aspect and game difficulty, making it an appealing experience. The game aims to have a difficulty level that makes it interesting, without being discouraging or too easy, as this could lead to boredom. This aspect can be adjusted to suit the patient's personality and injury level. The game was divided between levels in order to give the patient a sense of accomplishment upon finishing each stage and to allow the therapist to stop the game after a certain number of levels, according to the severity of the injury and the stage of therapy in which the patient is at the moment.

4.6 Information Management

One of the most important part of this software is the ability to save and display quantitative patient data throughout sessions, to be later analysed by the therapist. Two assets were acquired from Unity's Asset Store to achieve this goal: Database Control Pro, by Solution Studios and MeshChartFree.

Database Control Pro is a solution made for creating an online server back end, allowing for the implementation of the login and register systems of the game, along with the ability to save and load data. This asset allows for the creation of an unlimited number of databases with an unlimited number of accounts, with virtually an unlimited amount of data, saved in a secure way. This is done through the creation of Command Sequences, a visual scripting solution that can be used to read and write data to and from the databases, as well as perform some operations with the data.

In the created database, each patient has an identifying row and then a set of data rows, each corresponding to one therapy session. In the identifying row, it is stored the patient's name, a password, the patient's date of birth and laterality and the number of sessions. After each session, the number of sessions column is updated and a row is created, starting at the third column, containing the following information:

- Name of the injured articulation;
- Pathology;
- Injured side;
- Name of the chosen game;
- Percentage of projectiles caught with each side;
- Maximum flexion angle;
- Maximum extension angle;
- Date of the session;
- Time spent on each level;
- Average time per successful catch of a projectile;
- Patient's perceived pain and satisfaction levels;
- Speed;
- Time between projectiles being launched;
- Evolution of the extension angle throughout the session;
- Evolution of the flexion angle throughout the session.

Some of these parameters are specific to the game and would be different in different games. The evolution of the flexion and extension angles was obtained by registering the angle every 0.1 s, adding it

to one of two arrays, depending on whether it was extension or flexion, and then finding local maximum flexion or extension angles. The gathered data can be later displayed as text, apart from the angle evolutions, which are only displayed as charts, as shown on Figure 23.

Besides the login, register and data saving command sequences, it was necessary to create command sequences to retrieve a session's date in order to identify it, to aggregate a session's data and display it as a meaningful text, showing each parameter and its value, and to retrieve the maximum flexion and extension evolution throughout a session or maximum flexion and extension registered for all sessions of a patient, in order to use those values to create charts

In order to create the charts, the MeshChartFree asset was used. The charts can contain one session's maximum flexion or extension evolution or the maximum flexion or extension of each session. This information may be useful to assess patient recovery through time.

Figure 21. Login window of the application.

Figure 22. Register window of the application

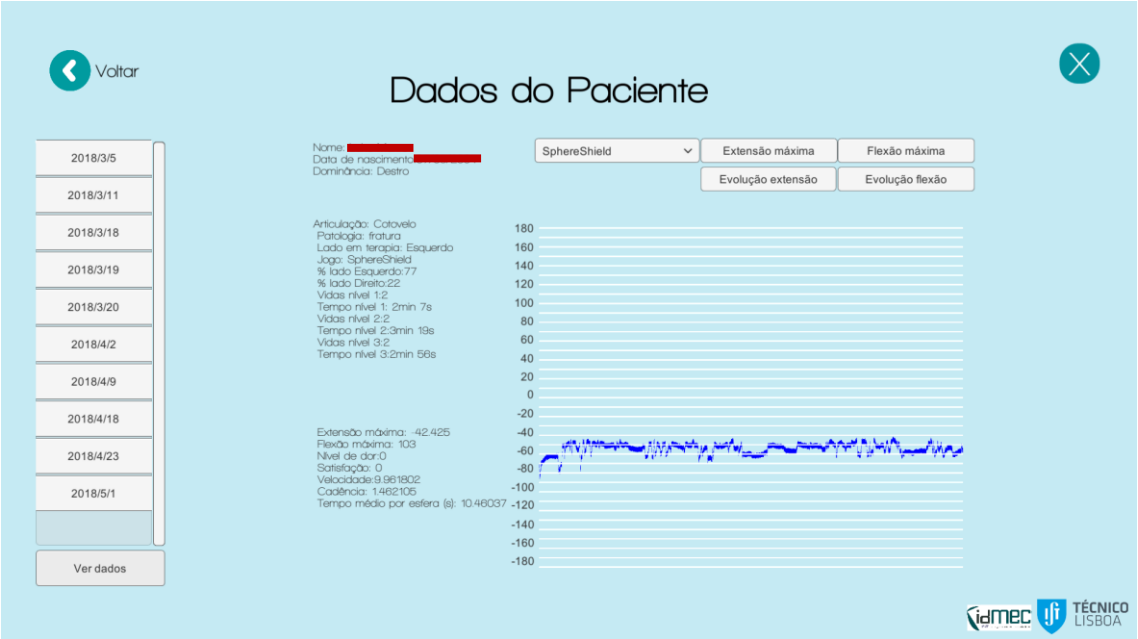

Figure 23. Patient data window of the application.

5 Application to Paediatric Rehabilitation

5.1 Pathology

The elbow joint unites three bones: the humerus, which forms the upper part of the joint and widens near the end into the medial and lateral epicondyles, the ulna, situated on the inside of the joint and the radius, the smaller of the forearm bones, situated on the outside of the joint. Both the radius and ulna have cup-shaped heads in order to articulate with the capitulum of the humerus. The elbow is a modified hinge joint composed of three articulations, which provide most of the elbow's stability against varus and valgus stress and extreme extension and flexion: the radiocapitellar, ulnohumeral and proximal radioulnar joints (Hickey and Loebenberg, 2006). Around 55% of varus stability in extension and up to 75% in 90^{o} flexion is provided by the ulnohumeral articulation, while valgus stability is divided between the anterior capsule, the bony articulation and one of the ligamentous structures, the medial collateral ligament (MCL) both in extension and in 90^{o} flexion (Hickey and Loebenberg, 2006). The humeroulnar joint is responsible for most of the extension and flexion of the elbow, while the radiocapitellar joint is responsible for pronation and supination. There are two main ligaments supporting the elbow joints: the medial collateral ligament (MCL) and the lateral collateral ligament (LCL). The MCL consists of three bundles: anterior, posterior and transverse (Hickey and Loebenberg, 2006; Halstead, 2017). The anterior bundle originates on the anteroinferior medial epicondyle and inserts onto the medial coronoid process and contributes to 55% to 75% of valgus stability. It can be divided into two bands, where the anterior band provides resistance to valgus stress up to 90^{o} flexion and the posterior band becomes more relevant between 120^{o} and full flexion (Hickey and Loebenberg, 2006). The LCL is a complex consisting of the annular, radial collateral, and lateral ulnar collateral ligaments. The annular ligament stabilizes the proximal radioulnar joint and is the termination point of the radial collateral ligament, originated in the lateral epicondyle (Hickey and Loebenberg, 2006). The lateral ulnar collateral ligament is the primary restraint to varus stress, and originates from the lateral epicondyle, blending with the annular ligament but arching superficial and distal to it (Hickey and Loebenberg, 2006). A healthy elbow will usually have a range of motion (ROM) of extension to 0^{o} and flexion to 150^{o} (Halstead, 2017). In this case, the convention used for angle measurement was that extension goes from 0^{o} (full extension) to -90^{o} and flexion goes from 90^{o} (corresponding to a -90^{o} extension) to the maximum value, around 150^{o}, as shown in Figure 26.

Traumatic elbow dislocations are rare in children with an incidence of 3% to 6% of all elbow injuries, but they are the most common pediatric joint dislocations (Rasool, 2004). They are usually the result of a fall onto an outstretched hand, resulting in an axial compressive and valgus load on the supinated extremity, and 10% to 50% are sports related (Hickey and Loebenberg, 2006; Halstead, 2017). Elbow dislocations can be classified in terms of the position of the proximal radio-ulnar joint in relation to the distal humerus as posterior, anterior, medial, lateral and, less frequently, convergent or divergent (Rasool, 2004), being that

posterior or posterolateral dislocations account for approximately 90% of dislocations (Halstead, 2017; Cohen and Hastings, 1998). They can also be complete or partial (subluxation), depending on whether the joint surfaces are completely separated or not. Dislocation of the elbow is usually accompanied by damage to the capsule and ligaments and may be associated with extraarticular or intraarticular fractures (Borris *et al*., 1987; Hickey and Loebenberg, 2006). In the paediatric population, it is important to note that there are 6 ossification centres (capitellum, radial head, medial epicondyle, trochlea, olecranon and lateral epicondyle) which may be mistaken for fractures when seen on an x-ray (Halstead, 2017).

Following reduction of the joint, i.e. manipulation of the joint back in place, several treatments can be employed with varying degrees of success, but most authors recommend 3 to 10 days of joint immobilization (Ross *et al*.,1999). Although most of the time patients end up completely recovering elbow function, albeit with a slightly reduced range of extension (Ross *et al*.,1999), several studies conclude that early motion of the injured elbow is associated with an improved and faster outcome. Prolonged immobilization (longer than 3 weeks) may mean a more severe loss of ROM (Ross *et al*.,1999; Halstead, 2017).

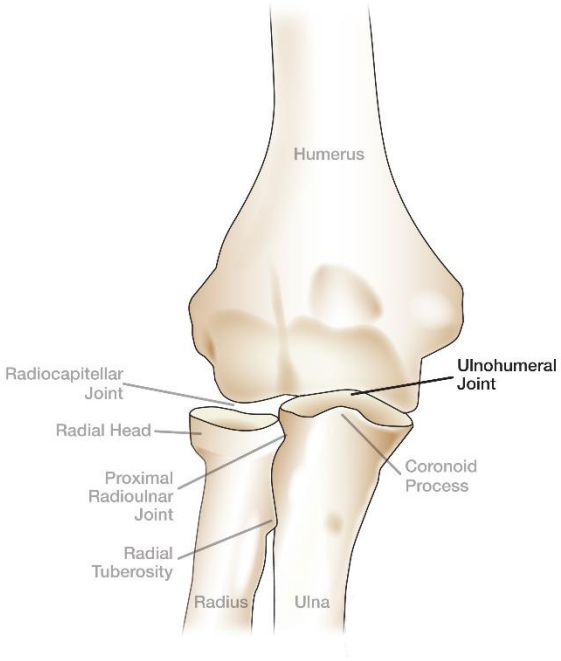

Figure 24. Articulations composing the elbow joint. (http://www.assh.org/handcare/)

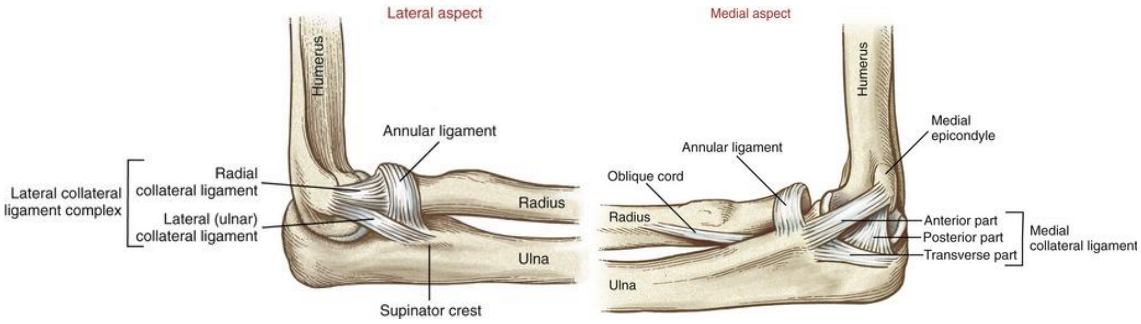

Figure 25. Medial and lateral view of elbow ligaments. (https://clinicalgate.com/elbow-and-forearm-3/)

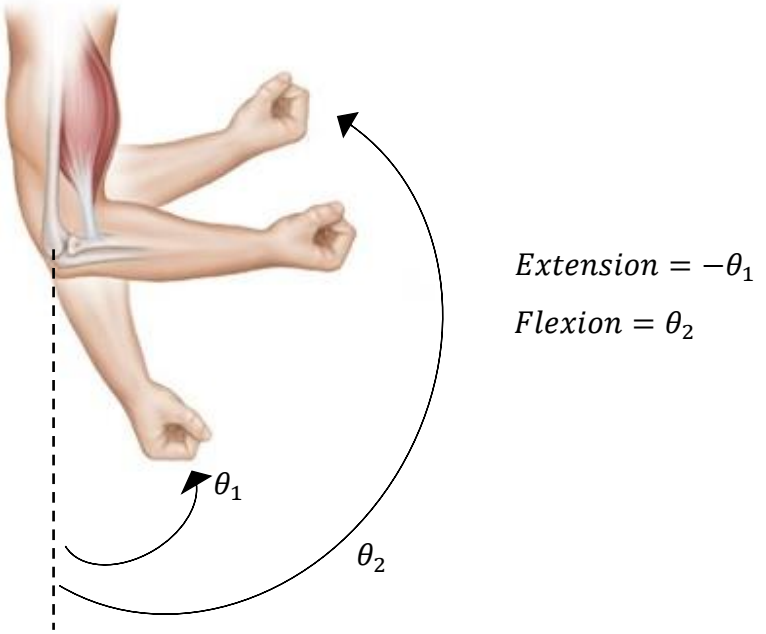

Figure 26. Angle measurement method.

5.2 Control group and work group

In order to assess the viability of the created application, both as a therapeutic game and as a tool to assess, save and display patient information, it was necessary to test it. Before starting trials on injured subjects, it was desirable to understand what the ideal game parameters would be, in terms of projectile velocity and time between projectiles being launched, and whether the game was enjoyable or not, for subjects in the previously defined age range. A control group of 11 subjects, 10±3 years old (4 female) with no physical conditions was created. Each subject was registered in the database and equipped with the inertial sensors, then tried the game and answered questions regarding their experience, namely:

- Game enjoyability (High/Medium/Low)
- Perceived game speed (Fast/Adequate/Slow)
- Equipment comfort (Yes/No)

The data acquired by the software on each session was stored in the database.

The software was also used on a work group of one 13 years old male subject who had suffered a complete lateral elbow dislocation on the left side and was performing occupational therapy at Lisbon's Children's Hospital (Hospital D. Estefânia – CHLC) following 3 weeks of immobilization. The subject was equipped with the inertial sensors, registered in the database and for a total of 6 sessions, the patient went through the conventional therapy session and in the end played the developed therapeutic game and the acquired data was registered in the database. It was possible to accompany the subject from the first session after immobilization. The subject was also questioned regarding the parameters and subjective experience with the game.

The parameters used were:

- Speed: 10 u/s;
- Minimum time between launches: 1.5 s;
- Number of hits to destroy character: 2;
- Torso lateral flexion restriction: 20 ^{o};
- Percentage of projectiles launched with the colour of the injured side: 50 % for the control group, 70 % for the injured subject (as suggested by the occupational therapist).

Since all participants were underage, an informed consent form, which is presented in Appendix B, was signed by their legal representatives, in order to get permission for their participation with clear understanding of the implications and consequences of the experiment. The control group sessions occurred at the Atlético Clube de Portugal installations, apart from subject 10, who was tested at Lisbon's Biomechanics Laboratory (LBL). The work group sessions occurred in Lisbon's Children's Hospital.

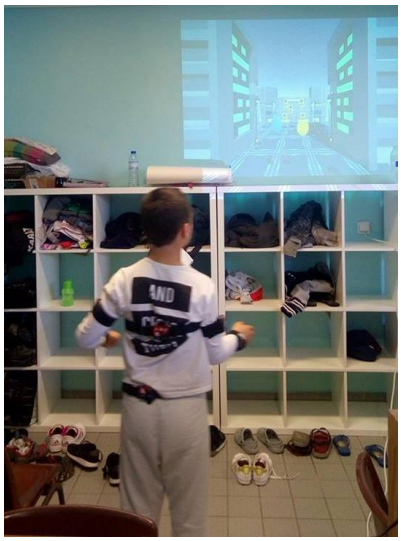

Figure 27. Subject belonging to the control group playing the game

5.3 Results

5.3.1 Control Group

On table 3, the relevant control group results are presented.

Table 3. Results for the control group

Subject	Age (years)	Gender (M/F)	Enjoyment	Perceived speed	Comfort	Average time per successful catch (s)	Dominant hand	Catches with left arm (%)	Catches with right arm (%)
1	10	F	High	Slow	Yes	14	Right	60.0	40.0
2	9	M	High	Adequate	Yes	14	Right	50	50
3	11	M	High	Adequate	Yes	5	Right	38.7	61.3
4	9	F	Medium	Fast	Yes	32	Left	50	50
5	10	F	High	Adequate	Yes	25	Right	55.5	44.4
6	12	M	High	Slow	Yes	5	Right	56.8	43.2
7	7	M	High	Adequate	Yes	18	Right	55	45
8	8	F	High	Slow	Yes	13	Right	45	55
9	9	M	High	Adequate	Yes	18	Right	50	50
10	13	M	Low	Fast	Yes	35	Right	46.2	53.8
11	11	M	Medium	Adequate	Yes	15	Right	41.2	58.8
Average	10	-	-	-	-	18	-	49.9	50.1
Mode	9	M	High	Adequate	Yes	-	Right	-	-
Standard deviation	1.68	-	-	-	-	9.2		6.4	6.4

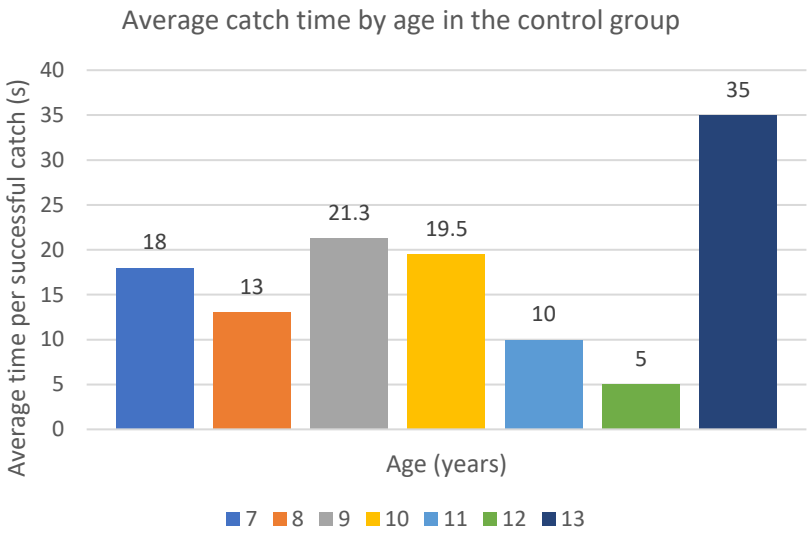

Figure 28. Average successful catch time by age in the control group

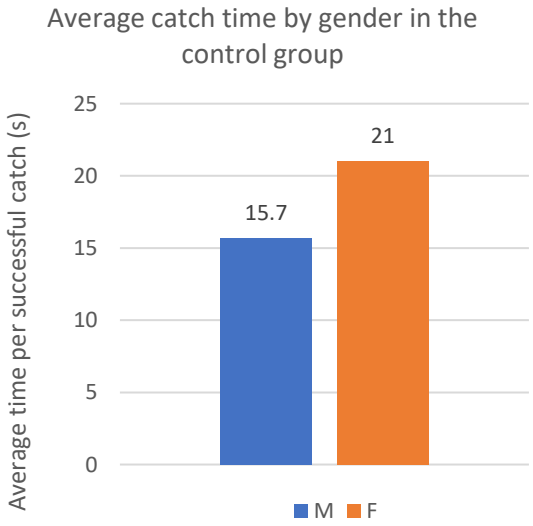

Figure 29. Average successful catch time by gender in the control group

As can be seen in Table 3, most subjects in the control group enjoyed the experience and all reported being comfortable with the equipment on. Despite the fact that there were some issues with the sensors on the right arm, the percentage of catches with each arm was very similar for the cases where the projectiles had a 50 % chance of being either colour. This result shows that the game was designed properly to promote movement of both arms. There was a significant standard deviation of the average time per successful catch. The small sample size of the control group means that the conclusions that can be taken from the data analysis may not translate into the general population. As seen in Figure 28 there is an increase in average time at 13 years old. This subject (subject 10) was tested in a different setting than the rest of the control group, where they were at a bigger distance from the game projection, and

reported some difficulty seeing the projectiles, which might have influenced both the time per successful catch and enjoyment. Another reason might be that the game is not adequately appealing for that age. However, it is worth noting that from 9 to 12 years the average successful catch time decreased, which might indicate that the game should be made faster for older subjects, as expected. However, the sample size is not big enough to allow that conclusion. In general, it was noticed that the subjects level of enthusiasm and competitivity, along with the non-existence of issues with the sensors during the session, was more relevant for the time the subject would take than the subject's age. Regarding subject gender, the female subjects have a higher average catch time than the males, being slightly above the general average while the males were slightly below, as seen in Figure 29. However, due to the small sample size, it is not possible to rule out the possibility that this difference may be due to age as well. In order to properly assess the relation between age or gender and the average time between catches, it would be necessary to have a bigger and more balanced control group both in terms of gender and age. Given these results, however, it appears that the game should be recommended for children under the age of 13 and that the therapist should lower the speed of the game for younger patients and/or female patients.

5.3.2 Work Group

The injured subject on which the software was tested at Lisbon's Children's Hospital reported no pain during the game, a high level of satisfaction and comfort with the equipment and considered the game speed adequate to make the game entertaining and challenging without the session becoming too tiring or painful. The subject's therapist reported a high level of interest in the application, namely due to the data acquisition capabilities and torso movement restriction. The relevant results for the work group are presented in Table 4.

Table 4. Results for the work group

Session number	Date (dd/mm)	Average time per successful catch (s)	Time per level (mm:ss)			Maximum flexion (º)	Maximum extension (º)	Catches (%)	
			1	2	3			Left	Right
1	05/04	15	01:44	02:39	06:23	-	-27.1	43.2	56.8
2	11/04	13	02:04	02:00	05:45	115.7	-25.6	59.1	40.9
3	18/04	15	01:50	02:08	08:02	121.7	-32.7	51.1	48.9
4	02/05	15	01:50	03:12	05:54	122.1	-33.3	61.4	38.6
5	09/05	16	02:48	03:37	00:00	106.7	-30.3	83.3	16.7
6	18/05	11	01:44	02:41	00:00	116.2	-35.8	87.5	12.5
7	01/06	10	02:07	03:19	02:56	103	-42.4	77.7	22.3
Average		14	02:06	02:48	04:09	114.2	-32.5	66.2	33.8
Standard deviation		02	00:23	00:32	02:47	6.6	4.9	-	-

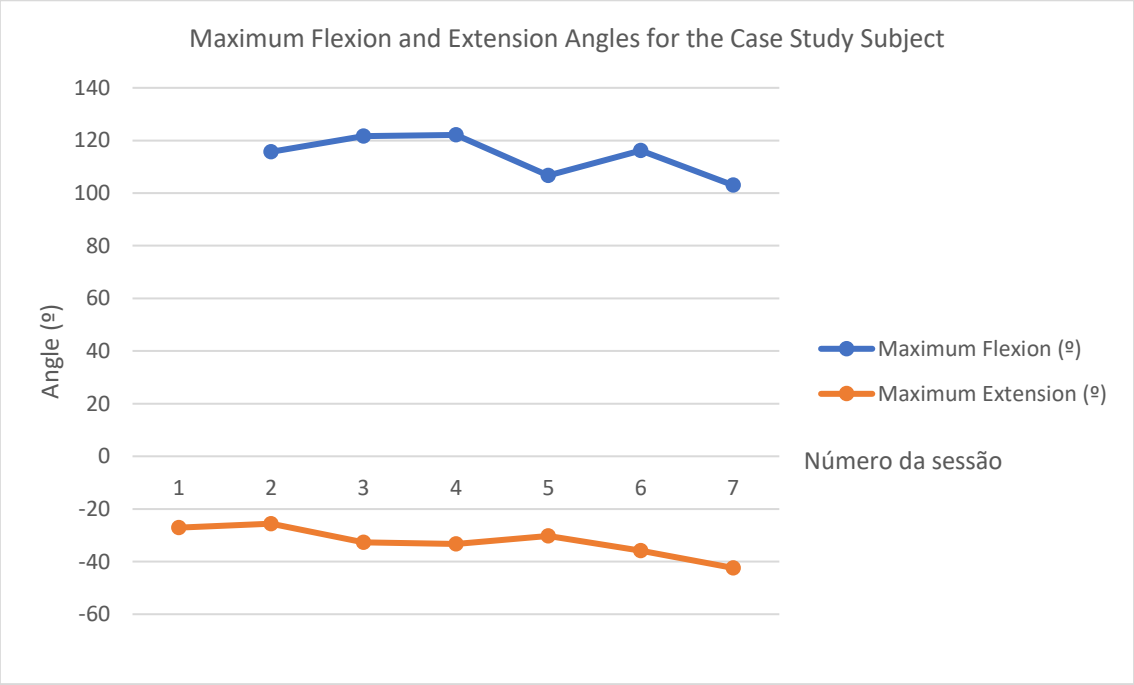

Figure 30. Maximum Flexion and Extension Angles for the Case Study Subject

Considering the information on Table 4 it is possible to see that, despite the fact that the left elbow was injured, the percentage of catches on the left side was higher. This is due to the fact that after session 1 the percentage of projectiles meant to be caught with the injured side was increased from 50 % to 70 %. For the first four sessions, it is possible to see an increase in projectiles caught with the injured side, proving this to be an effective method to increase movement on that side. In the last three sessions, movement reconstruction on the right arm was not completely correct, accounting for the sharp increase in the use of the left side. The average time per successful catch aligns with the average for male subjects. However, due to the subject's age, it was expected that his interest would be lower, making the time between catches higher. Yet, the subject reported a high level of satisfaction with the game and his average time per catch is less than half the control group value for the same age. This difference can be explained by the visual difficulties and lack of enjoyment that subject 10 from the control group reported.

As expected, the time increases with each level, due to the higher number of characters to destroy and the increased difficulty reaching them. With constant parameters, the time spent on each level does not fluctuate much, although it would be expected to decrease after a bigger number of sessions.

In terms of joint angles, as seen in Figure 30, both the extension and flexion angle fluctuate through the sessions, with no visible long-term improvements. This is not an expected result, particularly since the subject was being subjected to conventional therapy as well and the measurements done with a goniometer corroborate the results obtained from the software. It was found that the subject suffered from posttraumatic calcifications, leading to stiffness and leading to these results.

During two sessions, the speed was increased to 15 u/s and the time between launches was decreased to 0.5 s. The results from those sessions can be viewed on Table 5.

Table 5. Results for the work group on the sessions where parameters were changed.

Session Number	Date (dd/mm)	Average time per successful catch (s)	Time per level (mm:ss)			Maximum flexion (º)	Maximum extension (º)	Catches (%)	
			1	2	3			Left	Right
1	19/04	25	03:24	03:29	12:26	143.4	-49.3	69.6	30.4
2	20/04	18	03:04	02:36	08:50	115.4	-45.4	66.7	33.3
Average		22	03:14	03:03	10:38	129.4	-47.35	68.1	31.9
Standard deviation		04	00:10	00:26	01:48	14	1.95	1.4	1.4

As expected, the time per level was much higher in these sessions. While the subject never reported pain, he reported higher level of exhaustion and frustration in these sessions, meaning that these parameters were not ideal. It is worth noting, however, that with these parameters the subject performed more movements with his left arm, meaning that an increase in the game's difficulty might make the patient more eager to not miss opportunities and more distracted, leading them to use the injured arm more. On the second session, the patient actually took less time on the second level than the first, and even in the first, the difference between the two levels is only 5 s. This might be due to an increase in the time needed to adapt to the game when the difficulty is increased. The registered angles, however, mark these sessions as the ones with the worst results. However, in one of these sessions there was no previous therapy session, which usually increases the range of motion, and on the other the patient had shown poor results already in conventional therapy. These results can also be attributed to the patient having less time to concentrate on the task and perform it to the maximum of his abilities, leading to poorer results. The maximum flexion value obtained in session 1 was attributed to issues with motion tracking.

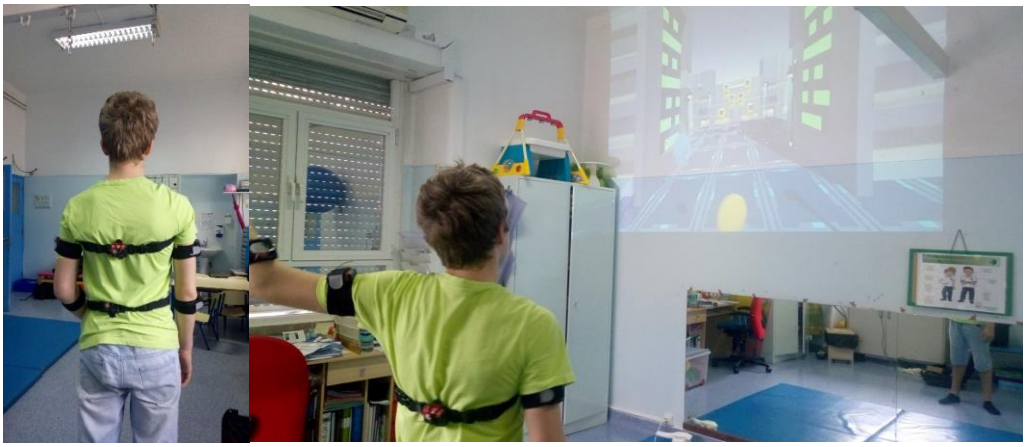

Figure 31. Work group subject wearing the sensors (left) and playing the game (right).

5.4 Discussion

Although the control group was too small to draw accurate conclusions from, it was possible to have a general idea of certain aspects of the game: the speed of the game should most likely be reduced in female patients and younger patients, older patients might not show the same interest in the game as younger ones and the personality of the patient, namely in terms of competitivity and motivation, must be taken into account. It might also be important to have the game in a setting where the patient can be close to the game's projection, in order to avoid visual difficulties. Since all the subjects in the control group practiced sports, the adequate parameters for them might not be adequate for a more sedentary

population. However, since most elbow dislocations occur due to sports activities, it is probable that most patients who would use the game in a clinical setting would find the settings adequate. Despite the conclusions regarding the relation between game parameters and age and gender, in a clinical setting the most important aspects to be taken into account are the severity of the injury and the number of therapy sessions the patient has done before. A patient that would otherwise enjoy the game in a difficult setting might struggle with it if their injury is severe, and the application of inadequate game parameters might lead to further lesion of the elbow by forcing the patient to perform motions that they are not apt to. It is also worth noting that an increase in difficulty in the work group led to worse results, either through patient demotivation, exhaustion or decreased focus on the task or due to circumstances particular to this case, meaning that while the results seem to indicate that a moderate difficulty will lead to better results, it would be necessary to have more sessions with variable parameters in order to assess the relation between difficulty and results.

It is worth noting that the case study subject's posttraumatic calcifications meant that, despite the therapy sessions and the work being done outside the hospital, there was no permanent improvement of either the extension or flexion angle. In fact, over several sessions, both the results from the created software and from the therapy sessions show that despite there being some improvements from one session to the next, the progress was not linear and some sessions showed worse results than the session before. Therefore, the lack of patient improvement over the sessions makes it difficult to assess the viability of the software as a therapeutic tool. The subject, however, reported positively on the therapeutic game, considering it challenging but not demotivating, and a more interesting experience than conventional exercises. Along with the results from the control group which indicate, in general, a high level of enjoyment playing the game, this allows to conclude that patient motivation would increase using systems such as the one created in this thesis to perform therapeutic exercises.

The restriction on torso lateral flexion proved to be an effective mechanism in reducing movement substitutions. During the first session with the work group subject, it was noted that the game stopped several times due to torso lateral flexion, but during the remaining sessions the subject reduced significantly the number of moments of movement substitution. In the control group, despite there only being one session per subject, it was noted that in the beginning of the session the number of substitution movements was much higher than by the end. Since this is one of the main concerns of occupational therapists when correcting patient movement, this system would reduce the need for supervision during the performance of the therapeutic movements. Besides, the fact that there is a punishment (the inability to advance in the game) for an incorrect movement made the subjects perform the movements correctly much faster than if they are simply being verbally corrected, meaning that the use of this system could increase efficiency in the treatment by reducing the number of incorrectly performed movements.

It is worth mentioning that in the start of sessions, subjects tended to need some time to adapt to the virtual environment, taking some time to understand the relative positions of the disks and the projectiles

and where the disks had to be in order to catch the projectiles. While most subjects were quick to adapt to the environment, some had difficulties, reducing their motivation and enjoyment of the game.

During the time the application was being tested with both the control and work group, some changes were made to the game, either by request of the therapist or because issues were noticed. Namely, character position was changed in order to make the subjects perform more flexion and extension and the percentage of projectiles with the colour of the disk on the monitored side and the torso lateral flexion restriction angle were made adaptive. In terms of the control group, only the first parameter is relevant, since it means that not all subjects were subjected to the exact same game, which might influence the results. In terms of the work group, this means that during the first sessions, the restriction angle was higher, making the game easier, and the percentage of projectiles of each colour was 50 %, which might have interfered with the results, since the subject was not being forced to move the injured arm as much.

One issue arose from the way the sensors were secured to the body. While the arms and forearm sensors remained stable throughout the session, the pelvis and trunk sensors, particularly the trunk sensor, moved down the subject's back due to the way they were being held in place. The straps which held them quickly loosened due to the patient's arm movements, making the sensor slide down and get a different orientation, meaning that the avatar's pose would not correspond to the subject's, and even on the tightest setting, the pelvis and trunk straps were not adequate for small children.

6 Conclusions and Future Developments

Systems such as the one created for this thesis show great promise in therapeutic applications by increasing patient motivation, guiding the patient to perform the movements correctly and allowing for a quick, accurate quantitative analysis of relevant parameters.

In the future, it would be important to improve the way the game makes the subject perform progressively broader movements by analysing character positions that would require the patient to achieve certain extension of flexion angles and imposing, for example, angles that the patient must be performing in order for the catch to be considered valid. It would be interesting to create other games in order to appeal to a broader population and to give the patients more options in order to reduce boredom.

In order to adapt the system to children and ensure that the sensors are placed correctly by any person, it would be important to create a better way to attach the sensors to the body. Besides, it was not yet possible to avoid using the F.A.B. Recorder software and access the frd2fab.exe program directly, which means that the calibration step is done before the created software is executed and the F.A.B. Recorder software must be stopped after calibration to allow access to the UDP port. It would then be necessary to improve the SDK in order to allow direct access to frd2fab.exe.

Due to the limited time and the difficulty of the work group subject's injury, it was not possible to assess accurately the therapeutic validity of this software. A more prolonged study with more subjects would be required in order to assess the patient's improvement, along with how the novelty effect affected the way patients responded to the game, i.e. if throughout time patient interest in the game remained high. In a more distant future, with a more complete software, it would be relevant to compare patients subjected to just conventional therapy and therapeutic exercises and patients who were subjected to the other aspects of conventional therapy but only performed exercises using the games. It would also be necessary to assess the influence of game difficulty in the results obtained, to discover whether a higher difficulty will make the patient less aware of his injury and more eager, obtaining better results, or have a demotivating effect and not let the patient concentrate on the task, leading to poorer results.

Exploration of ways to make this system available for home-based treatments, allowing therapists access to the data registered when patients perform the exercises at home should also be considered, since one of the advantages of this system is the automatic correction of movements, reducing the need for a therapist to be present in order to make the patient perform the movements correctly.

The use of other movement tracking and biofeedback systems, namely VR headsets, that would allow for a much higher level of immersion, could provide a more interesting experience and greatly expand the possibilities for the type of application that could be created. However, these systems still present some issues that must be resolved before therapeutic use can be considered.

It is also worth noting that Unity is a very versatile game engine, allowing for quick changes to adapt the same game for several age groups with changes of scenery, as well as the creation of more games for the same or different pathologies.

References

A

Agmon, M., Perry, C. K., Phelan, E., Demiris, G., Nguyen, H. Q. (2011) A pilot study of Wii Fit exergames to improve balance in older adults. *Journal of Geriatric Physical Therapy*, 34(4), 161-167.

Anderson-Hanley, C., Arciero, P.J., Brickman, A.M., Nimon, J.P., Okuma, N., Westen, S.C., Merz, M.E., Pence, B.D., Woods, J.A., Kramer, A.F., Zimmerman, E.A. (2012) Exergaming and older adult cognition: a cluster randomized clinical trial. *American Journal of Preventive Medicine*, 42(2), 109-128.

Antonin, Š. (2017) Comparison of Unity and Unreal Engine. *Czech Technical University in Prague.*

Archambault, P., Tao, G., Solomon, J. M., Kairy, D., Levin, M. F. (2015) Technologies applied to PMR. *Annals of Physical and Rehabilitation Medicine*, 58, 103-107.

B

Best, R., & Begg, R. (2006). Overview of movement analysis and gait features. *Computational intelligence for movement sciences: neural networks and other emerging techniques*, *1*, 1-69.

Betker, A. L., Szturm, T., Moussavi, Z. K., Nett, C. (2006) Video game-based exercises for balance rehabilitation: a single-subject design. *Archives of Physical Medicine and Rehabilitation*, 87(8), 1141-1150.

Bonnechère, B., Jansen, B., Salvia, P., Bouzahouene, H., Omelina, L., Moiseev, F., Sholukha, V., Cornelis, J., Rooze, M., & Jan, S. V. S. (2014). Validity and reliability of the Kinect within functional assessment activities: comparison with standard stereophotogrammetry. *Gait & Posture*, *39*(1), 593-598.

Borris, L. C., Lassen, M. R., Christensen, C. S. (1987) Elbow dislocation in children and adults: A long-term follow-up of conservatively treated patients. *Acta Orthopaedica Scandinavica*, 58(6), 649-651)

Broeren, J., Rydmark, M., Sunnerhagen, K. S. (2004) Virtual Reality and Haptics as a Training Device for Movement Rehabilitation After Stroke : A Single-Case Study. *Archives of Physical Medicine and Rehabilitation*, 85(8), 1247-1250

C

Carlomagno, N., Tore, S., Sgambelluri, R., Fragnito, V., Paloma, F. G. (2011) New Technologies for Motor Didactics. *Journal of Physical Education and Sport,* 13(2), 154-159

Ceseracciu, E., Sawacha, Z., Cobelli, C. (2014) Comparison of Markerless and Marker-Based Motion Capture Technologies through Simultaneous Data Collection during Gait: Proof of Concept. *PLoS One*. 9(3), 1-7.

Chang, Y. J., Chen, S. F., & Huang, J. D. (2011). A Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities. *Research in Developmental Disabilities*, *32*(6), 2566-2570.

Charlton, I. W., Tate, P., Smyth, P., & Roren, L. (2004). Repeatability of an optimised lower body model. *Gait & Posture*, *20*(2), 213-221.

Clark, R. A., Pua, Y. H., Fortin, K., Ritchie, C., Webster, K. E., Denehy, L., & Bryant, A. L. (2012). Validity of the Microsoft Kinect for assessment of postural control. *Gait & Posture*, *36*(3), 372-377.

Cohen M S, Hastings II H. Operative release for elbow contracture. The lateral collateral ligament sparing technique. Orthop Clin North Am 1999; 30: 133-9.

Corazza, S., Muendermann, L., Chaudhari, A. M., Demattio, T., Cobelli, C., & Andriacchi, T. P. (2006). A markerless motion capture system to study musculoskeletal biomechanics: visual hull and simulated annealing approach. *Annals of Biomedical Engineering*, *34*(6), 1019-1029.

Craighead, J., Burke, J., Murphy, R. (2008) Using the Unity Game Engine to Develop SARGE: A Case Study. *Itsec*, 4552

D

Davis, R. B., Ounpuu, S., Tyburski, D., & Gage, J. R. (1991). A gait analysis data collection and reduction technique. *Human movement science*, *10*(5), 575-587.

Della Croce, U., Leardini, A., Chiari, L., & Cappozzo, A. (2005). Human movement analysis using stereophotogrammetry: Part 4: assessment of anatomical landmark misplacement and its effects on joint kinematics. *Gait & Posture*, *21*(2), 226-237.

De Luca, R., Torrisi, M., Piccolo, A., Bonfiglio, G., Tomasello, P., Naro, A., Calabrò, R. S. (2017) Improving post-stroke cognitive and behavioral abnormalities by using virtual reality: A case report on a novel use of nirvana. *Applied Neuropsychology: Adult,* 1-5

Diebel, J. (2006) Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58, 1-35.

Doyle, J., Kelly, D., Patterson, M., Caulfield, B. (2011) The Effects of Visual Feedback in Therapeutic Exergaming on Motor Task Accuracy. *International Conference on Virtual Rehabilitation*, 1-5.

F

Fitzgerald, D., Trakarnratanakul, N., Smyth, B., Caulfield, B. (2010) Effects of a Wobble Board-Based Therapeutic Exergaming System for Balance Training on Dynamic Postural Stability and Intrinsic Motivation Levels. *Journal of Orthopaedic & Sports Physical Therapy*, 40(1), 11-19.

Fuller, J., Liu, L. J., Murphy, M. C., & Mann, R. W. (1997). A comparison of lower-extremity skeletal kinematics measured using skin-and pin-mounted markers. *Human Movement Science*, *16*(2), 219-242.

G

Giggings, O. M., Persson, U. M., Caulfield, B. (2013) Biofeedback in rehabilitation. *Journal of NeuroEngineering and Rehabilitation*, 10(1), 1-11.

Groh, F. (2012) Gamification: State of the Art Definition and Utilization. *Research Trends in Media Informatics,* 39-46.

H

Halstead, M. E. (2017) Elbow Dislocation [Internet]. [cited 2018 April 25]. Avaliable from: https://emedicine.medscape.com/article/96758-overview.

Hansson, G., Asterland, P., Holmer, N. G., & Skerfving, S. (2001). Validity and reliability of triaxial accelerometers for inclinometry in posture analysis. *Medical and Biological Engineering and Computing*, *39*(4), 405-413.

Hesse, S., & Uhlenbrock, D. (2000). A mechanized gait trainer for restoration of gait. *Journal of Rehabilitation Research and Development*, *37*(6), 701-708.

Hickey, D. G., Loebenberg, M. I. (2006). Elbow instability. *Bulletin of the NYU Hospital for Joint Diseases*, 64 (3-4), 166-171.

Howcroft, J., Klejman, S., Fehlings, D., Wright, V., Zabjek, K., Andrysek, J., Biddiss, E. (2012) Active video game play in children with cerebral palsy: Potential for physical activity promotion and rehabilitation therapies. *Archives of Physical Medicine and Rehabilitation*, 93(8), 1448-1456.

Hsu, S. H., Wen, M. H., Wu, M. C. (2009). Exploring user experiences as predictors of MMORPG addiction. *Computers and Education,* 53(3), 990-999.

K

Komatireddy, R., Chokshi, A., Basnett, J., Casale, M., Goble, D., Shubert, T. (2014) Quality and Quantity of Rehabilitation Exercises Delivered By A 3-D Motion Controlled Camera: A Pilot Study. *International Journal of Physical Medicine & Rehabilitation,* 2(4), 1-14.

Kuipers, J. (2000) Quaternions and Rotation Sequences. *Geometry, Integrability and Quantization*, 127-143.

Kurillo, G., Chen, A., Bajcsy, R., & Han, J. J. (2013a). Evaluation of upper extremity reachable workspace using Kinect camera. *Technology and Health Care*, *21*(6), 641-656.

Kurillo, G., Han, J. J., Obdrzálek, S., Yan, P., Abresch, R. T., Nicorici, A., & Bajcsy, R. (2013b). Upper extremity reachable workspace evaluation with Kinect. *MMVR*, 247-253.

L

Laver, K.E., George, S., Thomas, S., Deutsch, J.E., Crotty, M. (2011) Virtual reality for stroke rehabilitation. *The Cochrane Database of Systematic Reviews*.

Leardini, A., Chiari, L., Della Croce, U., & Cappozzo, A. (2005). Human movement analysis using stereophotogrammetry: Part 3. Soft tissue artifact assessment and compensation. *Gait & Posture*, *21*(2), 212-225.

Lee, J. C. (2008) Hacking the Nintendo Wii Remote. *IEEE C*, 39-45.

Li, G., & Buckle, P. (1999). Current techniques for assessing physical exposure to work-related musculoskeletal risks, with emphasis on posture-based methods. *Ergonomics*, *42*(5), 674-695.

M

Macedo, D. V., Rodrigues, M. A. F., Serpa, Y. R. (2015) Desenvolvimento de Aplicações Gráficas Interativas com a Unreal Engine 4. *Revista de Informática Teórica e Aplicada*, 22(2), 181-202.

Mader, S., Natkin, S., Levieux, G. (2012) How to analyse therapeutic games: The player/game/therapy model. *International Conference on Entertainment*, 193-206.

Maclean, N., Pound, P., Wolfe, C., Rudd, A. (2002) The concept of patient motivation: A qualitative of stroke professionals' attitudes. *Stroke,* 33(2), 444-448.

Mack, J., Stojsih, S., Sherman, D., Dau, N., Bir, C. (2010) Amateur Boxer Biomechanics and Punch Force. *28 International Conference on Biomechanics in Sports*, 2-5.

Marsh, T. (2011) Serious games continuum: Between games for purpose and experiential environments for purpose. *Entertainment Computing*, 2(2), 61–68.

Matallaoui, A., Koivisto, J., Hamari, J., Zarnekow, R. (2017) How Effective Is "Exergamification" A Systematic Review on the Effectiveness of Gamification Features in Exergames*. 50th Hawaii International Conference on System Sciences*, 3316-3325.

McKee, M.G. (2008) Biofeedback: an overview in the context of heart-brain medicine. *Cleveland Clinic Journal of Medicine,* 75(2) 31-35.

Menache, A. (2000). *Understanding motion capture for computer animation and video games*. Elsevier.

Merians, A. S., Jack, D., Boian, R., Tremaine, M., Burdea, G. C., Adamovich, S. V., Recce, M., Poizner, H. (2002). Virtual Reality – Augmented. *Physical Therapy*, 82(9), 899-915.

Moeslund, T. B., Hilton, A., & Krüger, V. (2006). A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, *104*(2), 90-126.

N

Norouzi-Gheidari, N., Levin, M. F., Fung, J., Archambault, P. (2013) Interactive virtual reality game-based rehabilitation for stroke patients. *2013 International Conference on Virtual Rehabilitation (ICVR)*, 220-221.

P

Perry, J. C., Rosen, J., & Burns, S. (2007). Upper-limb powered exoskeleton design. *IEEE/ASME transactions on mechatronics*, *12*(4), 408.

Petridis, P., Dunwell, I., Panzoli, D., Arnab, S., Protopsaltis, A., Hendrix, M., Freitas, S. (2012) Game Engines Selection Framework for High-Fidelity Serious Applications. *International Journal of Interactive Worlds.*

Piron, L., Tonin, P., Piccione, F., Iaia, V., Trivello, E., Dam, M. (2005). Virtual Environment Training Therapy for Arm Motor Rehabilitation. *Presence Teleoperators & Virtual Environments* 14(6):732-740

R

Rahimi, F., Duval, C., Jog, M., Bee, C., South, A., Jog, M., Edwards, R., Boissy, P. (2011) Capturing whole-body mobility of patients with Parkinson disease using inertial motion sensors: Expected challenges and rewards. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, 5833-5838.

Rasool, M. N. (2004). Dislocations of the elbow in children. *The Journal of Bone & Joint Surgery (Br)*, 86(7), 1050-1058.

Roetenberg, D., Luinge, H., Slycke, P. (2009) Xsens MVN: full 6DOF human motion tracking using miniature inertial sensors. *Xsens Motion Technologies BV*, 1-7.

Ross, G., McDevitt, E.R., Chronister, R. Ove, P. N. (1999) Treatment of Simple Elbow Dislocation Using an Immediate Motion Protocol. *The American Journal of Sports Medicine*, 27(3), 308-311.

Roy, A., Krebs, H. I., Williams, D. J., Bever, C. T., Forrester, L. W., Macko, R. M., Hogan, N. (2009) Robot-Aided Neurorehabilitation: A Novel Robot for Ankle Rehabilitation. *IEEE Trans Robotics*, 25(3), 569-582.

S

Salen, K., Zimmerman, E. (2004). Rules of Play: Game Design Fundamentals. *Cambridge: MIT Press*.

Schepers, H. M., Roetenberg, D., Veltink, P. H. (2010) Ambulatory human motion tracking by fusion of inertial and magnetic sensing with adaptive actuation. *Medical and Biological Engineering and Computing*, 48(1), 27-37.

Sinclair, J., Hingston, P., & Masek, M. (2007). Considerations for the design of exergames. *5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia.*

Skalski, A., Machura, B. (2015) Metrological analysis of microsoft kinect in the context of object localization. *Metrology and Measurement Systems*, 22(4), 469-478.

Sweetser, P., Wyeth, P. (2005) GameFlow : A Model for Evaluating Player Enjoyment in Games. *ACM Computers in Entertainment,* 3(3), 1-24.

T

Tanaka, K., Parker, J., Baradoy, G., Sheehan, D., Holash, J. R., Katz, L. (2012). A Comparison of Exergaming Interfaces for Use in Rehabilitation Programs and Research. *Loading*, 6(9), 69-81.

Tong, R. K. Y. (2014) Assisted technology for daily living. *SpringerPlus*, 3(1), 2-3.

Tong, R. K. Y., Hang, C. H., Chong, L. K. W., Lam, N. K. F. (2012) KineLabs 3D motion software platform using Kinect. *2012 International Conference on Computerized Healthcare (ICCH).*

Trenholme, D., Smith, S. P. (2014) Computer game engines for developing first-person virtual environments Computer game engines for developing first-person virtual environments. *Virtual Reality* ,12(3), 181-187.

W

Webster, D., Celik, O. (2014) Systematic review of Kinect applications in elderly care and stroke rehabilitation. *Journal of NeuroEngineering and Rehabilitation*, 11, 108-111.

Whittinghill, D. M., Brown, J. S. (2014) Gamification of physical therapy for the treatment of pediatric cerebral palsy: A pilot study examining player preferences. *ASEE Annual Conference and Exposition, Conference Proceedings*.

Winter, D. (2009) Biomechanics as an interdiscipline. *Biomechanics and Motor Control of Human Movement*, 10-11.

Wollersheim, D. (2010). Physical and Psychosocial Effects of Wii Video Game Use among Older Women. *International Journal of Emerging Technologies and Society*, 8(2), 85-98.

Z

Zhou, H., Hu, H. (2008) Human motion tracking for rehabilitation-A survey. *Biomedical Signal Processing and Control*, 3(1), 1-18.

Appendix

Appendix A. UDP Protocol provided by Biosyn.

UDP Definitions

To use the Realtime FAB System, frd2fab.exe must run on the computer that is connected to the FAB Belt Clip receiver. The frd2fab.exe program reads and interprets the raw data and then sends processed packets to a local UDP port. The standard UDP local port used is 2123.

The FAB system uses unique ID's for each sensor. The table below outlines the FAB sensor IDs:

Sensor ID	Sensor name/location on body
0	(Used by Belt Clip)
1	Left Arm Sensor
2	Right Arm Sensor
3	Upper Back Sensor
4	Lower Back Sensor
5	Left Leg Sensor
6	Right Leg Sensor
7	Left Wrist Sensor
8	Right Wrist Sensor
9	Left Calf Sensor
10	Right Calf Sensor
11	Head Sensor
128	Left Foot Sensor
129	Right Foot Sensor

There are 2 types of packets that frd2fab.exe sends over the UDP port; Header Packets and Data Packets. The packet types are distinguished by the length the packets.

Header Packets

Header packets are sent at the beginning of each session when frd2fab.exe are first turned on. Header packets are 24 bytes long and have the following format:

FAB File Format ID	1 byte	FAB File Format ID = 0x53. This is a file-format identifier.
FAB File Version	1 byte	11 is currently the latest version.
SensorIncludeFlags3D	2 bytes	16 bits indicating the clinician's sensor-inclusion list for the 3D sensors. This is a bitmask indicating which sensors should have been used for the session, set by the clinician with the Sensor Inclusion menu in the FAB GUI program or the corresponding Belt Clip menu. Bits 0 through 10 correspond to sensor IDs 1 through 11, respectively. A binary one indicates the corresponding sensor should have been used. A binary zero indicates the sensor was not required.
SensorIncludeFlagsFW	1 byte	8 bits indicating the clinician's sensor-inclusion list for the Foot Weight sensors. This is a bitmask indicating which sensors should have been used for the session, set by the clinician with the Sensor Inclusion menu in the FAB GUI program or the corresponding Belt Clip menu. Bits 0 through 1 correspond to sensor IDs 128 through 129, respectively. A binary one indicates the corresponding sensor should have been used. A binary zero indicates the sensor was not required.
Reserved	11 bytes	0 (for now)
FRDVersion	1 byte	The version number of the raw data file from which this processed data file was generated. Affects how data is interpreted.
StartDateTime	7 byte	seconds (8 bits), minutes (8 bits), hours (8 bits), day-of-month (8 bits), month (8 bits), year (16 bits)

Data Packets

The data packets contain the sensors angle and pressure data.

Flag1	1 byte	BIT7 = 0 for regular foot weight data, 1 to calibrate all weight on Right Foot BIT6 = 0 for regular foot weight data, 1 to calibrate all weight on Left Foot BIT5 = Reserved BIT4 = 0 for the first calibration position, 1 for the second calibration position BIT3 = 0 for regular data, 1 for calibration data BIT2 = 0 BIT1 = 0 BIT0 = MSB of Belt Clip Battery Voltage
Flag2	1 byte	lower 8 bits of Belt Clip Battery Voltage
TimeIncrem	1 byte	The number of sampling periods that have elapsed between the last record and this one. FRDVersions 0 through 2 sets this field to zero, in which case "1" should be assumed.
Reserved	1 byte	Unused
AbsoluteX	2 bytes	AbsoluteX/Y/Z and RelativeX/Y/Z values are signed 16-bit parameters, for future use in providing lateral movement and positioning information
AbsoluteY	2 bytes	
AbsoluteZ	2 bytes	
RelativeX	2 bytes	
RelativeY	2 bytes	
RelativeZ	2 bytes	
LowerBackHeading	2 bytes	All headings and angles are signed 16-bit numbers, representing degrees times 80 (e.g. +14400 represents +180.00 degrees); the range is -28800 to +28800 (-360.00 to +360.00 degrees).
LowerBackPitchAngle	2 bytes	
LowerBackRollAngle	2 bytes	
SpineRotationAngle	2 bytes	
SpinePitchAngle	2 bytes	
SpineRollAngle	2 bytes	
LeftArmAzimuthAngle	2 bytes	
LeftArmPolarAngle	2 bytes	
LeftWristAzimuthAngle	2 bytes	
LeftWristPolarAngle	2 bytes	
RightArmAzimuthAngle	2 bytes	
RightArmPolarAngle	2 bytes	
RightWristAzimuthAngle	2 bytes	
RightWristPolarAngle	2 bytes	
LeftLegAzimuthAngle	2 bytes	
LeftLegPolarAngle	2 bytes	
LeftCalfAzimuthAngle	2 bytes	
LeftCalfPolarAngle	2 bytes	
RightLegAzimuthAngle	2 bytes	
RightLegPolarAngle	2 bytes	
RightCalfAzimuthAngle	2 bytes	
RightCalfPolarAngle	2 bytes	

LeftFootSensor	2 bytes	
LFReserved1	2 bytes	
SensorErrorFlags	2 bytes	
RightFootSensor	2 bytes	
RFReserved1	2 bytes	
SensorLowBatteryFlags	2 bytes	
LeftArmYawAngle	2 bytes	
LeftArmPitchAngle	2 bytes	
LeftArmRollAngle	2 bytes	
LeftWristYawAngle	2 bytes	
LeftWristPitchAngle	2 bytes	
LeftWristRollAngle	2 bytes	
RightArmYawAngle	2 bytes	
RightArmPitchAngle	2 bytes	
RightArmRollAngle	2 bytes	
RightWristYawAngle	2 bytes	
RightWristPitchAngle	2 bytes	
RightWristRollAngle	2 bytes	
LeftLegYawAngle	2 bytes	
LeftLegPitchAngle	2 bytes	
LeftLegRollAngle	2 bytes	
LeftCalfYawAngle	2 bytes	
LeftCalfPitchAngle	2 bytes	
LeftCalfRollAngle	2 bytes	
RightLegYawAngle	2 bytes	
RightLegPitchAngle	2 bytes	
RightLegRollAngle	2 bytes	
RightCalfYawAngle	2 bytes	
RightCalfPitchAngle	2 bytes	
RightCalfRollAngle	2 bytes	
HeadYawAngle	2 bytes	
HeadPitchAngle	2 bytes	
HeadRollAngle	2 bytes	

Appendix B. Informed Consent Form

Declaração de Consentimento Informado, Esclarecido e Livre

Entidades Promotoras: Instituto Superior Técnico (IST) e Hospital Dona Estefânia – Centro Hospitalar de Lisboa Central (CHLC).

Investigadores Responsáveis pelo Estudo: Sara Margarida Almeida Alberto (Instituto Superior Técnico, Universidade de Lisboa); Miguel Tavares da Silva (LAETA, IDMEC, Instituto Superior Técnico, Universidade de Lisboa

Nome do Estudo: Desenvolvimento de um Sistema de Biofeedback para Apoio à Reabilitação Motora

Deverá ler atentamente toda a informação aqui contida. Caso surja alguma dúvida, deve sentir-se livre para colocar qualquer questão, assim como discutir com terceiros a decisão de participação neste estudo.

Objectivos e Descrição do Estudo

O presente estudo tem como principal objetivo o desenvolvimento de um *software* para apoio à reabilitação motora que utiliza um conjunto de unidades de medição inercial (IMUs) para adquirir o movimento de um utilizador, sendo esse movimento processado em tempo real e integrado no âmbito de uma aplicação computacional desenvolvida em ambiente Unity. Esta aplicação permite fornecer ao utilizador um retorno visual do movimento realizado e o consequente enquadramento do desempenho obtido num conjunto de objetivos a atingir. Como resultado, espera-se alcançar com este tipo de tecnologia, frequentemente designada por "*gamification"*, um maior empenho e interesse do utilizador no processo de reabilitação e consequentemente melhores resultados. Nesta fase do estudo, pretende-se simular sessões de reabilitação utilizando o *software* desenvolvido, de forma a ajustar parâmetros de utilização à faixa etária alvo. Os dados recolhidos serão igualmente armazenados nas bases de dados do Laboratório de Biomecânica de Lisboa, podendo ser utilizados em futuros estudos relacionados com esta temática

O presente estudo encontra-se inserido numa tese de mestrado de Engenharia Biomédica.

Detalhes do Estudo

Duração esperada para a sua participação: Cada sessão terá uma duração de aproximadamente meia hora, podendo o número de sessões variar entre 3 e 5.

Procedimentos a serem realizados: O estudo consistirá na simulação de sessões de reabilitação utilizando o *software* desenvolvido, sendo pedido *feedback* aos participantes de modo a ajustar parâmetros de utilização. A aquisição da cinemática fará uso de um sistemas MOCAP inercial (F.A.B.^{TM} Systems – BIOSYN SYSTEMS).

Participação voluntária: A sua participação é voluntária e pode recusar-se a participar ou retirar-se a qualquer momento, sem qualquer tipo de consequências para si. No caso de decidir abandonar o estudo, a sua relação com o Instituto Superior Técnico não será afectada.

Benefícios da participação: O presente estudo não contempla uma compensação directa para o voluntário. Contudo, os resultados obtidos serão usados para o desenvolvimento e aperfeiçoamento das metodologias utilizadas, podendo vir a beneficiá-lo ou a outros no futuro. Os dados serão igualmente usados para validação e aperfeiçoamento dos modelos computacionais desenvolvidos no Instituto Superior Técnico.

Riscos e complicações do exame: A realização do exame não contém riscos para si nem para a sua saúde. O procedimento de aquisição de movimento é um método não invasivo, que considera a utilização de um sistema MOCAP inercial (F.A.B.^{TM} Systems – BIOSYN SYSTEMS) que faz uso de sensores inerciais *wireless*, usualmente conhecidos como IMUs, colocados sobre a pele ou roupa através de bandas elásticas. Estes dispositivos medem acelerações, velocidades angulares e desvios ao campo magnético terrestre, não causando desta forma lesões no utilizador. Não serão feitos estudos que requeiram dispêndios energéticos significativos.

Confidencialidade dos dados e anonimato: Apenas os responsáveis e investigadores participantes no estudo terão acesso detalhado aos dados recolhidos. Após término do estudo, os dados serão arquivados e a confidencialidade dos mesmos, assim como o anonimato, garantido.

Divulgação dos dados: A confidencialidade do paciente estará sempre salvaguardada, sendo que apenas dados de carácter científico poderão ser divulgados. Não serão apresentadas nem partilhadas informações de cariz pessoal e imagens nas quais seja possível identificar o voluntário. Será esperado apresentar os resultados do estudo em seminários, conferências e revistas científicas da especialidade.

Esclarecimentos: Para qualquer questão relacionada com a sua participação, não hesite em contactar o investigador responsável pelo estudo:

- Sara Margarida Almeida Alberto – sara.a.alberto@tecnico.ulisboa.pt;
- Miguel Tavares da Silva – MiguelSilva@tecnico.ulisboa.pt;

Autorização para Participação no Ensaio

Li ou foi-me lido o presente documento e estou consciente do que esperar quanto à minha participação no estudo acima mencionado. Tive a oportunidade de colocar todas as questões que pretendi e as repostas esclareceram todas as minhas dúvidas. Assim, declaro que aceito voluntariamente participar neste estudo. Informo igualmente que foi dada, a mim ou ao meu representante, uma cópia deste documento devidamente assinado por um dos responsáveis pelo estudo.

Nome Participante: _____

Nome do Representante Legal (caso aplicável)*: _____

_____

Data:__/__/____

*Responsável deverá aceitar o consentimento no caso de o participante ser menor

Nome do Investigador:_____

_____

Data:__/__/____

Appendix C. Code

Database and Scene Manager

```
using UnityEngine;
using System;
using UnityEngine.UI;
using System.Collections;
using DatabaseControl; // << Include the DatabaseControl namespace
using System.Collections.Generic;
using System.Text.RegularExpressions;
using UnityEngine.EventSystems;
using System.Linq;

public class LoginDemo_Accounts : MonoBehaviour {

    //Database
    string databaseName = "";
    bool canRunSequences = false;

    //Camera
    Camera mainCam;

    //Parents
    public GameObject CanvasParent;
    public GameObject TheParent;

    //Login
    public GameObject loginParent;
    public InputField Login_UsernameField;
    public InputField Login_PasswordField;
    public Text Login_Error;
    string username = "";
    string password = "";
    string loginError = "";

    //Register
    public GameObject registerParent;
    public InputField Register_UsernameField;
    public InputField Register_PasswordField;
    public InputField Register_ConfirmPasswordField;
    public Text Register_Error;
    string registerError = "";
    public InputField Register_Bday;
    public InputField Register_Bmonth;
    public InputField Register_Byear;
    public ToggleGroup Register_Dominancetoggles;
    string Register_Dominance;

    //Loading
    public GameObject loadingParent;

    //ContinueNewSeeData
    public GameObject ContinueNewSeedata;
    public Dropdown New_Artic;
    public Dropdown New_Pato;
    public Text New_LoggedInAs;
    public List<List<string>> Patologias = new List<List<string>>();
    public List<string> Ombro = new List<string> { "a", "b" };
    public List<string> Cotovelo = new List<string> { "luxação",
"outros" };
    public List<string> Joelho = new List<string>();
    string artic;
    string pato;

    //Cotovelo
    public string ChosenGame;

    //SphereShieldSettings
    public InputField level1;
    public InputField level2;
    public InputField level3;
    float speed;
    string side;
    public string Arm;

    //IMUCalib
    public Text Countdown;
    public Text IMUs;
    public bool starttimer = false;
    float timeLeft = 20.0f;
    public List<NamedList<string>> AllGames = new
List<NamedList<string>>();
    public NamedList<string> SphereShield = new
NamedList<string>("SphereShield") { "Pélvis", "Tronco", "Braço
Esquerdo", "Antebraço Esquerdo", "Braço Direito", "Antebraço Direito"
};
```

```
//GameCanvas
public GameObject GameCanvas;
string satisfaction;
string pain;
public Text TheEnd;

//PatientData
public GameObject DataParent;
public Text generalData;
public GameObject buttonPrefab;
List<float> maxflexs = new List<float>();
public float[] angov;
public string maxflexsstring;
public Text Dados;
public Text Dados2;
int numSessions;
int requestedSession;
string date;
string gameop;
public LineChart line1;
int linenum = 0;
public Dropdown GamesOverview;
List<float> maxexts = new List<float>();
public string maxextsstring;
public LineChart line2;
public int linenum2;
public float[] stringtofloat;
public string[] temp;
float[] Leveltimes = new float[3];
string[] timesString = new string[3];

//Game
public int noEnemies;
public GameObject Avatart;
public Text ScoreEsquerdo;
public Text ScoreDireito;
public Button experimentOver;
string scoreE;
string scoreD;
public int CurLevel;

void Start ()
{

          mainCam = Camera.main;
    //Gets the databaseName as it was setup through the editor
        GameObject linkObj = GameObject.Find("Link");
        if (linkObj == null)
        {
            Debug.LogError("DCP Error: Cannot find the link object in
the scene so scripts running Command Sequences don't know the
database name");
        } else
        {
            DCP_Demos_LinkDatabaseName linkScript =
linkObj.gameObject.GetComponent<DCP_Demos_LinkDatabaseName>() as
DCP_Demos_LinkDatabaseName;
            if (linkScript == null)
            {
                Debug.LogError("DCP Error: Cannot find the link
script on link object so scripts running Command Sequences don't know
the database name");
            } else
            {
                linkScript.databaseName = "final";
                if (linkScript.databaseName == "")
                {
                    Debug.LogError("DCP Error: This demo scene has
not been setup. Please setup the demo scene in the Setup window
before use. Widnow>Database Control Pro>Setup Window");
                } else
                {
                    databaseName = linkScript.databaseName;
                    canRunSequences = true;
                }
            }
        }

        GameCanvas.gameObject.SetActive(false);
        ResetAllUIElements();
        AllGames.Add(SphereShield);
        Patologias.Add(Ombro);
        Patologias.Add(Cotovelo);
        Patologias.Add(Joelho);
        New_Pato.AddOptions(Patologias[New_Artic.value]);

    }

    public float fps;
    void Update ()
    {
        fps = 1.0f / Time.deltaTime;
```

```
        if (FindObject(TheParent,
"PatientData").gameObject.activeSelf == true)
        {
            line1.UpdateData(angov);
        }

               Pause();

        //Writes the login and register errors to the ui texts
        Login_Error.text = loginError;
        Register_Error.text = registerError;
        New_LoggedInAs.text = "Logged in as: " + username;

        if (starttimer == true)
        {
            timeLeft -= Time.deltaTime;
            Countdown.text = "A calibrar... " + (int)timeLeft;
            if (timeLeft < 0)
            {
                CanvasParent.gameObject.SetActive(false);
                ChangeActive(ChosenGame);
                Time.timeScale = 1;
                starttimer = false;
                timeLeft = 20f;
                Countdown.text = "A calibrar... " + (int)timeLeft;

            }
        }

        if (ChosenGame!= "" )
        {

            if(FindObject(TheParent,
ChosenGame).gameObject.activeSelf == true && Time.timeScale == 1)
            {
                Leveltimes[CurLevel - 1] += Time.deltaTime;
            }
        }

    }



    #region button pressed methods
    //Called when login button is pressed
    public void LoginStuff_LoginButtonPressed ()
    {
        if (Login_UsernameField.text.Length < 6)
        {
            //show login error as the username is too short
            loginError = "Username too short";
            return;
        }
        if (Login_PasswordField.text.Length < 4)
        {
            //show login error as the password is too short
            loginError = "Password too short";
            return;
        }
        if (canRunSequences == true) // << Makes sure the demo scene
has been setup correctly so this script knows the database name
        {

            username = Login_UsernameField.text;
            password = Login_PasswordField.text;
            loginParent.gameObject.SetActive(false);
            loadingParent.gameObject.SetActive(true);
            Login_UsernameField.text = "";
            Login_PasswordField.text = "";
            StartCoroutine(Login()); // << Runs the IEnumerator which
runs the command sequence
            StartCoroutine(NoOfSessions());
        }
    }

    //Called when the register button is pressed
    public void RegisterStuff_RegisterButtonPressed ()
    {
        if (Register_UsernameField.text.Length < 6)
        {
            //show register error as the username is too short
            registerError = "Username too short";
            return;
        }
        if (Register_PasswordField.text.Length < 4)
        {
            //show register error as the password is too short
            registerError = "Password too short";
        }
        if (Register_PasswordField.text !=
Register_ConfirmPasswordField.text)
        {
```

```
            //show register error as the passwords dont match
            registerError = "Passwords don't match";
        }
        if (canRunSequences == true) // << Makes sure the demo scene
has been setup correctly so this script knows the database name
        {

                        registerParent.gameObject.SetActive(false);
                        loadingParent.gameObject.SetActive (true);
            //registerParent.gameObject.SetActive(true);
            username = Register_UsernameField.text;
            password = Register_PasswordField.text;
            Register_UsernameField.text = "";
            Register_PasswordField.text = "";
            Register_ConfirmPasswordField.text = "";
            StartCoroutine(Register()); // << Runs the IEnumerator
which runs the command sequence
        }
    }

    //Called when the set data button is pressed
    public void Data_SetDataButtonPressed ()
    {
        loadingParent.transform.parent.gameObject.SetActive(true);
        loadingParent.gameObject.SetActive(true);

        for (int i = 0; i < Leveltimes.Length; i++)
        {
            float minutes = Mathf.Floor(Leveltimes[i] / 60);
            float seconds = Leveltimes[i] % 60;
            timesString[i] = Mathf.RoundToInt(minutes) + "min" +" " +
Mathf.RoundToInt(seconds) + "s";
        }
              numSessions++;
        StartCoroutine(SetData());
    }

    #endregion

    #region Run Command Sequences
    //Run the command sequence to login a user
    IEnumerator Login ()
    {
        //Run the command sequence called 'Login' on the database
name which has been retrieved in the start method. Sends the username
and password (from the ui input fields) as parameters
        IEnumerator e = DCP.RunCS(databaseName, "Login", new
string[2] { username, password });

        //wait for the command sequence to finish loading
        while (e.MoveNext())
        {
            yield return e.Current;
        }

        //get the command sequence result
        string returnText = e.Current as string;

               if (returnText != "UserError" && returnText!=
"PassError")
        {
                        returnText = returnText.Replace('$', '\n');
                        generalData.text = returnText;

            ResetAllUIElements();
            loadingParent.gameObject.SetActive(false);
            ContinueNewSeedata.gameObject.SetActive(true);

            //Login was successful
            loginError = ""; // << make ui text for login error
disappear

        } else
        {
            loadingParent.gameObject.SetActive(false);
            loginParent.gameObject.SetActive(true);
            if (returnText == "UserError")
            {
                //The username does not exist
                loginError = "Utilizador não encontrado";
            } else
            {
                if (returnText == "PassError")
                {
                    //The username was found but the password was
incorrect
                    loginError = "Password incorreta";
                } else
                {
                    //Some other error which is unlikely to happen
but should be considered if it does
```

```
                    loginError = "Problem with server. Try again
later.";
                }
            }

        }
    }

    //Run the command sequence to register a user
    IEnumerator Register ()
    {
        string bdate = Register_Bday.text + "/" +
Register_Bmonth.text + "/" + Register_Byear.text;
        Toggle[] alltoggles =
Register_Dominancetoggles.GetComponentsInChildren<Toggle>();
        for (int i = 0; i < alltoggles.Length; i++)
        {
            if (alltoggles[i].isOn)
            {
                Register_Dominance = alltoggles[i].name;
            }
        }

        //Run the command sequence called 'Register' on the database
name which has been retrieved in the start method. Sends the username
and password (from the ui input fields) as parameters
        IEnumerator e = DCP.RunCS(databaseName, "Register", new
string[4] { username, password, bdate, Register_Dominance });

        //wait for the command sequence to finish loading
        while (e.MoveNext())
        {
            yield return e.Current;
        }

        //get the command sequence result
        string returnText = e.Current as string;
        returnText = returnText.Replace('$', '\n');
        generalData.text = returnText;


        if (returnText != "username in use")
        {
            ResetAllUIElements();
            loadingParent.gameObject.SetActive(false);
            registerParent.gameObject.SetActive(false);
            ContinueNewSeedata.gameObject.SetActive(true);
            //Registration was successful
            registerError = ""; // << make ui text for register error
disappear

        } else
        {
            loadingParent.gameObject.SetActive(false);
            registerParent.gameObject.SetActive(true);
            if (returnText == "username in use")
            {
                //The username is already in use
                registerError = "Nome de utilizador já em uso";
            } else
            {
                //Some other error which is unlikely to happen but
should be considered if it does
                registerError = "Problem with server. Try again
later.";
            }
        }
    }

        IEnumerator NoOfSessions()
        {
                IEnumerator e = DCP.RunCS(databaseName,
"NoOfSessions", new string[2] {username, password});
                while (e.MoveNext())
        {
            yield return e.Current;
        }
                string returnText = e.Current as string;
         numSessions = int.Parse(returnText);
        }

        IEnumerator Datas(int requested)
        {
                string reqstring = requested.ToString ();
                IEnumerator e = DCP.RunCS (databaseName, "Datas", new
string[3] { username, password, reqstring});

                while (e.MoveNext())
                {
                        yield return e.Current;
                }
```

```
            //get the command sequence result
            string returnText = e.Current as string;

            if (returnText != "Error")
            {
                date = returnText;
            }
        }

    IEnumerator GetOverview(int row, int col)
    {
        string gamename =
GamesOverview.options[GamesOverview.value].text;
        string colstring = col.ToString();
            string rowstring = row.ToString();
        IEnumerator e = DCP.RunCS(databaseName, "GetOverview", new
string[5] { username, password, rowstring, colstring, gamename });
        while (e.MoveNext())
        {
                yield return new WaitForSeconds(5);
                yield return e.Current;
        }
        string returnText = e.Current as string;
        angov[row] = float.Parse(returnText);
            GameObject gameopt = FindObject(DataParent, gamename);
            gameopt.gameObject.SetActive (true);
        loadingParent.transform.parent.gameObject.SetActive(false);
        loadingParent.gameObject.SetActive(false);
    }

       public void overviewButton(int col)
    {
        angov = new float[numSessions];
        for (int i = 0; i < numSessions ; i++)
            {
                StartCoroutine(GetOverview(i, col));
            }
        line1.Chart(angov, linenum);
        linenum++;
        line1.gameObject.transform.parent.gameObject.SetActive(true);

    }

       public List<string> gameoptions = new List<string> ();
       IEnumerator GetGame(int row)
       {
            string rowstring = row.ToString();
            IEnumerator e = DCP.RunCS(databaseName, "GetGame", new
string[3] { username, password, rowstring });
            while (e.MoveNext())
            {
                yield return e.Current;
            }

            string returnText = e.Current as string;

            if (returnText != "Error")
            {
                gameop = returnText;
            }
        loadingParent.gameObject.SetActive(false);
       }

       public IEnumerator gameOptions()
       {
        for (int i = 0; i< numSessions; i++)
        {
            yield return StartCoroutine(GetGame(i));

            gameoptions.Insert(i, gameop);
            gameoptions = gameoptions.Distinct().ToList();

            GamesOverview.AddOptions(gameoptions);
        }
    }

       public void dropdownGames()
       {
        StartCoroutine(gameOptions());
       }

    //Run the command sequence to set user data
    IEnumerator SetData()
    {
            setSessionFlex ();
            setSessionExt ();
        int tot = Int32.Parse(ScoreDireito.text) +
Int32.Parse(ScoreEsquerdo.text);
        string percentE;
        string percentD;
        if (tot != 0)
        {
```

```
            //quero uma ou duas casas decimais
            percentE = string.Format("{0:0.0}",
((Int32.Parse(ScoreEsquerdo.text) * 100) / tot).ToString());
            percentD = string.Format("{0:0.0}",
((Int32.Parse(ScoreDireito.text) * 100) / tot).ToString());

        }
        else
        {
            percentE = "0";
            percentD = "0";
        }
        float totaltime = Leveltimes[0] + Leveltimes[1] +
Leveltimes[2];
        string avg = string.Format("{0:0.0}", (totaltime /
(float)tot).ToString());
        string speed = PlayerPrefs.GetFloat("Velocidade
inicial").ToString();
                string minWait = PlayerPrefs.GetFloat("Cadência
mínima").ToString();
        scoreD = ScoreDireito.text;
        scoreE = ScoreEsquerdo.text;
        artic = PlayerPrefs.GetString("Artic");
        pato = PlayerPrefs.GetString("Pato");
        side = PlayerPrefs.GetString("ArmKey");
               string lives1 = PlayerPrefs.GetInt ("1").ToString();
               string lives2 = PlayerPrefs.GetInt ("2").ToString();
               string lives3 = PlayerPrefs.GetInt ("3").ToString();
        string extString = PlayerPrefs.GetFloat("MaxExt").ToString();
        string flexString =
PlayerPrefs.GetFloat("MaxFlex").ToString();
        satisfaction =
Mathf.RoundToInt(PlayerPrefs.GetFloat("Satisfaçao")).ToString();
        pain =
Mathf.RoundToInt(PlayerPrefs.GetFloat("Dor")).ToString();

        //Run the command sequence called 'Set Data' on the database
name which has been retrieved in the start method. Sends the
username, password and data string (from the ui input fields) as
parameters
               IEnumerator e = DCP.RunCS(databaseName, "SetSession",
new string[23] { username, password, artic, pato, side, ChosenGame,
percentE, percentD, lives1, timesString[0], lives2, timesString[1],
lives3, timesString[2], extString, pain, satisfaction, speed,
minWait, maxextsstring, flexString, maxflexsstring, avg});
        while (e.MoveNext())
        {
            yield return e.Current;
        }

        //get the command sequence result
        string returnText = e.Current as string;

        if (returnText != "Error")
        {

            //set data was successful
loadingParent.transform.parent.gameObject.SetActive(false);
            loadingParent.gameObject.SetActive(false);

        }
        else
        {
            ResetAllUIElements();
            username = "";
            password = "";
            loginParent.gameObject.SetActive(true);
            loadingParent.gameObject.SetActive(false);
            Login_Error.text = "Error: Unknown Error. Please try
again later.";
        }
    }

    //Run the command sequence to get user data
       IEnumerator GetData(/*int requested*/)
    {
               string butt = PlayerPrefs.GetString("datadobotao");
               string[] yearmonthday = butt.Split (new string[] { "/"
}, StringSplitOptions.RemoveEmptyEntries);
             //Run the command sequence called 'Get Data' on the
database name which has been retrieved in the start method. Sends the
username and password (from the ui input fields) as parameters
              IEnumerator e = DCP.RunCS(databaseName, "NewGetData",
new string[5] { username, password, yearmonthday[0], yearmonthday[1],
yearmonthday[2]});

        //wait for the command sequence to finish loading
        while (e.MoveNext())
        {
            yield return e.Current;
```

```
        }

        //get the command sequence result
        string returnText = e.Current as string;
               returnText = returnText.Replace('$','\n');
       Dados.text = returnText;
        IEnumerator b = DCP.RunCS(databaseName, "GetSession", new
string[5] { username, password, /*reqstring*/yearmonthday[0],
yearmonthday[1], yearmonthday[2] });

        //wait for the command sequence to finish loading
        while (b.MoveNext())
        {
            yield return b.Current;
        }
        string returnText2 = b.Current as string;
        returnText2 = returnText2.Replace('$', '\n');
        Dados2.text = returnText2;


    }

    public void ButtonFunction()  //Passa a ser só coroutine GetData
    {
        StartCoroutine(GetData());
    }
    #endregion



    #region My Stuff

    //Called by Button Pressed Methods to Reset UI Fields
    void ResetAllUIElements()
    {
        //This resets all of the UI elements. It clears all the
strings in the input fields and any errors being displayed
        Login_UsernameField.text = "";
        Login_PasswordField.text = "";
        Register_UsernameField.text = "";
        Register_PasswordField.text = "";
        Register_ConfirmPasswordField.text = "";
        Login_Error.text = "";
        Register_Error.text = "";
        New_LoggedInAs.text = "";
    }
```

```
    private void Pause()
    {
                if (Input.GetKeyDown(KeyCode.P) &&
CanvasParent.activeInHierarchy == false)
        {
            if (Time.timeScale == 1)
            {

                Time.timeScale = 0;
                GameCanvas.gameObject.SetActive(true);
            }
            else if (Time.timeScale == 0)
            {

                Time.timeScale = 1;
                GameCanvas.gameObject.SetActive(false);
            }
        }
    }

    public void ExitGame()
    {
        Derp();
        Application.Quit();
    }

    public void LogOut()
    {
        ResetAllUIElements();
        username = "";
        password = "";
        Derp();

EventSystem.current.currentSelectedGameObject.transform.parent.gameOb
ject.SetActive(false);
        CanvasParent.gameObject.SetActive(true);
        loginParent.gameObject.SetActive(true);
    }

    public static GameObject FindObject(GameObject parent, string
name)
    {
        Transform[] trs =
parent.GetComponentsInChildren<Transform>(true);
        foreach (Transform t in trs)
```

```
        {
            if (t.name == name)
            {
                return t.gameObject;
            }
        }
        return null;
    }

    public void ChangeActive(string newscene)
    {
        GameObject change = FindObject(TheParent, newscene);

EventSystem.current.currentSelectedGameObject.transform.parent.gameOb
ject.SetActive(false);
        change.transform.parent.gameObject.SetActive(true);
        change.gameObject.SetActive(true);
    }

    public void BackToChoose()
    {
        GameObject choosescene = FindObject(TheParent,
PlayerPrefs.GetString("Artic"));


EventSystem.current.currentSelectedGameObject.transform.parent.gameOb
ject.SetActive(false);
        choosescene.transform.parent.gameObject.SetActive(true);
        choosescene.gameObject.SetActive(true);
    }

    public void OutOfGame()
    {
        Time.timeScale = 1;
              GameObject robotGroup = FindObject(TheParent,
CurLevel.ToString());
              robotGroup.gameObject.SetActive(false);
        GameObject CurrentGame = FindObject(TheParent, ChosenGame);
        CurrentGame.gameObject.SetActive(false);
    }

    public void chosenGame()
    {
        ChosenGame =
EventSystem.current.currentSelectedGameObject.name;
    }

    public void ChangeActiveIfPrefs(string calib)
    {

        if (PlayerPrefs.HasKey("Level") &&
PlayerPrefs.HasKey("ArmKey") && level1.text!= null && level2.text !=
null && level3 != null)
        {
            GameObject change = FindObject(TheParent, calib);

EventSystem.current.currentSelectedGameObject.transform.parent.gameOb
ject.SetActive(false);
            change.transform.parent.gameObject.SetActive(true);
            change.gameObject.SetActive(true);
        }

        for (int i = 0; i <= AllGames.Count; i++)
        {
            if (ChosenGame == AllGames[i].Name)
            {
                if (IMUs.text == "")
                {
                    for (int j = 0; j <= AllGames[i].Count; j++)
                    {
                        IMUs.text += "\n" + AllGames[i][j];
                    }
                }
            }
    }

    public void StartCalibrating()
    {
        starttimer = !starttimer;
        Countdown.text = "A calibrar... " + (int)timeLeft;
    }


    public void GetArm()
    {
        Arm = EventSystem.current.currentSelectedGameObject.name;
        PlayerPrefs.SetString("ArmKey", Arm);
        PlayerPrefs.Save();
    }

    public void Level(int level)
```

```
    {
        PlayerPrefs.SetInt("Level", level);
        PlayerPrefs.Save();
        CurLevel = level;

    }

    public void Derp()
    {
        PlayerPrefs.DeleteAll();
    }

    public void changeDropDown()
    {
        New_Pato.options.Clear();
        New_Pato.AddOptions(Patologias[New_Artic.value]);
    }

    public void chooseFromDropDownNew()
    {
        string ArticName = New_Artic.options[New_Artic.value].text;
        string PatoName = New_Pato.options[New_Pato.value].text;
        GameObject change = FindObject(TheParent, ArticName);
        ContinueNewSeedata.gameObject.SetActive(false);
        PlayerPrefs.SetString("Artic", ArticName);
        PlayerPrefs.SetString("Pato", PatoName);
        PlayerPrefs.Save();
        change.gameObject.SetActive(true);
    }


    public void setFloat()
    {
        string namestring =
EventSystem.current.currentSelectedGameObject.name;
        PlayerPrefs.SetFloat(namestring,
EventSystem.current.currentSelectedGameObject.GetComponent<Slider>().
value);
        PlayerPrefs.Save();
        string name =
EventSystem.current.currentSelectedGameObject.name;
        Text textthing =
EventSystem.current.currentSelectedGameObject.GetComponentInChildren<
Text>();
        if (textthing != null)
        {
            textthing.text = string.Format("{0:0.0}",
EventSystem.current.currentSelectedGameObject.GetComponent<Slider>().
value);
        }
    }

    public void setLives()
    {
        PlayerPrefs.SetInt("1", Int32.Parse(level1.text));
        PlayerPrefs.SetInt("2", Int32.Parse(level2.text));
        PlayerPrefs.SetInt("3", Int32.Parse(level3.text));
        PlayerPrefs.Save();
    }

    public void setLevel()
    {
            GameObject robotGroup = FindObject(TheParent,
CurLevel.ToString());
        robotGroup.gameObject.SetActive(true);
        GameObject[] noRobots =
GameObject.FindGameObjectsWithTag("Enemy");
        noEnemies = noRobots.Length;
    }

    public void EnemyDown()
    {
        noEnemies--;
        speed += 0.2f;
        if (noEnemies == 0 && CurLevel < 3)
        {
            CurLevel++;
            Level(CurLevel);
            setLevel();
        }
        else if (noEnemies == 0 && (CurLevel == 3 ||
PlayerPrefs.GetInt("Level") == 3))
        {
            Time.timeScale = 0;
            GameCanvas.gameObject.SetActive(true);
            TheEnd.text = "Fim de sessão";
        }
    }

    public IEnumerator ButtonCreate(int i)
    {
```

```
        yield return StartCoroutine(Datas(i));
               GameObject newbutton =
(GameObject)Instantiate(buttonPrefab,FindObject(DataParent,
"ButtonPanel").transform, false);
        var rt = newbutton.GetComponent<RectTransform>();

        float height = rt.rect.height;
               rt.localPosition = new Vector3(0, 222.1f - (i *
height), 0);
               newbutton.GetComponentInChildren<Text>().text = date;
    }

    public IEnumerator ButtonDate(int i)
    {
        yield return StartCoroutine(Datas(i));
    }

    public void createButtons()
        {
        if (canRunSequences == true)
        {

EventSystem.current.currentSelectedGameObject.transform.parent.gameOb
ject.SetActive(false);
            CanvasParent.gameObject.SetActive(true);
            loadingParent.gameObject.SetActive(true);


            for (int i = 0; i < numSessions; i++)
            {
                StartCoroutine(ButtonCreate(i));
            }

                    CanvasParent.gameObject.SetActive(false);
                    loadingParent.gameObject.SetActive(false);
                    DataParent.gameObject.SetActive(true);
        }
    }

       public void EndExperiment()
       {
              Vector3 newpos = new Vector3 ();

EventSystem.current.currentSelectedGameObject.gameObject.SetActive(fa
lse);
               newpos.x = 11.47f;
               newpos.y = 12.03f;
               newpos.z = 30.63f;
               Avatart.GetComponent<SkinnedMeshRenderer> ().enabled =
false;

               mainCam.transform.position = newpos;
               setLevel ();
       }

    public void BacktoGame()
    {
        Vector3 newpos = new Vector3();
        newpos.x = 9.62f;
        newpos.y = 12.03f;
        newpos.z = 28.57f;
        mainCam.transform.position = newpos;
        experimentOver.gameObject.SetActive(true);
        Avatart.GetComponent<SkinnedMeshRenderer>().enabled = true;
    }

       public void setSessionExt()
       {
              GameObject manager =
GameObject.FindGameObjectWithTag("Player");
               var udpscript = manager.GetComponent<UDPReceive>();
               List<float> angles = udpscript.getextEvolution();
               int j = 0;
               int k = 0;
               for(int i = 0; i< angles.Count; i++)
               {
                      j = i + 1;
                      k = i + 2;
                      if (k < angles.Count)
                      {
                             if (angles[i] >angles[j] && angles[k] >
angles[j])
                             {
                                    maxexts.Add(angles[j]);
                             }
                      }

                      else if (j < angles.Count)
                      {
                             if (angles[i] > angles[j])
                             {
                                    maxexts.Add(angles[j]);
```

```
                        }
                }
        }

        for(int u = 0; u<maxexts.Count; u++)
        {
                maxextsstring += maxexts[u] + "$";
        }
    }

    public void setSessionFlex()
    {
        GameObject manager =
GameObject.FindGameObjectWithTag("Player");
        var udpscript = manager.GetComponent<UDPReceive>();
        List<float> angles = udpscript.getflexEvolution();
        int j = 0;
        int k = 0;
        for(int i = 0; i< angles.Count; i++)
        {
                j = i + 1;
                k = i + 2;
                if (k < angles.Count)
                {
                        if (angles[i] < angles[j] && angles[k]
< angles[j])
                        {
                                maxflexs.Add(angles[j]);
                        }
                }

                else if (j < angles.Count)
                {
                        if (angles[i] < angles[j])
                        {
                                maxflexs.Add(angles[j]);
                        }
                }
        }

        for(int u = 0; u<maxflexs.Count; u++)
        {
                maxflexsstring += maxflexs[u] + "$";
        }
    }
```

```
    public IEnumerator getSessionAngles(int col)
    {
        string butt = PlayerPrefs.GetString("datadobotao");
        string[] yearmonthday = butt.Split(new string[] { "/" },
StringSplitOptions.RemoveEmptyEntries);
        IEnumerator e = DCP.RunCS(databaseName, "GetAngles", new
string[6] { username, password, yearmonthday[0], yearmonthday[1],
yearmonthday[2], col.ToString() });
        while (e.MoveNext())
        {
            yield return e.Current;
        }

        string returnText = e.Current as string;
        temp = returnText.Split(new string[] { "$" },
StringSplitOptions.RemoveEmptyEntries);
        angov = Array.ConvertAll<string, float>(temp, float.Parse);
    }

    public IEnumerator AnglesAux (int col)
    {
        angov = new float[9000];
        yield return StartCoroutine(getSessionAngles(col));
    }

    //ver os lines
       public void getAngles(int col)
       {
              StartCoroutine (AnglesAux(col));
    }


    #endregion
}
```

SDK and Avatar animation

```
using UnityEngine;
using System.Collections.Generic;
using System.IO;
using System.Timers;
using System;
```

```
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Collections;

public class UDPReceive : MonoBehaviour
{

    // receiving Thread
    Thread receiveThread;

    public int count = 1;
    public List<GameObject> Stormtrooper = new List<GameObject>();
    public List<double[][]> FabRot = new List<double[][]>();
    public List<double[][]> Rots = new List<double[][]>();

    public int rotcount;

    public Vector3 rotPelvis;
    public Vector3 rotTrunk;
    public Vector3 rotRArm;
    public Vector3 rotLArm;
    public Vector3 rotRForearm;
    public Vector3 rotLForearm;

    public Quaternion QuatPelvisFab = new Quaternion();
    public Quaternion QuatTrunkFab = new Quaternion();
    public Quaternion QuatRArmFab = new Quaternion();
    public Quaternion QuatRForearmFab = new Quaternion();
    public Quaternion QuatLArmFab = new Quaternion();
    public Quaternion QuatLForearmFab = new Quaternion();

    Quaternion QuatPelvisFabAux = new Quaternion();

    double[][] MatAvatar = new double[4][];
    double[][] MatFab = new double[4][];
    double[][] Inv = new double[4][];
    double[][] MatRot = new double[4][];



    // GameObjects
    public GameObject PelvisAvatar;
    public GameObject TrunkAvatar;
    public GameObject RArmAvatar;
    public GameObject LArmAvatar;
    public GameObject RForearmAvatar;
    public GameObject LForearmAvatar;

    // chart stuff
    public List<float> flexEvolution = new List<float>();
       public List<float> extEvolution = new List<float>();
    private static int nEventsFired = 0;
    public float timer = 0f;
    public float wait = 0.1f;

    // udp stuff
    UdpClient client;
    public int port = 2123;
    public string lastReceivedUDPPacket = "";
    public string allReceivedUDPPackets = ""; // clean up this from
time to time!

    public byte[] bytes;
       public byte[] header;

    public float restrictAngle;

    private double[][] RotRollPelvis;
    private double[][] RotPitchPelvis;
    private double[][] RotYawPelvis;
    private double[][] MatRotTotPelvis;
    private double[][] RotRollTrunk;
    private double[][] RotPitchTrunk;
    private double[][] RotYawTrunk;
    private double[][] MatRotTotTrunk;
    private double[][] RotRollLArm;
    private double[][] RotPitchLArm;
    private double[][] RotYawLArm;
    private double[][] MatRotTotLArm;
    private double[][] RotRollLForearm;
    private double[][] RotPitchLForearm;
    private double[][] RotYawLForearm;
    private double[][] MatRotTotLForearm;
    private double[][] RotRollRArm;
    private double[][] RotPitchRArm;
    private double[][] RotYawRArm;
    private double[][] MatRotTotRArm;
    private double[][] RotRollRForearm;
    private double[][] RotPitchRForearm;
    private double[][] RotYawRForearm;
```

```
private double[][] MatRotTotRForearm;
    public float Updating;
    public float MaxFlex;
    public float MaxExt;
    List<Vector3> Restrictions = new List<Vector3>();
    float relevantAngle;
    public GameObject GameCanvas;
    public GameObject CanvasParent;
    public int stoptimes;

public void Start()
{
    restrictAngle = PlayerPrefs.GetFloat("restrictAngle");
    receiveThread = new Thread(
        new ThreadStart(ReceiveData));
    receiveThread.IsBackground = true;
    receiveThread.Start();

    Stormtrooper.Insert(0,PelvisAvatar);
    Stormtrooper.Insert(1, TrunkAvatar);
    Stormtrooper.Insert(2, RArmAvatar);
    Stormtrooper.Insert(3, LArmAvatar);
    Stormtrooper.Insert(4, RForearmAvatar);
    Stormtrooper.Insert(5, LForearmAvatar);


    FabRot.Insert(0, MatRotTotPelvis);
    FabRot.Insert(1, MatRotTotTrunk);
    FabRot.Insert(2, MatRotTotRArm);
    FabRot.Insert(3, MatRotTotLArm);
    FabRot.Insert(4, MatRotTotRForearm);
    FabRot.Insert(5, MatRotTotLForearm);

    if (PlayerPrefs.GetString ("Artic") == "Cotovelo") {
                Restrictions.Insert (0, rotTrunk);
          }


          // else if... (para outras articulações)


}


void OnDisable()
{
    if (receiveThread != null)
        receiveThread.Abort();

    client.Close();

}

public float leftarm;
public float rightarm;

void Update()
{

    while (count < 500)
    {

        count++;
    }

    if (count == 500)
    {
        FabRot[0] = MatRotTotPelvis;
        FabRot[1] = MatRotTotTrunk;
        FabRot[2] = MatRotTotRArm;
        FabRot[3] = MatRotTotLArm;
        FabRot[4] = MatRotTotRForearm;
        FabRot[5] = MatRotTotLForearm;

        for (int i = 0; i<Stormtrooper.Count; i++)
        {
            MatAvatar =
QuadMat.linda(Stormtrooper[i].transform.localRotation);
            MatFab = FabRot[i];
            Inv = QuadMat.MatrixInverse(MatFab);
            MatRot = QuadMat.MatrixProduct(MatAvatar, Inv);
            Rots.Insert(i, MatRot);
        }


        count++;
    }

       else
    {

            Updating = relevantAngle;
```

```
            FabRot[0] = MatRotTotPelvis;
            FabRot[1] = MatRotTotTrunk;
            FabRot[2] = MatRotTotRArm;
            FabRot[3] = MatRotTotLArm;
            FabRot[4] = MatRotTotRForearm;
            FabRot[5] = MatRotTotLForearm;


            for (int i = 0; i < Stormtrooper.Count; i++)
            {
                MatFab = FabRot[i];

                Stormtrooper[i].transform.localRotation =
(QuadMat.RottoQuat(QuadMat.MatrixProduct(Rots[i], MatFab)));
            }

        }


              if (PlayerPrefs.GetString ("Artic") == "Cotovelo")
              {
                     Restrictions [0] = rotTrunk;


                     if( PlayerPrefs.GetString("ArmKey") ==
"Esquerdo" )
                     {

                 relevantAngle = rotLForearm.x;

                     }

                     else if( PlayerPrefs.GetString("ArmKey") ==
"Direito" )
                     {
                 relevantAngle = rotRForearm.x;

            }

              }

              for (int j = 0; j < Restrictions.Count; j++)
              {
                     if ((Mathf.Abs(Restrictions[j].x) >
restrictAngle || Mathf.Abs(Restrictions[j].y) > restrictAngle) &&
GameCanvas.activeInHierarchy == (false)) //||
Input.GetKeyDown(KeyCode.P))//|| Mathf.Abs(Restrictions[j].z) > 30)
                     {

                            Time.timeScale = 0;
                            stoptimes++;
                            PlayerPrefs.SetInt ("Stoptimes",
stoptimes);
                            PlayerPrefs.Save ();
                     }
                     else if ((Mathf.Abs(Restrictions[j].x) <=
restrictAngle || Mathf.Abs(Restrictions[j].y) <= restrictAngle) &&
GameCanvas.activeInHierarchy == (false) ) //&&
!Input.GetKeyDown(KeyCode.P))//|| Mathf.Abs(Restrictions[j].z) <= 30)
                     {

                            Time.timeScale = 1;
                     }
              }

        leftarm = rotLForearm.x;

        rightarm = rotRForearm.x;

        if (Updating > 0 && Updating > MaxFlex && Time.timeScale == 1
)
        {
            MaxFlex = Updating;
            PlayerPrefs.SetFloat("MaxFlex", 90 + MaxFlex);
            PlayerPrefs.Save();
                     Updating = relevantAngle;
        }

               else if(Updating < 0 && Updating < MaxExt
&&Time.timeScale == 1)
                     {
                            MaxExt = Updating;
            PlayerPrefs.SetFloat("MaxExt", - (90 +MaxExt));
                            PlayerPrefs.Save();
                            Updating = relevantAngle;
                     }
```

```
        timer += Time.deltaTime;
               if (timer > wait && Time.timeScale == 1)
        {
                       if (relevantAngle > 0)
                       {
                              flexEvolution.Add (90 + relevantAngle);

                       } else if (relevantAngle <= 0)
                       {
                 extEvolution.Add(-(90 + relevantAngle));
                       }


            nEventsFired += 1;
            timer = 0f;
        }
    }

    public List<float> getflexEvolution()
    {
        return flexEvolution;
    }

       public List<float> getextEvolution()
       {
              return extEvolution;
       }

    // receive thread
    private void ReceiveData()
    {
        //receives data coming to port
        client = new UdpClient(port);
        while (true)
        {

            try
            {

                IPEndPoint anyIP = new IPEndPoint(IPAddress.Any, 0);
                bytes = client.Receive(ref anyIP);

                // Bytes mit der UTF8-Kodierung in das Textformat
kodieren.
                string text = Encoding.UTF8.GetString(bytes);

                // Den abgerufenen Text anzeigen.
                //print(">> " + text);

                // latest UDPpacket
                lastReceivedUDPPacket = text;


                //PELVIS
                rotPelvis.y = (bytes[16] << 8 | bytes[17]);
                rotPelvis.x = (bytes[18] << 8 | bytes[19]);
                rotPelvis.z = (bytes[20] << 8 | bytes[21]);

                if (rotPelvis.x > 32768)
                {
                    rotPelvis.x -= 65536;
                }
                if (rotPelvis.y > 32768)
                {
                    rotPelvis.y -= 65536;
                }
                if (rotPelvis.z > 32768)
                {
                    rotPelvis.z -= 65536;
                }

                rotPelvis /= 80;

                RotRollPelvis = QuadMat.MatRotRoll(rotPelvis.z);
                RotPitchPelvis = QuadMat.MatRotPitch(rotPelvis.x);
                RotYawPelvis = QuadMat.MatRotYaw(rotPelvis.y);

                MatRotTotPelvis =
QuadMat.MatrixProduct(QuadMat.MatrixProduct(RotYawPelvis,
RotPitchPelvis), RotRollPelvis);

                QuatPelvisFab = QuadMat.RottoQuat(MatRotTotPelvis);

                //TRUNK
                rotTrunk.y = (bytes[22] << 8 | bytes[23]);
                rotTrunk.x = (bytes[24] << 8 | bytes[25]);
                rotTrunk.z = (bytes[26] << 8 | bytes[27]);

                if (rotTrunk.x > 32768)
                {
                    rotTrunk.x -= 65536;
```

```
                }
                if (rotTrunk.y > 32768)
                {
                    rotTrunk.y -= 65536;
                }
                if (rotTrunk.z > 32768)
                {
                    rotTrunk.z -= 65536;
                }
                rotTrunk /= 80;


                RotYawTrunk = QuadMat.MatRotYaw(rotTrunk.y);
                RotPitchTrunk = QuadMat.MatRotPitch(rotTrunk.x);
                RotRollTrunk = QuadMat.MatRotRoll(rotTrunk.z);

                MatRotTotTrunk =
QuadMat.MatrixProduct(QuadMat.MatrixProduct(RotYawTrunk,
RotPitchTrunk), RotRollTrunk);

                QuatTrunkFab = QuadMat.RottoQuat(MatRotTotTrunk);

                //RIGHTARM
                rotRArm.y = (bytes[84] << 8 | bytes[85]);
                rotRArm.x = (bytes[86] << 8 | bytes[87]);
                rotRArm.z = (bytes[88] << 8 | bytes[89]);

                if (rotRArm.x > 32768)
                {
                    rotRArm.x -= 65536;
                }
                if (rotRArm.y > 32768)
                {
                    rotRArm.y -= 65536;
                }
                if (rotRArm.z > 32768)
                {
                    rotRArm.z -= 65536;
                }

                rotRArm /= 80;


                RotRollRArm = QuadMat.MatRotRoll(rotRArm.z);
                RotPitchRArm = QuadMat.MatRotPitch(rotRArm.x);
                RotYawRArm = QuadMat.MatRotYaw(rotRArm.y);

                MatRotTotRArm =
QuadMat.MatrixProduct(QuadMat.MatrixProduct(RotYawRArm,
RotPitchRArm), RotRollRArm);

                QuatRArmFab = QuadMat.RottoQuat(MatRotTotRArm);


                //LEFTARM
                rotLArm.y = (bytes[72] << 8 | bytes[73]);
                rotLArm.x = (bytes[74] << 8 | bytes[75]);
                rotLArm.z = (bytes[76] << 8 | bytes[77]);

                if (rotLArm.x > 32768)
                {
                    rotLArm.x -= 65536;
                }
                if (rotLArm.y > 32768)
                {
                    rotLArm.y -= 65536;
                }
                if (rotLArm.z > 32768)
                {
                    rotLArm.z -= 65536;
                }

                rotLArm /= 80;


                RotRollLArm = QuadMat.MatRotRoll(rotLArm.z);
                RotPitchLArm = QuadMat.MatRotPitch(rotLArm.x);
                RotYawLArm = QuadMat.MatRotYaw(rotLArm.y);

                MatRotTotLArm =
QuadMat.MatrixProduct(QuadMat.MatrixProduct(RotYawLArm,
RotPitchLArm), RotRollLArm);

                QuatLArmFab = QuadMat.RottoQuat(MatRotTotLArm);


                //RIGHTFOREARM
                rotRForearm.y = (bytes[90] << 8 | bytes[91]);
                rotRForearm.x = (bytes[92] << 8 | bytes[93]);
                rotRForearm.z = (bytes[94] << 8 | bytes[95]);

                if (rotRForearm.x > 32768)
```

```
                {
                    rotRForearm.x -= 65536;
                }
                if (rotRForearm.y > 32768)
                {
                    rotRForearm.y -= 65536;
                }
                if (rotRForearm.z > 32768)
                {
                    rotRForearm.z -= 65536;
                }

                rotRForearm /= 80;


                RotRollRForearm = QuadMat.MatRotRoll(rotRForearm.z);
                RotPitchRForearm =
QuadMat.MatRotPitch(rotRForearm.x);
                RotYawRForearm = QuadMat.MatRotYaw(rotRForearm.y);

                MatRotTotRForearm =
QuadMat.MatrixProduct(QuadMat.MatrixProduct(RotYawRForearm,
RotPitchRForearm), RotRollRForearm);

                QuatRForearmFab =
QuadMat.RottoQuat(MatRotTotRForearm);

                //LEFTFOREARM
                rotLForearm.y = (bytes[78] << 8 | bytes[79]);
                rotLForearm.x = (bytes[80] << 8 | bytes[81]);
                rotLForearm.z = (bytes[82] << 8 | bytes[83]);

                if (rotLForearm.x > 32768)
                {
                    rotLForearm.x -= 65536;
                }
                if (rotLForearm.y > 32768)
                {
                    rotLForearm.y -= 65536;
                }
                if (rotLForearm.z > 32768)
                {
                    rotLForearm.z -= 65536;
                }
                rotLForearm /= 80;

                RotRollLForearm = QuadMat.MatRotRoll(rotLForearm.z);
                RotPitchLForearm =
QuadMat.MatRotPitch(rotLForearm.x);
                RotYawLForearm = QuadMat.MatRotYaw(rotLForearm.y);

                MatRotTotLForearm =
QuadMat.MatrixProduct(QuadMat.MatrixProduct(RotYawLForearm,
RotPitchLForearm), RotRollLForearm);

                QuatLForearmFab =
QuadMat.RottoQuat(MatRotTotLForearm);


                //UNCOMMENT IN THE FUTURE
                //allReceivedUDPPackets=allReceivedUDPPackets+text;

            }
            catch (Exception err)
            {
                print(err.ToString());
            }
        }
    }
}
```

Matrix and Quaternion Operations

```
using System;
using UnityEngine;
using System.Collections;
using System.IO;
using System.Text;
using System.Collections.Generic;


internal static class QuadMat
    {
        public static double[][] linda(Quaternion quaternion)
```

```
        {


         double[][] mat = MatrixCreate(3, 3);

        mat[0][0] = 1 - 2f * Mathf.Pow(quaternion.y, 2) - 2f *
Mathf.Pow(quaternion.z, 2);

        mat[0][1] = (2 * quaternion.x * quaternion.y) - (2 *
quaternion.z * quaternion.w);

        mat[0][2] = (2 * quaternion.x * quaternion.z) + (2 *
quaternion.y * quaternion.w);


        mat[1][0] = (2 * quaternion.x * quaternion.y) + (2 *
quaternion.z * quaternion.w);

        mat[1][1] = 1 - 2f * Mathf.Pow(quaternion.x, 2) - 2f *
Mathf.Pow(quaternion.z, 2);

        mat[1][2] = (2 * quaternion.y * quaternion.z) - (2 *
quaternion.x * quaternion.w);


        mat[2][0] = (2 * quaternion.x * quaternion.z) - (2 *
quaternion.y * quaternion.w);

        mat[2][1] = (2 * quaternion.y * quaternion.z) + (2 *
quaternion.x * quaternion.w);

        mat[2][2] = 1 - 2f * Mathf.Pow(quaternion.x, 2) - 2f *
Mathf.Pow(quaternion.y, 2);

        return mat;

        }


        public static double[][] MatrixCreate(int rows, int cols)

        {

            double[][] result = new double[rows][];

            for (int i = 0; i < rows; ++i)

                result[i] = new double[cols];

            return result;

        }


        static double[][] MatrixIdentity(int n)

        {

            // return an n x n Identity matrix

            double[][] result = MatrixCreate(n, n);

            for (int i = 0; i < n; ++i)

                result[i][i] = 1.0;


            return result;

        }


        public static double[][] MatrixProduct(double[][] matrixA,
double[][] matrixB)

        {

            int aRows = matrixA.Length; int aCols =
matrixA[0].Length;

            int bRows = matrixB.Length; int bCols =
matrixB[0].Length;

            if (aCols != bRows)

                throw new Exception("Non-conformable matrices in
MatrixProduct");


            double[][] result = MatrixCreate(aRows, bCols);


            for (int i = 0; i < aRows; ++i) // each row of A

                for (int j = 0; j < bCols; ++j) // each col of B

                    for (int k = 0; k < aCols; ++k) // could use k
less-than bRows

                        result[i][j] += matrixA[i][k] *
matrixB[k][j];
```

```
            return result;
        }

        static double[][] MatrixDecompose(double[][] matrix, out
int[] perm,
          out int toggle)
        {
            // Doolittle LUP decomposition with partial pivoting.
            // rerturns: result is L (with 1s on diagonal) and U;
            // perm holds row permutations; toggle is +1 or -1 (even
or odd)
            int rows = matrix.Length;
            int cols = matrix[0].Length; // assume square
            if (rows != cols)
                throw new Exception("Attempt to decompose a non-
square m");

            int n = rows; // convenience

            double[][] result = MatrixDuplicate(matrix);

            perm = new int[n]; // set up row permutation result
            for (int i = 0; i < n; ++i) { perm[i] = i; }

            toggle = 1; // toggle tracks row swaps.
                        // +1 -greater-than even, -1 -greater-than
odd. used by MatrixDeterminant

            for (int j = 0; j < n - 1; ++j) // each column
            {
                double colMax = Math.Abs(result[j][j]); // find
largest val in col
                int pRow = j;
                //for (int i = j + 1; i less-than n; ++i)
                //{
                //  if (result[i][j] greater-than colMax)
                //  {
                //    colMax = result[i][j];
                //    pRow = i;
                //  }
                //}

                // reader Matt V needed this:
                for (int i = j + 1; i < n; ++i)
                {
                    if (Math.Abs(result[i][j]) > colMax)
                    {
                        colMax = Math.Abs(result[i][j]);
                        pRow = i;
                    }
                }
                // Not sure if this approach is needed always, or
not.

                if (pRow != j) // if largest value not on pivot, swap
rows
                {
                    double[] rowPtr = result[pRow];
```

```
    result[pRow] = result[j];
    result[j] = rowPtr;

    int tmp = perm[pRow]; // and swap perm info
    perm[pRow] = perm[j];
    perm[j] = tmp;

    toggle = -toggle; // adjust the row-swap toggle
}

// ------------------------------------------------
// This part added later (not in original)
// and replaces the 'return null' below.
// if there is a 0 on the diagonal, find a good row
// from i = j+1 down that doesn't have
// a 0 in column j, and swap that good row with row j
// ------------------------------------------------

if (result[j][j] == 0.0)
{
    // find a good row to swap
    int goodRow = -1;
    for (int row = j + 1; row < n; ++row)
    {
        if (result[row][j] != 0.0)
            goodRow = row;
    }
    if (goodRow == -1)
        throw new Exception("Cannot use Doolittle's method");

    // swap rows so 0.0 no longer on diagonal
    double[] rowPtr = result[goodRow];
    result[goodRow] = result[j];
    result[j] = rowPtr;

    int tmp = perm[goodRow]; // and swap perm info
    perm[goodRow] = perm[j];
    perm[j] = tmp;

    toggle = -toggle; // adjust the row-swap toggle
}
// ------------------------------------------------
// if diagonal after swap is zero . .
//if (Math.Abs(result[j][j]) less-than 1.0E-20)
//  return null; // consider a throw

for (int i = j + 1; i < n; ++i)
{
    result[i][j] /= result[j][j];
    for (int k = j + 1; k < n; ++k)
    {
        result[i][k] -= result[i][j] * result[j][k];
    }
}
```

```
    } // main j column loop

    return result;
} // MatrixDecompose

public static double[][] MatrixInverse(double[][] matrix)
{
    int n = matrix.Length;
    double[][] result = MatrixDuplicate(matrix);

    int[] perm;
    int toggle;
    double[][] lum = MatrixDecompose(matrix, out perm,
      out toggle);
    if (lum == null)
        throw new Exception("Unable to compute inverse");

    double[] b = new double[n];
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            if (i == perm[j])
                b[j] = 1.0;
            else
                b[j] = 0.0;
        }
            double[] x = HelperSolve(lum, b); //

            for (int j = 0; j < n; ++j)
                result[j][i] = x[j];
        }
        return result;
    }

    static double MatrixDeterminant(double[][] matrix)
    {
        int[] perm;
        int toggle;
        double[][] lum = MatrixDecompose(matrix, out perm, out
toggle);
        if (lum == null)
            throw new Exception("Unable to compute
MatrixDeterminant");
        double result = toggle;
        for (int i = 0; i < lum.Length; ++i)
            result *= lum[i][i];
        return result;
    }

    static double[] HelperSolve(double[][] luMatrix, double[] b)
    {
        // before calling this helper, permute b using the perm
array
        // from MatrixDecompose that generated luMatrix
```

```
        int n = luMatrix.Length;
        double[] x = new double[n];
        b.CopyTo(x, 0);

        for (int i = 1; i < n; ++i)
        {
            double sum = x[i];
            for (int j = 0; j < i; ++j)
                sum -= luMatrix[i][j] * x[j];
            x[i] = sum;
        }

        x[n - 1] /= luMatrix[n - 1][n - 1];
        for (int i = n - 2; i >= 0; --i)
        {
            double sum = x[i];
            for (int j = i + 1; j < n; ++j)
                sum -= luMatrix[i][j] * x[j];
            x[i] = sum / luMatrix[i][i];
        }

        return x;
    }

    static double[][] MatrixDuplicate(double[][] matrix)
    {
        // allocates/creates a duplicate of a matrix.
        double[][] result = MatrixCreate(matrix.Length,
matrix[0].Length);
        for (int i = 0; i < matrix.Length; ++i) // copy the
values
            for (int j = 0; j < matrix[i].Length; ++j)
                result[i][j] = matrix[i][j];
        return result;
    }
    public static Quaternion MattoQuad(double[][] weee)
    {
        Quaternion result = new Quaternion();
        result.x = (float)weee[1][0];
        result.y = (float)weee[2][0];
        result.z = (float)weee[3][0];
        result.w = (float)weee[0][0];
        return result;
    }

    public static double ConvertToRadians(double angle)
    {
        return (Math.PI / 180) * angle;
    }

    public static double[][] MatRotRoll(double angle)
    {
        double[][] matRot = MatrixCreate(3, 3);
        matRot[0][0] = Math.Cos(ConvertToRadians(angle));
        matRot[0][1] = Math.Sin(ConvertToRadians(angle));
        matRot[1][0] = -Math.Sin(ConvertToRadians(angle));
        matRot[1][1] = Math.Cos(ConvertToRadians(angle));
        matRot[2][2] = 1;
```

```
    return matRot;
}

public static double[][] MatRotPitch(double angle)
{
    double[][] matRot = MatrixCreate(3, 3);
    matRot[1][1] = Math.Cos(ConvertToRadians(angle));
    matRot[1][2] = Math.Sin(ConvertToRadians(angle));
    matRot[2][1] = -Math.Sin(ConvertToRadians(angle));
    matRot[2][2] = Math.Cos(ConvertToRadians(angle));
    matRot[0][0] = 1;

    return matRot;
}

public static double[][] MatRotYaw(double angle)
{
    double[][] matRot = MatrixCreate(3, 3);
    matRot[0][0] = Math.Cos(ConvertToRadians(angle));
    matRot[2][0] = Math.Sin(ConvertToRadians(angle));
    matRot[0][2] = -Math.Sin(ConvertToRadians(angle));
    matRot[2][2] = Math.Cos(ConvertToRadians(angle));
    matRot[1][1] = 1;

    return matRot;
}
```

```
public static double[][] Transpose(double[][] matrix)
{
    int rows = matrix.Length;
    int cols = matrix[0].Length;

    double[][] result = new double[rows][];

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            result[j][i] = matrix[i][j];
        }
    }

    return result;
}

public static Quaternion RottoQuat(double[][] rot)
{

    Quaternion q = new Quaternion();
    double tr = rot[0][0] + rot[1][1] + rot[2][2];


    if (rot[1][1] > -rot[2][2] && rot[0][0] > -rot[1][1] &&
rot[0][0] > -rot[2][2])
    {
        double S = Math.Sqrt(tr + 1.0) * 2;
```

```
            q.w = 0.25f * (float)(S);
            q.x = (float)((rot[2][1] - rot[1][2]) / S);
            q.y = (float)((rot[0][2] - rot[2][0]) / S);
            q.z = (float)((rot[1][0] - rot[0][1]) / S);
        }
        else if ((rot[0][0] > rot[1][1]) & (rot[0][0] > rot[2][2]) &&
(rot[1][1] < -rot[2][2]))
        {
            double S = Math.Sqrt(1.0 + rot[0][0] - rot[1][1] -
rot[2][2]) * 2;
            q.w = (float)((rot[2][1] - rot[1][2]) / S);
            q.x = 0.25f * (float)(S);
            q.y = (float)((rot[0][1] + rot[1][0]) / S);
            q.z = (float)((rot[0][2] + rot[2][0]) / S);
        }
        else if (rot[1][1] > rot[2][2] && rot[0][0] < rot[1][1] &&
rot[0][0] < -rot[2][2])
        {
            double S = Math.Sqrt(1.0 + rot[1][1] - rot[0][0] -
rot[2][2]) * 2;
            q.w = (float)((rot[0][2] - rot[2][0]) / S);
            q.x = (float)((rot[0][1] + rot[1][0]) / S);
            q.y = 0.25f * (float)(S);
            q.z = (float)((rot[1][2] + rot[2][1]) / S);
        }
        else if(rot[1][1] < rot[2][2] && rot[0][0] < -rot[1][1] &&
rot[0][0] < rot[2][2])
        {
            double S = Math.Sqrt(1.0 + rot[2][2] - rot[0][0] -
rot[1][1]) * 2;
            q.w = (float)((rot[1][0] - rot[0][1]) / S);
            q.x = (float)((rot[0][2] + rot[2][0]) / S);
            q.y = (float)((rot[1][2] + rot[2][1]) / S);
            q.z = 0.25f * (float)(S);
        }
        return q;
    }
}
```

Game Mechanics – Projectile Pooler

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpherePooler : MonoBehaviour
{

    public GameObject pooledSphere;
    public int pooledAmount;
    public List<GameObject> pooledSpheres;

    void Start()
    {
        pooledSpheres = new List<GameObject>();
        for (int i = 0; i < pooledAmount; i++)
        {
            GameObject sphere =
(GameObject)Instantiate(pooledSphere);
```

```
            sphere.transform.parent = this.transform.parent;
            sphere.SetActive(false);
            pooledSpheres.Add(sphere);
        }
    }

    public GameObject GetPooledSphere()
    {
        for (int i = 0; i < pooledSpheres.Count; i++)
        {
            if (!pooledSpheres[i].activeInHierarchy)
            {
                return pooledSpheres[i];
            }
        }
        GameObject sphere = (GameObject)Instantiate(pooledSphere);
        sphere.transform.parent = this.transform.parent;
        sphere.SetActive(false);
        pooledSpheres.Add(sphere);
        return sphere;
    }
}
```

Game Mechanics – Characters Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class SphereGenerator : MonoBehaviour
{

    public int maxSpheres;
    public int currSpheres = 0;
    public float wait;
    public float timer = 0f;
    float speed;
    public float minWait;
    public float maxWait = 20;
    public int lives;
    public SpherePooler[] spherePools;
    private int sphereSelector;
    GameObject newSphere;
    public float percentage;
       public float minRangex;
       public float maxRangex;
       public float minRangey;
       public float maxRangey;
       public string arm;
    Color armcolor;

    void Start()
       {
              Settings ();
        SetLives();
```

```
    }

    void Update()
    {
                currSpheres = GameObject.FindGameObjectsWithTag
("Sphere").Length;

        timer += Time.deltaTime;
        wait = Random.Range(minWait, maxWait);

               if (timer > wait && currSpheres <= 1)
        {
            Spawn();
            timer = 0f;
                      wait = Random.Range(minWait, maxWait);
                      currSpheres = GameObject.FindGameObjectsWithTag
("Sphere").Length;

            if (currSpheres >= maxSpheres)
            {

GameObject.FindGameObjectWithTag("Sphere").SetActive(false);
                currSpheres--;
            }
        }


    }
```

```
    public void Settings ()
    {
           arm = PlayerPrefs.GetString ("ArmKey");
     speed = PlayerPrefs.GetFloat("Velocidade inicial");
     minWait = PlayerPrefs.GetFloat("Cadência mínima");

     percentage = PlayerPrefs.GetFloat("Percentagem");
    }

public void SetLives()
{
    int level = PlayerPrefs.GetInt("Level");
           maxSpheres = (level+1) *2;
    lives = PlayerPrefs.GetInt(level.ToString());
}

void Spawn()
{

    int perCent = Random.Range(0, 100);
    GameObject[] Shields =
GameObject.FindGameObjectsWithTag("Shield");
    for(int i = 0; i < Shields.Length; i++)
    {
        if(Shields[i].name == arm)
        {
            armcolor =
Shields[i].GetComponent<MeshRenderer>().material.color;
        }
```

```
        }

        for(int j = 0; j < spherePools.Length; j++)
        {
            if(perCent < percentage)
            {
                                if
(spherePools[j].GetPooledSphere().gameObject.GetComponent<MeshRendere
r>().material.color == armcolor)
                {
                    newSphere = spherePools[j].GetPooledSphere();
                }
            }

            else if(perCent >= percentage)
            {
                                if
(spherePools[j].GetPooledSphere().gameObject.GetComponent<MeshRendere
r>().material.color != armcolor)
                {
                    newSphere = spherePools[j].GetPooledSphere();
                }
            }

        }
        newSphere.transform.parent =
this.gameObject.transform.parent;
               newSphere.transform.position = new
Vector3(transform.position.x, transform.position.y,
transform.position.z);
        newSphere.SetActive(true);
            Rigidbody rigidSphere = newSphere.GetComponent<Rigidbody>();
            Vector3 movement = rigidSphere.velocity;
                   movement.z = -speed;
    }

    public void LessSpheres()
    {
        currSpheres--;
    }

    public void LessLives()
    {
        this.lives--;
        if (this.lives == 0)
        {
this.gameObject.transform.parent.gameObject.SetActive(false);
            GameObject manager = GameObject.Find("Manager");
            var g = manager.GetComponent<LoginDemo_Accounts>();
            g.EnemyDown();
        }
    }
}
```

Game Mechanics – Disk Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
```

```
public class Shields : MonoBehaviour {

    public GameObject Shield;
    public GameObject remains;
    public Text scoreText;
    public int score;
    public int missed;
    public int level;
    public int countlevels;
    public string arm;
    Quaternion rotation;

    void Start ()
        {
        rotation = transform.rotation;
               SetArm ();
               SetInitialLevel ();
               countlevels = 1;
    }

       void Update ()
       {
               ZoomedPast ();
        transform.rotation = rotation;
    }

    void SetInitialLevel (){
               level = PlayerPrefs.GetInt ("InitialLevel");
    }

    public void SetArm ()
    {
               arm = PlayerPrefs.GetString ("ArmKey");
    }

public void IncrementScore()
{
    score++;
           scoreText.text = score.ToString();
}

public void ReduceScore()
{
    score--;
    scoreText.text = score.ToString();
}

public int getScore()
{
    return score;
}

void OnCollisionEnter(Collision collision)
{
    //right disk
```

```
        if (collision.gameObject.tag == "Sphere" &&
collision.gameObject.GetComponent<MeshRenderer>().material.color ==
Shield.GetComponent<MeshRenderer>().material.color)

        {


            var g =
collision.gameObject.transform.parent.GetComponentInChildren<SphereGe
nerator>();

            collision.gameObject.SetActive(false);

            GameObject instantremains = Instantiate(remains,
collision.gameObject.transform.position,
collision.gameObject.transform.rotation);


            foreach (var child in
instantremains.GetComponentsInChildren<Renderer>())

            {

                child.material.color =
Shield.GetComponent<MeshRenderer>().material.color;

            }

            collision.gameObject.transform.rotation =
collision.gameObject.transform.parent.rotation;

            IncrementScore();

            g.LessLives();




        }


        //wrong disk

        else if (collision.gameObject.tag == "Sphere" &&
collision.gameObject.GetComponent<MeshRenderer>().material.color !=
Shield.GetComponent<MeshRenderer>().material.color)

        {

            collision.gameObject.SetActive(false);

            GameObject instantremains = Instantiate(remains,
collision.gameObject.transform.position,
collision.gameObject.transform.rotation);

            foreach (var child in
instantremains.GetComponentsInChildren<Renderer>())

            {

                child.material.color = Color.red;

            }

            collision.gameObject.transform.rotation =
collision.gameObject.transform.parent.rotation;

        }

    }


       public void ZoomedPast(){

              GameObject[] Spheres =
GameObject.FindGameObjectsWithTag ("Sphere");

              for (int i = 0; i < Spheres.Length; i++)

              {

                     if (Spheres [i].transform.localPosition.z >= 15
&& Spheres[i].GetComponent<MeshRenderer>().material.color ==
Shield.GetComponent<MeshRenderer>().material.color)

                     {

                            Spheres[i].SetActive(false);


               Spheres[i].transform.rotation =
Spheres[i].transform.parent.rotation;

               var g =
Spheres[i].gameObject.GetComponentInParent<SphereGenerator>();

                     }

              }

       }
```

```
    public void UpdateMissed()
    {
        missed++;
    }
}
```