Neuroscientific fidelity metrics for interactive computer graphics scenes

Mavromichelaki Evangelia

Committee

Katerina Mania (Supervisor)
Michail Lagoudakis
Antonios Deligiannakis

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering.

School of Electrical and Computer Engineering
Laboratory of Distributed Multimedia Information Systems and Applications – *TUC/MUSIC*
CHANIA 2016

Abstract

This work presents a 3D interactive gaming paradigm called, Cyberball3D+, for the secluded space of an fMRI scanner. The Cyberball3D+ game is a virtual ball-toss game, where the participant is either excluded or not from ball tossing played by three virtual players and the subject in the scanner. It has been used in simple sketch mode by neuroscientists for research on ostracism, social exclusion or rejection as well as discrimination and prejudice.

The game proposed was designed to render an interactive Virtual Environment (VE) on an fMRI display, enabling the conduct of formal neuroscientific experiments and investigating the effects of social exclusion, empathy and different levels of anthropomorphism on human brain activity. Although this work focused on the technical implementation of the system, the goal was to use this system to explore whether the pain felt by someone when socially excluded is the same when observing other people get socially excluded. Moreover, for the first time, we proposed a validated neuroscientific measure of character believability and emotional engagement.

The system was developed in close collaboration between the Technical University of Crete, where the technical implementation took place and the Brighton and Sussex Medical School, where the initial fMRI experiments were conducted using the system proposed.

Ten healthy adult volunteers (8 female, 2 male) underwent fMRI at Brighton and Sussex Medical School. In a block design they participated in several rounds of the Cyberball3D+ task, which included from combinations of low and high anthropomorphism, inclusion of all avatars or exclusion of self and exclusion of other, simulating social exclusion or empathy for social exclusion. Each round was 75 seconds long. Two buttons from a 4-button interface were used to throw the ball left or right. User interactions were synchronized to the fMRI scanner by using trigger information. A frequency modulated audio signal was generated at prescribed times within the experimental phase. The audio signal was fed into a biometric recorder, which also recorded heartbeat, scanning synchronization etc. A log was generated marking the exact time the sync pulses were sent to the biometric recorder as well as logs for user interactions. Neuroscientists used a dedicated user interface to select the level of anthropomorphism of all avatars, the gender of each avatar and the fairness of each round.

The results demonstrated that participating in a high anthropomorphism environment rather than a low anthropomorphism environment activated both frontal cortex and superior temporal gyrus. This suggests that compared to more human like avatars, playing the non-anthropomorphic avatars is less subjectively rewarding and potentially anxiogenic. In addition, the results indicated that when studying complex emotional responses, a high level of anthropomorphism of synthetic characters engages neuroscientific patterns of brain activation similar to real-world circumstances.

A broader aim of this work was to assess whether such powerful social-psychological studies could be usefully carried out within VEs advancing cognitive neuroscience and computer graphics as well as serious gaming research.

Declaration

The work in this thesis is original and no portion of the work referred to here has been submitted in support of an application for another degree or qualification of this or any other university or institution of learning.

Signed: Date:

Mavromichelaki Evangelia

Acknowledgements

Σε αυτή την ενότητα θα χρησιμοποιήσω την ελληνική γλώσσα σε αντίθεση με το υπόλοιπο κείμενο για να μην αλλάξω το μοτίβο των προκάτοχών μου.. :-P

Αρχικά οφείλω ένα μεγάλο ευχαριστώ στην επιβλέπουσα καθηγήτριά μου την κ. Κατερίνα Μανιά για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον και πρωτότυπο θέμα, για την συνεχή καθοδήγηση, επίβλεψη και την άψογη συνεργασία. Η εργασία αυτή μου έδωσε τη δυνατότητα να ασχοληθώ με ένα θέμα που με προβληματίζει ιδιαίτερα και το συναντάω καθημερινά στη ζωή μου. Σαν ευαίσθητος άνθρωπος βιώνω έντονα την απόρριψη του συναθρώπου μου αλλά και του ίδιου μου του εαυτού μου από διάφορες κοινωνικές ομάδες. Όλοι οι άνθρωποι δυστυχώς έχουν βιώσει στη ζωή τους τον κοινωνικό αποκλεισμό από οποιαδήποτε κοινωνική ομάδα. Όνειρο μου σαν προγραμματίστρια των Η/Υ είναι να ασχολούμε με την ανάπτυξη προτζεκτ που αφορούν τον άνθρωπο γενικά και ειδικά στην βελτίωση της σωματικής και ψυχικής του υγείας και είναι μεγάλη μου χαρά που αυτή η εργασία μου έδωσε αυτήν την δυνατότητα.

Επίσης ένα μεγάλο ευχαριστώ χρωστάω στον καθηγητή των προπτυχιακών μου σπουδών τον κ. Νικόλαο Παπαδάκη για την σύσταση του στην κ. Μανιά και την ενημέρωση του ότι υπάρχει μια θέση στο μεταπτυχιακό πρόγραμμα με την δυνατότητα χορήγησης υποτροφίας.

Θα ήθελα επίσης να ευχαριστήσω και τους συνεργάτες μας στην Αγγλία, τον Prof. Hugo Critchley, τον Dr. Neil Harrison και, την Dr. Jessica Eccles για την πολύτιμη βοήθειά τους και την διεξαγωγή των πειραμάτων. Αν και έχω ένα μικρό παράπονο ότι δεν μας έδωσαν αρκετά αποτελέσματα.

Επιπλέον, θέλω να ευχαριστήσω τους κυρίους Λαγουδάκη και Δεληγιαννάκη για το χρόνο που θα αφιερώσουν στην ανάγνωση αυτής της εργασίας (κάθε σχόλιο και διόρθωση είναι καλοδεχούμενη). Επίσης δεν μπορώ να μην ευχαριστήσω τον συνάδελφο και φίλο Κουλιέρη Γιώργο για την βοήθεια του όποτε την χρειάστηκα, τις συμβουλές του και την συμπαράστασή του καθώς και για τις ατέλειωτες ώρες που περάσαμε μαζί στα γραφεία του MUSIC. Εκτός από τον Γιώργο, ευχαριστώ και όλα τα παιδιά του εργαστηρίου του MUSIC και ιδιαίτερα τους Γιώργο, Μάριο και Ανδρέα που ήταν στο διπλανό γραφείο και γελάγαμε και φιλοσοφούσαμε στα διαλείμματά μας καθώς και τη συμπαράσταση που μου έδειχναν τα πρωινά που ερχόντουσαν στο γραφείο και εγώ βρισκόμουν ήδη εκεί από το προηγούμενο βράδυ σχεδιάζοντας το παιχνίδι.

Ταυτόχρονα ευχαριστώ θερμά όλους τους φίλους μου που ήταν δίπλα μου καθόλη την διάρκεια των μεταπτυχιακών σπουδών μου και μου έδιναν δύναμη κάθε φορά που αγχωνόμουν και απελπιζόμουν ότι κάτι δε θα πάει καλά. Επίσης, ευχαριστώ όλα τα παιδιά που συμμετείχαν στον 2^{ο} γύρο των πειραμάτων, Γιώργος, Γιώργος, Μάριος, Γρηγόρης, Νίκος, Βαγγέλης, Ελένη, Μπάμπης, Κώστας, Άννα, Χρυσή και Χρυσόστομος, χωρίς αυτούς δε θα μπορούσα να είχα βγάλει τα επιπλέον αποτελέσματα που χρειαζόμουν.

Για το τέλος κράτησα τους γονείς μου, Θανάση και Μαρία, που μου έδωσαν τα εφόδια και οικονομικά και ψυχικά να παλεύω και να μην το βάζω κάτω στις δύσκολες στιγμές. Παρά τις οικονομικές δυσκολίες και χωρίς φροντιστήρια κατάφερα να περάσω στη σχολή της 1^{ης} μου επιλογής <<Τμήμα Επιστήμη Υπολογιστών Ηρακλείου>> (ήθελα να ζήσω φοιτητική ζωή μακριά από τα Χανιά γι'αυτό και δήλωσα ως 2^{η} επιλογή το Πολυτεχνείο παρότι πέρναγα) και να συνεχίσω μετά τις μεταπτυχιακές μου σπουδές στο Πολυτεχνείο Κρήτης. Οι γονείς μου έδωσαν από το υστέρημα τους για να καταφέρω εγώ σήμερα να υπάρχω ως επιστήμονας, όσο μπορώ δηλαδή να λέγομαι, και να παραδίδω την μεταπτυχιακή μου εργασία και για αυτό τους την αφιερώνω και τους ευχαριστώ μέσα από την καρδιά μου!! Αφιερωμένο στους ΓΟΝΕΙΣ μου λοιπόν….

Publications

- Mavromihelaki, E., Eccles, J., Harrison, N., Critchley, H., & Mania, K. (2014, July). Cyberball3D+ for fMRI: implementing neuroscientific gaming. In ACM SIGGRAPH 2014 Posters (p. 29). ACM.

- Mavromihelaki, E., Eccles, J., Harrison, N., Grice-Jackson, T., Ward, J., Critchley, H., & Mania, K. (2014, September). Cyberball3D+: A 3D Serious Game for fMRI Investigating Social Exclusion and Empathy. In Games and Virtual Worlds for Serious Applications (VS-GAMES), 2014 6th International Conference on (pp. 1-8). IEEE.

Table of Contents

List of Figures

List of Tables

1 Chapter 1 – Introduction

Immersive Virtual Environments are virtual environments that present rich layers of synthesized sensory cues to the user so that the user feels enveloped by the mediated environment and is willing to believe that the environment is real **(Witmer and Singer 1998)**. The immersive virtual environments (IVEs) were first introduced as a research tool in the late 1980s by psychologists conducting studies on perception and spatial cognition. Particularly, social psychologists explored the incorporation of virtual humans within IVEs to examine the possibility of using virtual reality technology as a research tool to answer social scientific questions **(Loomis, Blascovich and Beall 1999)**. Immersive Virtual Environments allow the researcher to create experimental situations of 'mundane' realism compared to the rigidly controlled laboratory settings, eliciting genuine participant reactions to the stimuli.

3D characters are used in IVEs but it is still undetermined how design, stylization and behavioral factors interweave to make a character believable. When using synthetic characters in neuroscientific experiments, the aim is that people emotionally respond to them in a similar manner to humans. Although research has shown that users show empathy for 3D characters, it is challenging to identify at which level of anthropomorphism such emotional experiences occur. Also, it has been shown that people tend to respond realistically to events within synthetic environments and even to virtual humans in spite of their relatively low fidelity compared to reality **(Mania et al. 2010)**. For example, VEs have been used in studies of social anxiety and behavioural problems and individuals with paranoid tendencies have been shown to experience paranoid thoughts in the company of virtual characters **(Freeman et al. 2008)**. These provide specific examples of 'presence' – the tendency of participants to respond to virtual events and situations as if they were real.

One recent study suggests that people interact with virtual characters in a realistic and emotionally engaged way **(Slater et al. 2006)**. This study reported a scenario where experimental participants took part in a virtual version of the Milgram experiment, in which people were asked to administer increasingly severe punishments to virtual characters performing a memory task **(Milgram 1963)**. Participants showed autonomic responses which were consistent with states of intense emotional arousal, as would be expected if the experiment had used real participants. Thus, it is evident that people are able to emotionally engage with synthetic spaces and virtual characters as if they were real. Previous research evaluating whether virtual characters fulfill their role employs ratings of pleasantness through self-report after the viewing experience has occurred **(McDonnell,**

Breidt and Bülthoff 2012). It would be valuable to understand the cognitive processes involved while interacting with 3D characters in a gaming scenario. 3D characters could be employed to simulate experimental scenarios as part of neuroscientific protocols in the fMRI.

An economic game combined with Milgram's original experimental scenario has been implemented which, could be interactively played in an fMRI scanner **(Rivera et al. 2010)**. A lighting system has been designed to render an interactive VE on an fMRI display, enabling the conduct of formal neuroscientific experiments and investigating the effects of visual fidelity as well as varied lighting configurations of an indoors/outdoors space on feelings of presence, 'reality' and comfort **(Christodoulou et al. 2012)**. A 3D virtual system for fMRI has been designed investigating the influence of two prominent VR parameters, e.g. 3D-motion and interactivity, while brain activity is measured for a mental rotation task **(Sjölie et al 2010)**. The subjects perform a variation of the mental rotation task in a simple VE and the brain activity was recorded during three conditions of varied 3D-motion and interactivity. Also, an interactive digital game based on the Re-Mission video game has been designed for cancer patients in the fMRI in order to determine whether mesolimbic neural circuits associated with incentive motivation are activated, and if so, whether such effects stem from the participatory aspects of interactive gameplay, or from the complex sensory/perceptual engagement generated by its dynamic event-stream **(Cole, Yoo and Knutson 2012)**.

However, acquiring user input and the reacting to it in real-time while the user immersed in the constrained environment of an fMRI scanner is challenging, even more if 3D characters are included as part of the neuroscientific protocol. fMRI experiments usually employ simple display material, for example using photographs, video clips or simple computerized stimuli **(S. Lee, G.J. Kim and J. Lee 2004)**. It is argued that the precise presentation and control of dynamic perceptual stimuli (visual, auditory, olfactory, gustatory, ambulatory, and haptic conditions) in the VE allows neuropsychologists the opportunity to develop statistically and clinically significant tasks within a virtual world **(Parsons 2011)**. Using VEs and 3D characters in fMRI has the advantage that it is possible to involve participants in interactive animated environments which more realistically reflect social and emotional situations.

This work presents an interactive 3D gaming framework for fMRI experiments exploring whether artificial characters of varied anthropomorphism recruit brain activation associated with empathy and social pain **(Meyer et al. 2012)**. Such input is non-obtrusive

derived at the same time as the experience occurs. The 3D interactive gaming paradigm presented here, named Cyberball3D+, is based on the original Cyberball game, however, it is implemented for the secluded space of an fMRI scanner. The Cyberball game is a virtual ball-toss game where participant is either excluded or not from ball tossing played by three virtual players and the subject. It has been used in simple sketch mode by neuroscientists for research on ostracism, social exclusion or rejection as well as discrimination and prejudice.

The empathy feeling is a crucial component of human emotional experience and social interaction. The ability to share the affective states of our closest ones and complete strangers allows us to predict and understand their feelings, motivations, and actions. Empathy allows humans to understand and share one another's emotional experiences and it is important for successful social interactions **(Eisenberg and Miller 1987)**. Empathy refers to experiencing an affective response that is more consistent with another person's situation than one's own situation which suggests that vicarious emotions are pivotal to empathy **(Hoffman 2001)**.

During the first Cyberball experiments, participants were recruited to log on to an online experiment where they played a virtual ball-tossing game with two other participants who had logged on from somewhere else in the world **(Williams, Cheung and Choi 2000)**. Before the experiment, the participants were informed that Cyberball was a means to an end, and was, by itself, unimportant for the experiment. It was portrayed as merely a task that helped participants exercise their mental visualization skills, which they would purportedly use in the subsequent experimental task. After reading the instructions, they would view a game where the players were represented on the screen by animated icons. They would then play the Cyberball for about 5 min. The results showed that if participants were over-included (getting the ball for half the throws) or included (getting the ball for one third of the throws), they felt better than if they received the ball for only one sixth of the throws. Still, getting the ball for a sixth of the time was significantly better than not getting it at all. Fully ostracized participants answered a post-experimental questionnaire indicating lower levels of belonging, self-esteem, control, and meaningful existence **(Williams and Jarvis 2006)**.

Brain imaging studies demonstrate that physical pain sensations are processed across a network of connected brain regions, which together are proposed to form the 'pain matrix'. Most of the pain matrix is also activated if we observe empathically someone else in physical pain **(Eisenberger 2012)**. Interestingly, experiencing the 'pain of social exclusion' also engages these same regions during brain imaging studies of the original Cyberball task,

consistent with the notion of social pain. There is early work suggesting that a similar brain activation pattern is present when one sees and empathizes with someone else being ostracized **(Eisenberger 2012)**. The motivation behind our novel experiment was to assess how rendering this game in a computer generated VE displayed in fMRI would alter the experience of the game. Furthermore, the game was designed with both low and high anthropomorphism avatars in order to investigate the relationship between avatar fidelity and character believability.

1.1 Contributions

The general aim of this work was to create a 3D interactive gaming paradigm for the secluded space of an fMRI scanner. The goal was to experimentally explore changes in regional brain activity associated with the pain of social exclusion as well as with feelings such as the empathy felt when people observe other people get socially excluded and whether there are differences in relation to empathy for friends and strangers. The term social exclusion involves the lack of resources, rights, goods and services, and the inability to participate in the normal relationships and activities, available to the majority of people in a society. Many people face social exclusion in their daily life as well as other people empathizing with excluded people **(Levitas et al 2007)**.

In addition, this research investigated whether the level of anthropomorphism of the avatars may affect game playing as well as fMRI data acquired at the same time the game playing occurs. The goal was to discover the neural circuitry that supports such feelings as well as, for the first time, devised behavioral fidelity metrics of character believability and emotional engagement based on neural activity.

The Cyberball3D+ game presented here involved four players represented by 3D characters of different levels of anthropomorphism. The main player played the game while immersed in the fMRI scanner interacting with three 3D virtual characters for which their player behavior was programmed. The scenarios evaluated were either fair or unfair to the players simulating social exclusion. The system monitored the main player's reaction to the occluded players, e.g. players not receiving the ball in the game, by potentially throwing more balls to them because of empathy for social exclusion. The neuroscientists defined the fMRI gaming scenarios using a dedicated user interface in order to select the level of anthropomorphism applied to all characters, the gender of each character and the fairness of the game.

The contributions of this thesis are:

1. **For the first time,** the Cyberball game was implemented in three - dimensional form as well as it included three levels of anthropomorphism.
2. The neuroscientists were able to simulate any situation of the fairness of the game they wanted by filling in the percentages of the tosses that each player would throw to each one of the other players.
3. The innovative application designed to render an interactive Virtual Environment (VE) on an fMRI display, enabling the conduct of formal neuroscientific experiments and investigating the effects of social exclusion, empathy and different level of anthropomorphism on human brain activity.
4. **For the first time**, we got a validated neuroscientific measure of character believability and emotional engagement at the same time the experience was taking place.

Moreover, it is not straightforward to provide interactive synthetic stimuli to be displayed in fMRI displays due to the infrastructural and technical demands. fMRI experiments of this type usually employ simple display material, for example using photographs, video clips or simple computerized stimuli which are non-stereoscopic. Using VEs in fMRI has the advantage that it is possible to involve participants in interactive animated environments which more realistically reflect social and emotional situations. This seamless naturalism and interactivity is impossible to achieve with video clips. In addition, some experiments could be only conducted using synthetic stimuli for ethical reasons.

A broader aim of this work is to assess whether such powerful social-psychological studies could be usefully carried out within VEs advancing both cognitive neuroscience and computer graphics research.

1.2 Thesis Outline

This thesis is divided into a number of chapters, which will be outlined below.

Chapter 2 – Technical Background: This chapter analyzes the levels of anthropomorphism in computer graphics, the effect of the Uncanny Valley and the responses of participants that were elicited from the body motions and the facial expressions of the avatars. In addition, describes methods that were used by researchers in order to model and animate a 3d avatar as well as success real and fast rendering of the 3d avatar.

Furthermore, this chapter provides background information regarding the virtual games were used inside the fMRI scanner by neuroscientists in order to investigate social phenomena such as social exclusion and empathy.

Chapter 3 – Software Architecture and Development Framework: In this chapter, the technical requirements of the interactive 3D gaming system are introduced as well as the architecture of the application developed for the experiments is presented, along with the inherent architecture of the Unreal Development Kit (UDK) used to develop it.

Chapter 4 – Implementation: Chapter 4 describes in detail the implementation of the interactive 3d gaming framework. Specifically, examples and source code samples are demonstrated, in relation to analyze how the application met the requirements imposed by the expert psychiatrists in order to incorporate a formal neuroscientific protocol for the fMRI scanner.

Chapter 5 – UI Implementation: In this chapter, the implementation of the User Interfaces that were presented to the users in the fMRI scanner is described. The steps taken to create the individual UIs as Flash applications and embed them in the complete systems are presented as well. The challenges concerning the creation of interactive synthetic worlds and associated UIs as displayed in the fMRI scanner are presented and the solutions to overcome them are also explained.

Chapter 6 – Experiments: This chapter is concerned with the experimental methods employed when the actual experiments were conducted in the fMRI scanner. The experimental procedures are presented as well as the results of the experiments.

Chapter 7 – Conclusions: In the final chapter, the conclusions of this thesis are presented as well as hints about future work.

2 Chapter 2 – Technical Background

A Virtual Environment (VE) is a computer simulated scene which can be interactively manipulated by users as shown in Figure 1. Typical scenes used in such environments generally comprise of geometric and animating 3D models, shades, images and lights which are converted into final images through the rendering process. The rendering process must be completed in real time in order to provide scenes which are updated reacting to user interaction. An Immersive Virtual Environment (IVE) perpetually surrounds the user within the VE. IVEs are virtual environments that present rich layers of synthesized sensory cues to the user so that the user feels enveloped by the mediated environment and is willing to believe that the environment is real (**Witmer and Singer 1998**).

The immersive virtual environments (IVEs) were first introduced as a research tool in the late 1980s by psychologists conducting studies on perception and spatial cognition. Social psychologists explored the incorporation of virtual humans within IVEs to examine the possibility of using virtual reality technology as a research tool to answer social scientific questions **(Loomis, Blascovich and Beall 1999)**. The Immersive Virtual Environments permit the researcher to design experimental situations with more ordinary realism compared to the rigidly controlled laboratory settings, provoking more genuine participant reactions to the stimuli **(Sun, Fox, and Bailenson 2012)**.

Figure 1: Immersive virtual environment technology as a tool for social psychology. The source of the image is from the Hank Virtual Environments Lab.

The first section of this chapter analyzes the humanoid avatars in VEs. The second section presents the technology of the 3d avatars, while the third section proceeds by

describing previous researches that they used 3d games to explore neuroscientific phenomena. Finally, the fourth section presents experiments that were conducted for the investigation of the social exclusion and empathy.

2.1 Humanoid Avatars in VEs

A **humanoid** object is something that has a human appearance. Particularly, this term can refer to anything with uniquely human characteristics and/or adaptations, such as binocular vision (having two eyes), possessing opposable appendage (thumbs), or biomechanic bipedalism (the ability to walk in an upright position).

On the other side an avatar can be considered any form of representation that marks a user's entity. For example a name, a voice, a photograph or a top cap that is utilized as a part of a game, all these can be served as a user's avatar despite the fact that they may not look or act like the user **(Bailenson et al. 2008).** Diverse qualities such as the level of anthropomorphism (i.e., how the symbol looks like a human user) and behavioral realism (i.e., how the symbol acts like a human user) impacts how someone else sees and reacts to an avatar **(Blascovich et al 2002).**

Over time, avatars have become more complex creations, rendered in three dimensional forms with a vast range of animated movements that help in the expression of the avatar's personality and supplement various social interactions. The options for individual customization of avatars have increased significantly as well, allowing users to change a number of physical features including face shape, hair style, eye color, height, body shape, clothing, and even facial expressions. Using these diverse features, users are free to design not just a graphical marker of themselves, but *virtual humans* with unique appearances, distinctive personalities and individualized behavioral patterns.

3D human avatars (characters) are used in the film and game industry but it still remains undetermined how design, stylization and behavioral factors interweave to make a character believable. When using synthetic characters in experiments simulating realistic scenarios, the aim is that people emotionally respond to them in a similar manner to humans. It has been shown that there is no significant difference in relation to skin conductance response (SCR) scores when comparing psychological activity for certain events between scenarios involving either real actors or animated characters **(Ekanayake et al.**

2013). Although research has shown that users show empathy for 3D characters, it is challenging to identify at which level of anthropomorphism such emotional experiences occur.

2.1.1 Scale of Anthropomorphism and Uncanny Valley

Anthropomorphism can be characterized as far as either behavior or appearance. In behavioral terms, anthropomorphism suggests the task of human qualities such as mental capacities and behavior to objects that are not human **(DiSalvo & Gemperle 2003)**, while in appearance terms, anthropomorphism defines an object that has human appearance or visual characteristics **(DiSalvo & Gemperle 2003; Nowak 2003; Nowak & Rauh 2005; Shapiro 1997)**.

More generally, anthropomorphism refers to the attribution of a human form and behavior to non-human entities such as robots, animals, etc. However, uncanny valley effects – ie dips in user impressions – can arise: behavioural fidelity expectations increase alongside increases in visual fidelity and vice versa. Moreover, the influence of anthropomorphism and perceived agency on presence, copresence, and social presence in a Virtual Environment (VE) **(Nowak and Biocca 2003)** has been investigated. . During the experimental procedure they used three levels of anthropomorphism: high anthropomorphism, low anthropomorphism and no image. The results have depicted that the participants interacting with the less-anthropomorphic characters reported a higher level of presence and social presence than those interacting with either no image at all or with a highly anthropomorphic image. This indicates that high level anthropomorphic images set up higher expectations that lead to reduced presence when these expectations were not met.

The androids that were created by Hiroshi Ishiguro, for a short period, were indistinguishable from human beings **(Ishiguro H. 2005)**. These highly anthropomorphic androids struggle with so-called 'uncanny valley', a theory that defines that as a robot is made more human like in its appearance and movements, the emotional responses from a human being to the robot becomes increasingly positive and empathic, until a point is reached beyond which the response quickly becomes that of intense repulsion. However, as the appearance and movements continue to become less distinguishable from those of a human being, the emotional responses become positive once more and approaches human - to - human empathy levels.

An interesting behavioral measurement for anthropomorphism has been presented by **(Minato et al. 2005)** analyzing the differences between the reactions of the participants to a human and an android. Also, another experiment conducted by **(Gong 2007)** where participants manipulated 12 computer agents that are represented by four levels of anthropomorphism: low, medium, high and real human images. The results have shown that as the agent became more anthropomorphic, it received more social responses from the users.

The Uncanny Valley (UV) theory as described above indicates that near-photorealistic virtual humans often appear unintentionally eerie or creepy. This UV theory was first hypothesized by a robotics professor in the 1970s. He observed that as a robots come to look more human like, they seem more familiar, until a point is reached at which subtle deviations from human norms cause them to look creepy **(Mori 1970)**. Particularly, he hypothesized that the familiarity increases with human likeness until an uncanny valley is reached caused by sensitivity to perceived imperfections in near-humanlike forms as shown in Figure 2. A study by **(MacDorman 2006)** indicated that the perceived human likeness of a robot is not the only factor determining the perceived familiarity, strangeness, or eeriness of the robot. In a study of **(Seyama and Nagayama 2007)** the uncanny valley was investigated by measuring observers' impressions of facial images whose degree of realism was manipulated by morphing between artificial and real human faces. The results suggested that to have an almost perfectly realistic human appearance is a necessary but not a sufficient condition for the uncanny valley. Additionally, their findings showed that the uncanny valley emerges only when there is an abnormal feature. A series of psychophysical experiments were performed by **(McDonnell, Breidt and Bülthoff 2012)** aimed to investigate if using realistic rendering does in fact produce a more negative response than using lower quality or stylized rendering. Particularly, in the experiments a range of 11 different render styles are applied to identical geometry and motion pairs. The outcomes demonstrated that negative reactions occurred mainly for characters that used human texture maps, but that were not rendered with realistic eye and skin shaders. Cartoon characters were considered highly appealing, and were rated as more pleasant than characters with human appearance, when large motion artifacts were presented. They also were rated as friendlier than realistic styles and in this manner they might were more suitable for certain virtual interactions.

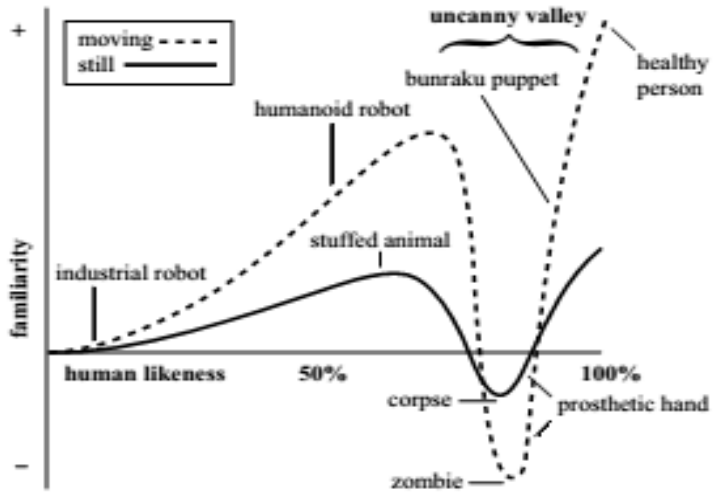


Figure 2: The relation between human likeness and perceived familiarity based on the hypothesis of Mori (Mori 1970).

In addition, (**McDonnell and Breidt 2010)** carried out more experiments about rendering styles in order to investigate if changing the rendering style of a virtual human alone can change how trustworthy they are perceived to be. The results of the experiments indicated that participants judged HQ (high quality style) to be lying more often than NPR (non – photorealistic rendering style) during the task. However, all styles were judged as equally trustworthy in the qualitative rating, which imply that a subconscious feeling of un-trustworthiness was felt by participant towards HQ.

In this work, we examined the neural responses of the social exclusion based on the different levels of anthropomorphism. Three levels of anthropomorphism (low, medium and high) were employed and in each round of our experimental game the appropriate 3D characters of the scene were displayed. Our results showed that compared to more human like avatars, playing the non-anthropomorphic avatars is less subjectively rewarding. Therefore, when studying complex emotional responses, a high level of anthropomorphism of synthetic characters is not only required but also able to engage common neuroscientific patterns of brain activation as in real-world circumstances.

Figure 3: Screenshots from three different levels of anthropomorphism. The source of the first image is from a virtual-storytelling experiment.

2.1.2 Animations and Facial Expressions

Humans express their emotions in many ways, in particular through face, eye and body motion. Therefore, the creators of virtual humans strive to convincingly depict emotional movements using a variety of methods. The human face is capable of producing a large variety of facial expressions that supply important information for the communication. Realistic computer animated faces were used by **(Griesser et al. 2007)** to investigate the spatiotemporal coactions of facial movements. Single regions (mouth, eyes, and eyebrows) of a computer animated face performing seven basic facial expressions were selected as well as combinations of these regions, were animated for each of the seven chosen facial expressions. In their experiments the participants were asked to recognize these animated expressions. The results showed that the animation system is good enough to support recognition of some of the computer animated avatar's facial expression with high accuracy.

A study of **(Curio et al. 2008)** presented the first study on high – level – after – effects in the recognition of dynamic facial expressions (Figure 4). In order to be analyzed the emotional content of motions portrayed by different characters, **(McDonnell et al. 2008)** created real and virtual replicas of an actor exhibiting six basic emotions: sadness, happiness, surprise, fear, anger and disgust. During the experiments participants were asked to rate the actions based on a list of 41 more complex emotions. The results showed that the perception of emotional actions was highly robust and to the most part independent of the character's body. In another study of **(McDonnell et al. 2009)** were found that both form and motion influence sex perception of virtual characters: for neutral synthetic motions, form determines perceived sex, whereas natural motion affects the perceived sex of both androgynous and realistic forms. A second investigation into the influence of body shape

and motion on realistic male and female models showed that adding stereotypical indicators of sex to body shapes influenced sex perception. Exaggerated female body shapes influences sex judgments more than exaggerated male shapes.

Figure 4: Examples of facial expressions retargeted from Motion Capture onto a morphable 3D face model (Curio et al. 2008).

2.1.3 Method for Rendering Real - Time Animated Crowds in VEs

It is challenging for the developers to simulate heterogeneous crowds. The assets required to vary humans, such as textures and accessories, can be expensive to purchase, time-consuming to create and require extensive memory and computing resources. In the study of **(McDonnell, Larkin, Hernández, Rudomin & O'Sullivan 2009)** they addressed these issues by developing a selective variation method for virtual humans. They investigated which body parts of virtual characters were most looked at in scenes containing duplicate characters or clones. Using an eye – tracking device, they recorded fixations on body parts while participants were asked to indicate whether clones were present or not. The outcomes indicated that the head and upper torso attract the majority of first fixations in a scene and were attended to most. This is true regardless of the orientation, presence or absence of motion, sex, age, size, and clothing style of the character. They also, developed a selective variation method to exploit this knowledge and perceptually validated their method. The results showed that selective color variation is as effective at generating the illusion of variety as full color variation as well as the head accessories, top texture and face texture variation are all equally effective at creating variety, whereas facial geometry alterations are less so.

2.2 Modeling and Animating 3D Avatars for VE and Games

Virtual characters in animated movies and games can be very expressive and have the ability to convey complex emotions. However, it has been suggested that the more anthropomorphic a humanoid robot or virtual character becomes the more eerie it is

perceived to be **(Mori 1970).** From a neuroscientific point of view some studies suggest that different neural networks are activated when viewing real or virtual stimuli **(Perani et al. 2001)**. It was founded by **(McDonnell et al. 2008)** that when realistic human body motion is used, it is the motion and not the body representation that dominates our perception of portrayed emotion. Also, she found that there is no difference between the emotion ratings for high and low resolution virtual models as well as between the emotion ratings for the zombie and the real human or the other characters.

An approach for the learning of structured dynamical models based on hierarchical Gaussian process latent variable models was presented by **(Taubert et al. 2012)**. The latent spaces of this model encoded postural manifolds and the dependency between the postures of the interacting characters. Their findings showed that the highly-realistic interactive movements were almost indistinguishable from natural ones. They also, supported that the synthesis of stylized interactive movements with high levels of realism is a difficult problem in computer graphics, especially for the developers that they have to convincingly depict emotional movements to virtual humans. The principles of the human motion connect many areas of the research such as the biomechanic, optimal control, machine learning, robotics, motor neuroscience and psychology. Each of these areas can give a different perspective on motion, helping the developers to understand how the human moves.

2.2.1 Methods for Modeling a 3D Avatar

The 3D modeling of the human body has many applications ranging from fashion to the production of movies and video games. The designing of the human body shape is very challenging compared to other 3D objects in the world **(Aggeliki Tsoli 2014)**. Although the human body consists of many parts that move in a rigid way, there are also many ways in which the human shape deforms non-rigidly; e.g. different muscles are flexed when a person is standing compared to when the same person is sitting, deformations due to breathing lead to a constantly changing body shape even when the person remains seated. Additionally, she found that the distinction of many human attributes ranging from age, gender to the emotional state of a person depends on the human body shape.

Several graphics approaches for modeling and generating human characters have taken an inside-out approach and relied on complex anatomically inspired, physically based models of the human body **(Glueck et al. 2012; Magnenat-Thalmann, Zhang and Feng 2009).** One of the first endeavors to model full-body skin deformations utilizing surface data was employed by **(Park and Hodgins 2006)** using a marker-based capture system with a

large number of markers distributed over the body of the subject. The outcomes were encouraging, yet the methodology experienced drawn out instrumentation of the subject and couldn't capture deformations of the whole surface of the human body. Many alternative approaches such as high-resolution 3D scanners, image-based 3D reconstruction systems, and low-resolution 3D capture systems offer to developers the ability to capture the whole surface of the body, potentially across time, and generalize easily to capturing different human shapes.

A way to present an articulated 3D deformable object by using a 3D triangular mesh was described by **(Aggeliki Tsoli 2014)**. The object that she used was determined mainly by three factors: pose, intrinsic shape, resolution. Particularly, the *resolution* referred to the number of vertices in the 3D mesh representing an articulated 3D object, while the *pose* referred to the relative joint rotations in the underlying skeleton or the rotation and translation of groups of triangles and finally the *Intrinsic shape* typically referred to the pose-independent shape, such as shape related to height and weight.

It is worth noting that **a polygon is a 3D structure consisting of three or more points called vertices that are connected in 3D space with lines called edges. The smallest polygon is simply a triangle serving as the basic unit of measurement for a 3D model. Several polygons together complete a mesh object that can be deformed and animated. A target polygon count refers to the target face count of a model.**

Another approach for the creation of a 3d human avatar by using Kinect was followed by **(Aitpayev and Gaber 2012)**. Particularly, their results suggest that it is very easy using Kinect to create one's own avatar as well as the whole process takes a few minutes by doing the following steps. At the beginning a camera scans the whole body; afterwards it scans the face and makes snapshots of the head to reconstruct the hairstyle. At the last the user chooses clothes and other accessories which will be applied to the final model. However, they reported that the hardest task of avatar creation is the reconstruction of facial geometry. In addition, researchers **from the Computer Graphics Group in Erlangen** have presented a system for generating 3D face avatars using the Kinect. A generic face model is fitted to the depth map obtained by the Kinect. The resulting 3D face model is finally textured using the captured RGB image. The proposed algorithm combines the advantages of robust no rigid registration and fitting of a morphable face model. Their results showed that it is possible to obtain a high quality reconstruction of the facial geometry and texture along with one-to-one correspondences with generic face model as illustrated.

An interesting finding was presented by **(Zhang et al. 2013)** that it suggests that there are mostly two categories, the *Sketch-based* and the *Gesture-based,* that are used by novices to create 3d models. The *Sketch-based* is a common approach to interactively interpret the user's 2D sketches into 3D shapes, while the *Gesture-based* is a method that allows the user to construct and manipulate 3D shapes using hand gestures performed in 3D space, often based on a virtual sculpting metaphor. Particularly, two paradigms where the *Sketch-based* method was used are the SKETCH **(Zeleznik, Herndon & Hughes 1996)** that supported constructing 3D scenes by sketching geometric primitives and the Teddy **(Igarashi et al. 1999)** that let the user to create rotund 3D objects by drawing their 2D silhouettes, as well as supporting operations like extrusion, cutting, and bending as shown in Figure 5.

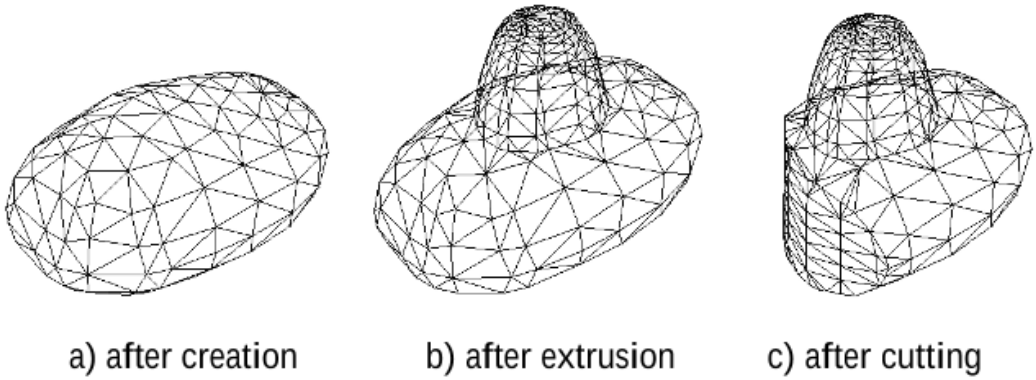


Figure 5: The use of the creation, extrusion and cutting operations for the modeling of a 3D object (Igarashi et al. 1999).

Contrary, two researches where the *Gesture-based m*ethod was used are the project of (**Nishino, Utsumiya & Korida 1998)** in which he used two-handed spatial and pictographic gestures and the project of **(Schkolne, Pruett & Schröder 2001)** that is called *Surface Drawing* in which he used repeated marking of the hand to construct 3D shapes.

In this work, we used the *Sketch-based* method because we found that it is the simplest and easiest method to create and model a 3D avatar. Particularly, for the modeling of the lowest fidelity 3d avatar two sketches were used that were depicted the front and the right view of a human silhouette, respectively. These sketches were imported and positioned on the scene of the 3ds Studio Max so as to form a vertical angle. Using the Editable Poly Modifier tool of the 3ds Studio Max we started to create small polygons and connect them to each other based on these two sketches until a human silhouette be modeled.

2.2.2 Methods for Animating a 3D Avatar

Human realistic facial and body animation remains a challenging goal in computer graphics and animation research. However, the success of human dynamic realism does not exclusively depend from computer graphics and animation research, but there is also a strong dependency on what we can learn from psychophysical and perceptual experimentation with faces and bodies. For example, unobtrusive varieties in articulation timing **(Krumhuber et al. 2007)** and even a slight dampening of element movement **(Theobald et al. 2009)** have been demonstrated to firmly impact how declarations and people are seen. Examination utilizing dynamic facial expressions as a part of software engineering and brain research is generally centered on facial models with control parameters in view of the Facial Action Coding System (FACS) **(Ekman et al. 2002).** FACS gives definite depictions of 44 facial activities – termed Action Units (AUs) – which endeavor to incorporate the fundamental set of unique facial developments proficient by a face. This gives to the scientists a standard for coding, evaluating and imparting results. Given such a model, experimenters may control these parameters to gauge the perceptual impact of AU varieties in a controlled way. FACS is additionally utilized widely as a premise for activity frameworks in feature recreations and films **(Sagar 2006; Duncan 2009)**. Henceforth, it has a noteworthy part to play in both facial movement and perception research.

The first Facial Action Coding System (FACS) was introduced by **(Cosker, Krumhuber & Hilton 2010)** and is a substantial model to be based on dynamic 3D scans of human faces for use in graphics and mental exploration. The model comprised of FACS Action Unit (AU) based parameters and has been freely accepted by FACS specialists. Utilizing this model, they investigated the perceptual contrasts between linear facial motions – represented by a linear blend shape approach – and real facial motions that have been synthesized through the 3D facial model. In their experiments, they also investigated the perceptual profits of nonlinear movement for different AUs. Their outcomes are insightful for designers of animation systems both in the entertainment industry and in the scientific research. They uncovered a critical general advantage to utilize captured nonlinear geometric vertex movement over linear mix shape motion. However, their findings suggested that not all motions need to be animated nonlinearly.

The first parameterized face model was designed by **(Parke and Waters 1996)** in 1974, with the objective of creating facial movements rapidly. Utilizing photogrammetric strategies, they gathered 3D information from real faces and created animations by interpolating between the facial expressions. A muscle-based activity framework was

showed by **(Sifakis et al. 2005)** where he used motion capture data in a non-direct improvement procedure to gauge facial muscle actuation parameters. A different approach was conducted by **(Blanz and Vetter 1999)** building up an algorithm that fitted a blend shape model onto a single picture, bringing about an estimation of the geometry and surface of the individual's face. In 2005, Joshi proposed a programmed physically roused division that took in the controls and parameters straightforwardly from the set of blendshapes **(Joshi et al. 2005)**. Recent movement frameworks determined facial movements in three-dimensional space by following markers appended to a person's face. A model in 2003 was displayed by **(Breidt et al. 2003)** consolidating 3D scans and motion capture data for highly realistic facial animation. Another model for multimodal complex feelings including motion expressivity and blended facial expressions was exhibited by **(Martin et al. 2006)**.

An interesting system for realistic facial animation was presented by **(Curio et al 2006)** where he decomposed facial motion capture data into semantically meaningful motion channels based on the Facial Action Coding System. They retargeted a captured performance onto a morphable 3D face model based on a semantic correspondence between motion capture and 3D scan data. The resulting facial animation uncovered a high level of realism by combining the high spatial resolution of a 3D scanner with the high temporal exactness of motion capture data that accounts for subtle facial movements with inadequate estimations.

In technical terms “Motion capture” (Mocap) is sampling and recording motion of humans, animals, and inanimate objects as 3D data", but in simple terms the Motion Capture is defined as “Recording of motion and playback” OR "One way of acting out an animation". Particularly, during the first phase of the Mocap the movements of the objects are captured in the real world and then in the second phase the data of the captured movement are inserted into a 3D model of the world in a virtual environment.

Motion capture is used as an analysis tool in biomechanics research, as well as in education, training and sports and recently in both cinema and video games. Motion capture is broadly utilized procedure as a part of the Film making. Now day’s motion capture is utilized as a part of movies to record the actors and proprietary software to animate the creatures and fight scenes. These animations made through software were constantly subjected to synchronization with the Mocap actor’s movements and also between the digital creatures, in order to create a believable battle or dialog scene (Figure 6).

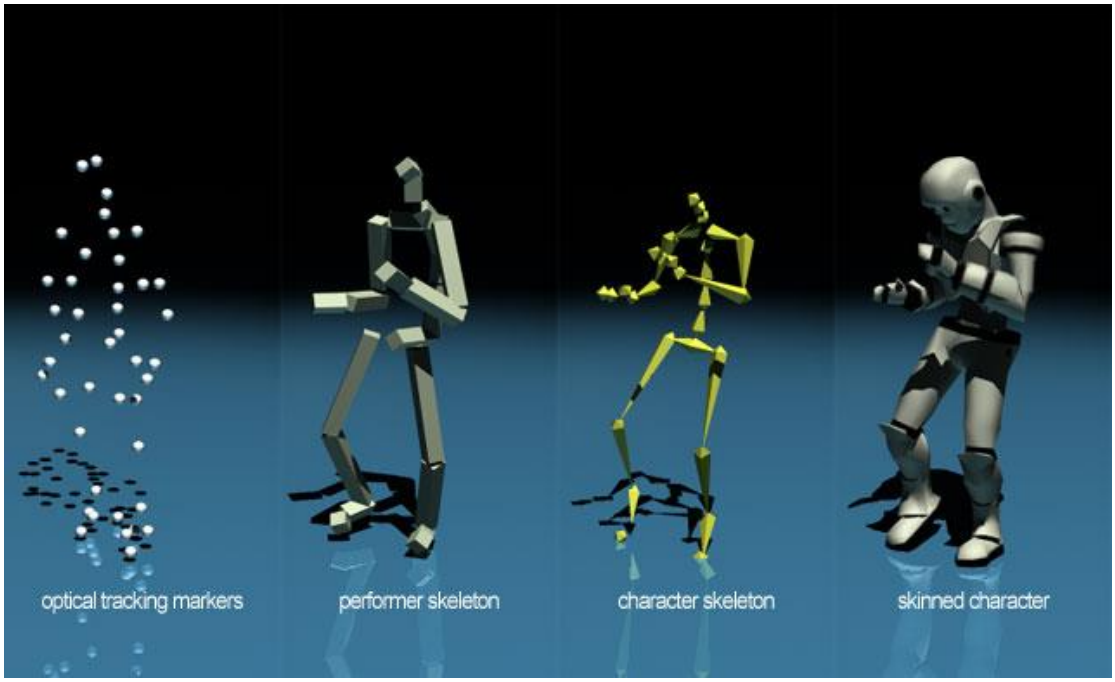


Figure 6: The use of the Fastmocap technology with markerless 3D motion capture with Microsoft kinect sensor for the creation of the animations of a 3D avatar.

In this work, motion capture data were not utilized, however, we tried the movement to be as natural as possible. Particularly, we used the Physique Modifier tool of the 3ds Studio Max for the modeling of the animations of the characters as well as for the association of the characters mesh with the biped's skeletal parts. This was accomplished through a process called skinning. Firstly, we applied the Physique Modifier to the characters and secondly we created the movements of the avatars using the *Auto-Key* and the *Animation Timeline* tools of the 3ds Studio Max. Using the *Auto Key* tool, when a change of objects' position, rotation, and scale happens the 3ds Max Studio creates a key storing the new value for the changed parameter at the current time. The 3ds Max Studio automatically fills in the frames between the different keys.

The process that was followed for the animation of the 3d avatars is described, analytically in the fourth chapter of this thesis.

2.3 Using VEs and Games for Neuroscientific Research

A broader aim of many works was to assess whether such powerful social-psychological studies could be usefully carried out within VEs advancing cognitive neuroscience and computer graphics as well as serious gaming research. It would be

valuable to understand the cognitive processes involved while interacting with 3D characters in a gaming scenario. 3D characters could be employed to simulate experimental scenarios as part of neuroscientific protocols in the fMRI. An economic game combined with Milgram's original experimental scenario has been implemented which, in the future, could be interactively played in an fMRI scanner **(Rivera et al. 2010).**

A lighting system has been designed to render an interactive VE on an fMRI display, enabling the conduct of formal neuroscientific experiments and investigating the effects of visual fidelity as well as varied lighting configurations of an indoors/outdoors space on feelings of presence, 'reality' and comfort **(Christodoulou et al. 2012)**. Another 3d virtual system for fMRI has been designed investigating the influence of two prominent VR parameters, e.g. 3d-motion and interactivity, while brain activity is measured for a mental rotation task **(Sjölie et al 2010)**. The subjects performed a variation of the mental rotation task in a simple VE and brain activity was recorded during three conditions of varied 3d-motion and interactivity. Also, an interactive digital game based on the Re-Mission video game has been designed for cancer patients in the fMRI in order to determine whether mesolimbic neural circuits associated with incentive motivation are activated, and if so, whether such effects stem from the participatory aspects of interactive gameplay, or from the complex sensory/perceptual engagement generated by its dynamic event-stream **(Cole, Yoo and Knutson 2012)**.

The Smart Ageing that was implemented by **(Tost et al 2014)** is a serious game in a 3D virtual environment aimed at the early detection of Mild Cognitive Impairments in persons ageing between 50 and 60, and at assessing cognitive impairments in persons already diagnosed or having neurodegenerative dementia. Smart Ageing is a telematic system where the users realize a set of screening tests structured in five daily-life tasks in a familiar environment addressing various cognitive skills: memory, executive functions, divided attention, short-term and long-term memory and spatial orientation and attention. The Trauma Treatment Game that was designed by **(Mayr, Horleinsberger** & **Petta 2014)** is another serious game specifically designed to provide individualized interventions to children suffering from complex trauma and comorbid disorders such as anxiety and depression to have undergone rigorous clinical evaluation.

In the experiments of **(Bhatt & Camerer 2005)** 16 subjects brain activity were scanned using fMRI as they made choices expressed beliefs and expressed iterated 2^{nd}-order beliefs (what they thought others believed they would do) in eight games. Cingulate cortex and prefrontal areas were differentially activated in making choices versus expressing

beliefs. Forming self-referential 2nd-order beliefs about what others thought you would do seemed to be a mixture of processes used to make choices and form beliefs. A similar work was presented by **(Yoshida et al. 2010)** developing a "stag hunt" game where human subjects interacted with a computerized agent using different degrees of sophistication (recursive inferences) and applied an ecologically valid computational model of dynamic belief inference. They showed that rostral medial prefrontal (paracingulate) cortex, a brain region consistently identified in psychological tasks requiring mentalizing, had a specific role in encoding the uncertainty of inference about the other's strategy.

An interesting research was conducted by **(Mathiak & Weber 2006).** Particularly, they recorded 13 experienced gamers (18–26 years) while playing a violent first-person shooter game (a violent computer game played in self-perspective) by means of distortion and dephasing reduced fMRI. They found that the occurrence of violent scenes revealed significant neuronal correlates in an event-related design. They also, proposed that virtual environments can be used to study neuronal processes involved in semi-naturalistic behavior as determined by content analysis.

On the contrary**, (Wang, Sourina and Nguyen 2011)** claimed that the EEG-based technology has become more popular in "serious" games designs and developments since new wireless headsets that meet consumer demand for wearability, price, portability and ease-of-use are coming to the market. Originally, EEG-based technologies were used in neurofeedback games and brain-computer interfaces. They conducted experiments efficiency of fractal dimension value and the results showed that the brain states were recognizable with the difference in fractal dimension value. Two EEG-based games "Brain Chi" and "Dancing Robot" where designed and implemented for concentration training. For more information, Electroencephalogram (*EEG*) is a noninvasive technique that allows recording the electrical potentials over the scalp which are produced by activities of brain cortex and reflect the state of the brain **(Nunez & Srinivasan 2006).** Neurofeedback is a technique that presents the real-time feedback to the user in the form of video display and/or sound based on the processing results of EEG signals taken from the scalp of the user **(Hammond 2007).**

It is claimed that using VEs and 3D characters in fMRI has the advantage that it is possible to involve participants in interactive animated environments which more realistically reflect social and emotional situations. In this work we present an interactive 3D gaming framework for fMRI experiments exploring whether artificial characters of varied anthropomorphism recruit brain activation associated with empathy and social pain **(Meyer**

et al. 2012). In addition, the goal of this work was to investigate whether the level of anthropomorphism of the avatars might affect game playing as well as fMRI data acquired at the same time the game playing occurred. **For the first time**, using this study devised behavioral fidelity metrics of character believability and emotional engagement based on neural activity.

2.4 The Examination of Social Exclusion and Empathy

The term 'social exclusion' first originated in Europe, where there has tended to be a greater emphasis on spatial exclusion. There is also a policy focus on those living in 'deprived areas', where poor housing, inadequate social services, weak political voice and lack of decent work all combine to create an experience of marginalization. However, there are various definitions of social exclusion. **(Walker and Walker 1997)** mentioned that the social exclusion is the dynamic process of being shut out from any of the social, economic, political and cultural systems which determine the social integration of a person in society. **(Burchardt et al, 2002)** referred that an individual is socially excluded if (a) he or she is geographically resident in a society but (b) for reasons beyond his or her control, he or she cannot participate in the normal activities of citizens in that society, and (c) he or she would like to so participate. In addition, **(Levitas et al. 2007)** claimed that the social exclusion is a complex and multi-dimensional process. Particularly, it involves the lack of resources, rights, goods and services, and the inability to participate in the normal relationships and activities, available to the majority of people in a society, whether in economic, social, cultural or political arenas. It affects both the quality of life of individuals and the equity and cohesion of society as a whole.

Many people face social exclusion in their daily life as well as other people empathizing with excluded people. Empathy is a complex form of psychological inference in which observation, memory, knowledge, and reasoning are combined to yield insights into the thoughts and feelings of others **(Ickes 1997)**. Behavioral research has suggested that empathy includes two primary components **(Davis 1983)**: (1) an affective component that involves sharing the emotional experiences of others, and (2) a cognitive component that involves thinking about, understanding and predicting some-one else's mental state.

The neuroscientists have shown strong interesting for these social phenomena and their effects on human's life as well as for the brain activation that occurs when people get social exclusion or empathize with someone else that gets exclusion. In the next subsections

we describe some researches that investigated the social exclusion and empathy feeling using fMRI scanner. In addition, in this study we used reinforcement learning method to examine the participant's behavior when the rewards were included in the game. Particularly, we wanted to investigate the differences in the behavior of the participants to the other players when they have the ability to get points in the case of the use of the reinforcement learning method in the game and when they have not any benefit to play with specific way in the case of the use of the standard scenes of the game. Therefore, the goal of this study was to explore if the participants included the excluded player in the game because they were feeling empathy or they wanted to win the game. The definition of the reinforcement learning method as well as some reinforcement learning algorithms are described analytically in a next subsection as well.

2.4.1 The Investigation of the Social Exclusion on Human's Life

In the experiments of **(Moor et al. 2012)** participants that were from 3 age groups (10–12, 14–16 and 19–21 year olds) participated in two tasks; first, they played the Cyberball game in order to induce feelings of social inclusion and exclusion, followed by a Dictator game where they were asked to divide coins between themselves and the players who previously included or excluded them. Results revealed that although social exclusion generated strong distress for all age groups, 10–12 year olds showed increased activity in the subgenual ACC in the exclusion game, which has been associated in previous studies with negative affective processing. Results of the Dictator game revealed that all age groups selectively punished the excluders by making lower offers. These offers were associated with activation in the temporoparietal junction (TPJ), superior temporal sulcus (STS) and the lateral PFC. Age comparisons revealed that adults showed additional activity in the insula and dorsal ACC when making offers to the excluders.

A similar research was conducted by **(Masten et al. 2009)** investigating the neural correlates of social exclusion during adolescents. Their findings showed that the adolescents with higher rejection sensitivity and interpersonal competence scores displayed greater neural evidence of emotional distress, and adolescents with higher interpersonal competence scores also displayed greater neural evidence of regulation, perhaps suggesting that adolescents who are vigilant regarding peer acceptance may be most sensitive to rejection experiences. Furthermore, the same authors **(Masten et al 2011)** conducted another block of experiments exploring the neural responses to peer exclusion among adolescents with ASD compared to typically developing adolescents. The results of their experiments showed that compared to typically developing adolescents, those with ASD

displayed less activity in regions previously linked with the distressing aspect of peer exclusion, including the subgenual anterior cingulate and anterior insula, as well as less activity in regions previously linked with the regulation of distress responses during peer exclusion, including the ventrolateral prefrontal cortex and ventral striatum. Interestingly, however, both groups self-reported equivalent levels of distress. This suggested that adolescents with ASD might engage in differential processing of social experiences at the neural level, but be equally aware of, and concerned about, peer rejection.

The fMRI additionally, was employed by **(Krill and Platek 2009)** to examine the sensitivity to social exclusion in three conditions: same-race, other-race, and self-resembling faces. In his experiments, the results showed that the participants demonstrated greatest ACC activation when being excluded by self-resembling and same-race faces, relative to other-race faces. Additionally, participants expressed greater distress and showed increased ACC activation as a result of exclusion in the same-race condition relative to the other-race condition.

They have been conducted many researches for the social situations as we described above, however the study of **Maurage in 2012** identified for the first time the cerebral correlates of interpersonal alterations in alcohol-dependence **(Maurage et al. 2012)**. In his experiments participants played the Cyberball game where they were first included by other players, then excluded, and finally re-included. The results showed that while both groups presented dorsal anterior cingulate cortex (dACC) activations during social exclusion, alcohol-dependent participants exhibited increased insula and reduced frontal activations (in ventrolateral prefrontal cortex) as compared with controls. In addition, the Alcohol-dependence was linked with increased activation in areas eliciting social exclusion feelings (dACC–insula), and with impaired ability to inhibit these feelings (indexed by reduced frontal activations).

In this study, we designed an interactive Virtual Environment (VE) on an fMRI display, enabling the conduct of formal neuroscientific experiments and investigating the effects of social exclusion, empathy and different level of anthropomorphism on human brain activity. For the investigation of the social exclusion we implemented several scenarios of the game simulating the inclusion or the exclusion of the human participants as well as the exclusion of other virtual players of the game. Our results presented that T-Contrast estimates showing main effect of exclusion of other versus inclusion, demonstrating activity in emotional brain regions such as bilateral parahippocampal, left superior and left middle frontal gyrus as well as in anterior cingulate and amygdala. The anterior cingulate cortex can

be partitioned anatomically taking into account cognitive (dorsal) and emotional (ventral) components. The dorsal part of the ACC is associated with the prefrontal cortex and parietal cortex and also the motor system and the frontal eye fields making it a central station for processing top-down and bottom-up stimuli and assigning appropriate control to other areas in the brain. fMRI studies have shown that the dorsal anterior cingulate cortex (dACC) and insula activations reflect the negative feelings and distress associated with social exclusion as well as the middle frontal gyrus (MFG) and inferior frontal gyrus (IFG) activations implicate in the regulation and inhibition of this emotional response.

2.4.2 The Investigation of the Empathy Feeling on the Human's Relationships

An interesting research was investigated by **(Decety and Lamm 2006)** exploring the human empathy through the Lens of Social Neuroscience. Their work demonstrated that adopting a self-perspective when observing others in pain, resulted in strong feelings of personal distress activated the pain matrix to a larger extent, as well as the amygdala. Such a complete self-other merging seems to be detrimental to empathic concern. In addition, they found that the best response to another person’s plight might not be distress, but efforts to soothe that distress. Conversely, when participants took the other’s perspective, there was less overlap between the neural circuits involved in the processing of first-hand experience of pain, and they indeed reported more feelings of empathic concern.

In other study of **(Masten**, **Morelli & Eisenberger 2011)** was examined empathy-related neural processing and resulting prosocial behavior during observed social exclusion. Their results indicated that the neural regions supporting empathy for social pain may differ from those previously linked with empathy for physical pain. For example, responses to observed physical pain may trigger an automatic, affective response such that most individuals spontaneously feel distress when they saw someone in physical pain. In contrast, observing social exclusion might require an additional layer of mentalizing to understand the situation and imagine the victim's affective responses, and thus, might only elicit pain-related neural activity among the most empathic individuals. Given that understanding social situations is complex and relatively ambiguous, simply watching someone experiencing social pain may not automatically elicit distress like observing physical pain.

Unlike the research of Masten the study of **(Novembre, Zanon, and Silani 2014)** provides evidence that experiences of social rejection can activate regions of the brain so far observed during experiences of physical pain and possibly responsible for coding the intensity of the threatening event. The aim of this study was to explore what extent first

person experiences of physical and social pain overlap as well as the commonalities and differences related to the vicarious experience of physical and social pain. In addition, their results showed that this pattern of brain activation extends to the witnessing of the same type of social pain in others.

Although, in healthy individuals, affective perspective taking has proven to be an effective means to elicit empathy and concern for others, in individuals with psychopathy the extent to which perspective taking can elicit an emotional response was studied for the first time by **(Decety et al. 2013)**. Their results demonstrated that while individuals with psychopathy exhibited a strong response in pain-affective brain regions when taking an imagine-self perspective, they failed to recruit the neural circuits that were activated in controls during an imagine-other perspective, and that might contribute to lack of empathic concern.

Another interesting research was done by **(Meyer et al. 2015)** investigating the neural responses during an empathy paradigm for friends and strangers. Their results demonstrated that the mechanisms linking interdependence to empathy differentiate between close others and strangers. Specifically, they found that MPFC activation when observing a friend's exclusion (vs. inclusion) positively correlated with trait interdependence and empathy, whereas MPFC activation when observing a stranger's exclusion (vs. inclusion) negatively correlated with trait interdependence and empathy.

One of the goals of this study was to explore the brain activation associated with the empathy feeling when a player got social exclusion. We simulated situations of the empathy feeling implementing rounds of the game where programmed players got excluded from the other virtual players. The outcomes of this work showed that watching the exclusion of other demonstrated activity in emotional brain regions such as bilateral parahippocampal and amygdala. One can attribute this activation pattern to the imprecise mapping of avatar features to normative 'top-down' representational expectancies of the human body within extrastriate visual cortices (particularly areas like STS that are functionally tied to humans emotional signals) and to the processing of negatively-valenced stimuli, activating lateral orbitofrontal cortex.

An extension of this implementation is to simulate situations of the empathy feeling for friends and strangers and explore whether these situations activate different brain regions. This can be employed by modelling the avatars in order to look like a friend or a stranger, respectively.

2.4.3 The Reinforcement Learning Method

Reinforcement learning is referred as an important method by which people learn and change behavior based on feedback as well as they plan actions in order to maximize reward. Particularly, Reinforcement learning is an approach to learn policies for agents acting in an unknown stochastic world, observing the states that occur and the rewards that are given at each step. Reinforcement learning is learning to maximize a reward signal by exploring many possible actions. The agent is not told the correct actions. Instead it explores the possible actions and remembers the reward it receives. With supervised learning, an agent takes an action and is then told what the correct action was.

The Reinforcement learning algorithms are usually used in video games in order to create automated opponents that perform well in strategy games. For instance, human players rapidly discover and exploit the weaknesses of hard coded strategies. To build better strategies a reinforcement learning approach is suggested for learning a policy that switches between high-level strategies. These strategies are chosen based on different game situations and a fixed opponent strategy. Strategy games are an important and difficult subclass of video games. In games such as Warcraft and Civilization players build cities, train workers and military units, and interact with other human or AI players. The goal of these games is to make the best use of limited resources to defeat the opposing players. The large state space and action set, uncertainty about game conditions as well as multiple cooperative and competitive agents make strategy games realistic and challenging.

All Reinforcement learning algorithms are based on estimating the value function which is the expected return starting from state s_t and following policy π. $V^{\pi}(s) = E^{\pi}\{R_t \mid s_t = s\}$ The value function is an estimate of how good it is for an agent choosing actions based the policy π to be in a given state. Similarly, the state-action value function is the expected return starting from state s, taking action α, and thereafter following policy π, defined as $Q^{\pi}(s, \alpha) = E^{\pi}\{R_t \mid s_t = s, \alpha_t = \alpha\}$.

The **five basic elements of a Reinforcement learning** system are:

1. **Agent**: The learning component of a RL system. He executes some actions $a \in A$, where A the set of the allowed actions, depending on the state of the environment in which it belongs.
2. **Model of the environment**: Mimics the behavior of the environment. For instance, given a state and an action it determines what the next action will be. The environment is described by a set of states $s \in S$, where S the total states of the environment.

3. **Policy π**: Determines the behavior of each agent at each time.
4. **Reward function**: Maps to each state a reward, which expresses whether this state is desirable. The reward function determines the set of the states that are good for the agent to be found.
5. **Value function:** Determines what actions are good in the long term. Unlike the reward function expressing the temporary value of a state of the environment, the value function reflects the long-term value of a state taking into account the states that may arise and the corresponding rewards.

Markov Decision Process (MPD) is a common method for modeling sequential decision-making with stochastic actions. We learn a policy for an MDP through reinforcement learning approaches. We represent the learning problem as an MDP, defined as a tuple (S, A, P, R) with:

- S, a finite set of states with designated initial state S_0.
- A, a finite set of actions.
- P, a set of state transition probabilities: P(s' |s, α), the probability of transitioning from state s to s' when action α is taken by the agent.
- R, a reward function: R(s, a), a real-valued immediate reward for taking action α in state s.

An MDP unfolds over a series of steps. At each step, the agent observes the current state, s, chooses an action, α, and then receives an immediate reward that depends on the state and action, R(s, α). The agent begins in the initial state S_o, which is assumed to be known. The state transitions according to the distribution P as given above and the process continues. The goal is to find a policy, which is a mapping, π, from states to actions, that maximizes the sum of rewards over the steps of the problem. The value of a policy π at state s can be calculated as:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum P(s' | s, \pi(s))\, V^{\pi}(s').$$

Where $\pi: S \rightarrow A$ is a mapping from states to actions according to policy π, $\gamma \in [0, 1)$ is the discount factor.

Q–learning method is a Reinforcement Learning algorithm that updates the value of a state-action pair after the action has been taken in the state and an immediate reward has been received **(Watkins & Dayan 1992)**. Values of state-action pairs, Q(s, α) are learned because the resulting policy is more easily recoverable than learning the values of states

alone, V(s). Q-learning converges to an optimal value function under conditions of sufficiently visiting each state-action pair, but often requires many learning episodes to do so. When an action α is taken in state s, the value of a state-action pair, or Q-value, is updated as

$Q(s, \alpha) = Q(s, \alpha) + a(r + \gamma V(s') - Q(s, \alpha))$, Where $\alpha \in [0, 1]$ is the learning rate, r is reward that is observed, γ is the discount factor, s' is the next state, and $V(s') = \max_{a'} Q(s', \alpha')$.

The learning rate *α* determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information. In fully deterministic environments, a learning rate of $\alpha = 1$ is optimal. When the problem is stochastic, the algorithms still converges under some technical conditions on the learning rate that require it to decrease to zero. In practice, often a constant learning rate is used, such as $\alpha = 0.1$ for all t.

The discount factor *γ* determines the importance of future rewards. A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. If the discount factor meets or exceeds 1, the action values may diverge.

A study was conducted by **(D'Mell 2012)** examining the effects of social reward on reinforcement learning. Participants completed three versions of a reinforcement learning task where feedback varied. These included a visual condition where correct and incorrect feedback was presented visually, a neutral condition where correct and incorrect feedback was presented by a human voice and a reinforced condition where feedback was presented by human voice saying positive and negative social statements. Learning strategies, for example, exploitative versus explorative behaviors and the impacts of positive versus negative feedback on learning were measured. The type of feedback received was not found to have any significant effect on early learning strategies. Additionally, there was no difference in the impacts of positive versus negative feedback on reinforcement learning.

In this work, we implemented a more sophisticated version of Cyberball3D+ including state of the art Reinforcement Learning methods in Games and incorporating a form of learning affecting players' behavior in Cyberball3D+. This sophisticated version of Cyberball3D+ made the game more competitive and intelligent integrating rules and scores as well as creating intelligent opponents (computer players) modeling their decision making behavior.

We created two potential rules that were the ability of each player to throw the ball to any of the rest of the players of the game and the inclusion of a reward system whereby points were given to or removed from a player depending on his/her ball throws. In particular, if a player threw the ball to the player that had the highest number of receiving tosses he/she lost a point while if a player threw the ball to the player that had the lowest number of receiving tosses he/she won a point. In addition, if a player threw the ball to a player that had a medium number of receiving tosses he/she did not lose or win points. The player that obtained the highest score of points he/she won the game. The score of the participant was displayed on the screen during the rounds of the game.

Participants were informed that they were able to throw the ball to any player in the game and also that there was a reward system. Although they were not informed on how they gain or lose points, they were able to figure it out on their own, if they paid attention on the pointing system since the score appear on the screen. The **purpose of this approach** was to make the players to understand that they had to feel empathy for the excluded players throwing the ball to them if they wanted to win the game.

2.4.4 The Variants of the Cyberball Game for the Investigation of the Social Exclusion

Cyberball is a ball-toss game that can be used for research on ostracism, social exclusion or rejection. It has also been used to study discrimination and prejudice. Cyberball was originally intended as a simulation of ostracism in the context of a research program **(see Williams 1997, 2001; Williams, Forgas & von Hippel 2005; Williams & Zadro 2005).**

During the first Cyberball experiments, the participants were recruited to log on to an online experiment where they played a virtual ball-tossing game with two other participants who had logged on from somewhere else in the world **(Williams, Cheung and Choi 2000)**. Before the experiment, the participants were informed that Cyberball was a means to an end, and was, by itself, unimportant for the experiment. It was portrayed as merely a task that helped participants exercised their mental visualization skills, which they would purportedly use in the subsequent experimental task. After reading the instructions, they would view a game where the players were represented on the screen by animated icons. They would then play the Cyberball for about 5 min. The results showed that if participants were over-included (getting the ball for half the throws) or included (getting the ball for one third of the throws), they felt better than if they received the ball for only one sixth of the throws. Still, getting the ball for a sixth of the time was significantly better than

not getting it at all. Fully ostracized participants answered a post-experimental questionnaire indicating lower levels of belonging, self-esteem, control, and meaningful existence **(Williams and Jarvis 2006)**.

Brain imaging studies demonstrate that physical pain sensations are processed across a network of connected brain regions, which together are proposed to form the 'pain matrix'. Most of the pain matrix is also activated if we observe empathically someone else in physical pain **(Eisenberger 2012)**. Interestingly, experiencing the 'pain of social exclusion' also engages these same regions during brain imaging studies of the original Cyberball task, consistent with the notion of social pain. There is early work suggesting that a similar brain activation pattern is present when one sees and empathizes with someone else being ostracized **(Eisenberger 2012)**.

Researchers from around the world have used the Cyberball game for similar purposes, sometimes with minor modifications, but always for the purpose of manipulating inclusion and exclusion. **(Krill, Platek & Wathne 2008)** modified the cyberball game so it could be played over the internet. They programmed Cyberball to generate 60 tosses during each round and to delay each toss for 0.5–3 seconds in an effort to make the game seem realistic. Players were randomly assigned to either an inclusion round or an exclusion round. In the inclusion round the subject was involved in playing the game (throwing the ball) with confederates (computerized opponents) throughout the duration of the game. In the exclusion round, the participants were allowed to participate during the beginning of the game (six throws). After six throws the game was programmed to stop tossing the ball to the participant; the participant saw the computerized confederates playing amongst themselves but no longer received a throw during this exclusion period.

In the variant of the Cyberball game of **(Andari et al. 2010)** participants engaged in a multi round ball-toss game over a computer network with three fictitious partners. The researchers manipulated the amount of reciprocation exhibited by the three fictitious players. The critical task manipulation was the probability that each of the three fictitious players would throw the ball to the participants, which allowed us to create different cooperative behavior profiles (good, bad, and neutral). At game start, probabilities were homogeneous for all players, that was, the participant had a probability P = 1/3 of receiving the ball from any of the three players. After a predetermined number of rounds, player profiles diverged such that player A (the "good" profile) passed 70% of its played balls to player D (the participant), 20% to player C (the bad profile), and 10% to player B (the neutral profile); player C (the bad profile) passed 10% of its played balls to player D (the participant),

20% to player A (the good profile), and 70% to player B (the neutral profile); player B (the neutral profile) passed 30% of its played balls to player D (the participant), 40% to player C (the bad profile), and 30% to player B (the good profile). The game included a monetary incentive to enhance the participant's cognitive engagement in the task. Any player receiving the ball earned 2€. To optimize cognitive engagement in the task, the participant was told that each ball received was worth 2 euros and that when returning the ball two outcomes were possible: either the recipient would toss the ball back to the participant, generating further income, or toss it to another player, earning that player 2 euros. The participant's cumulative gains were displayed on the screen and he/she was led to expect a percentage of the gains at the end of the game. The participant was instructed that the game ended after a total of 80 tosses.

In the experiments of **(Lelieveld et al. 2012)** the participants were informed that they would participate in a three-player game of Cyberball. During the game, half of the participants were equally included by the other players (i.e., they received one third of the tosses) and half of the participants were excluded (i.e., they received one toss at the beginning and then never received another toss). In the financial compensation condition, participants obtained 50 euros' cent for each ball that was not thrown to them. There was a counter made visible in the Cyberball screen, which was incremented with 50 cents each time the participant was not given a ball (after the 30 throws, participants thus earned €14,50).

In this work the **Cyberball3D+** game was played by three virtual players and the human subject and many scenarios were evaluated by the neuroscientists, some fair and other unfair to the subject (simulating social exclusion) or to the other avatars (simulating empathy for social exclusion). The human subject played several rounds of the game. Each round included varied scenarios simulating situations of social exclusion or empathy as well as varied levels of anthropomorphism of the avatars. **For the first time,** the Cyberball game was implemented in three - dimensional form as well as it included three level of anthropomorphism (Figure 7). In addition, in this project three versions of the game were implemented including the five basic scenes, the probabilities version and the Reinforcement Learning version. Each of these versions simulated different situations of social exclusion or empathy. Especially, in the probabilities version of the game the neuroscientists were able to simulate any situation of the fairness of the game they wanted by filling in the percentages of the tosses that each player would throw to each one of the other players.

This game proposed was designed to render an interactive Virtual Environment (VE) on an fMRI display, enabling the conduct of formal neuroscientific experiments and investigating the effects of social exclusion, empathy and different level of anthropomorphism on human brain activity. **For the first time**, we got a validated neuroscientific measure of character believability and emotional engagement at the same time the experience was taking place.

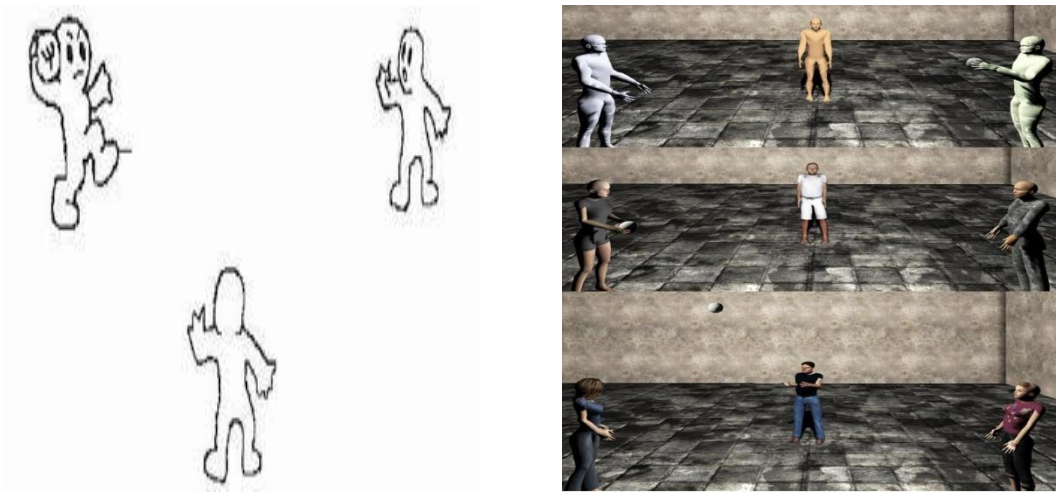


Figure 7: The original version of the cyberball game (left). The version of the cyberball game of this work in three - dimensional form (right).

2.5 Summary

In this chapter were presented some methods that were used by researchers in order to model and animate a 3d avatar as well as success real and fast rendering of the 3d avatar. Furthermore, in this chapter we provided background information regarding the virtual games were used inside fMRI scanner by neuroscientists in order to investigate social phenomena such as social exclusion and empathy. In addition, we described different variants of the Cyberball game that were used for the investigation of the social exclusion as well as for the investigation of the empathy for the social exclusion.

3 Chapter 3 – Software Architecture and Development Framework

In this chapter, we describe the software architecture and the development framework that is used in this project as well as the tools that are used to build several parts of this project's application, such as the Flash authoring environment.

3.1 Virtual Reality Technology and Game Engines

A game engine is a software framework designed for the creation and development of video games. Video game developers use them to create games for video game consoles, mobile devices and personal computers. The core functionality typically provided by a game engine includes a rendering engine for 2D or 3D graphics, a physics engine or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, and a scene graph. A game engine provides the framework and the Application User Interface (API) for the developer to use and communicate with the hardware. It consists of separate autonomous systems, each handling a specific process, e.g. the graphics system, the sound system, the physics system, etc.

There are several game engines offering a wide range of tools to create interactive environments, primarily used for the development of a computer game. Game engines are also powerful platforms for the development of any 3D interactive environment.

In the following subsections, are described three different game engines but two of them were rejected and one engine (UDK) was selected for the implementation of this 3D gaming system.

3.1.1 Unity 3D

Unity game engine offers a vast array of features and a fairly easy to grasp interface. Its bread and butter is cross-platform integration, meaning games can be quickly and easily ported onto Android, iOS, Windows Phone 8, and BlackBerry, making it a great game engine for the development of mobile games.

The game engine supports assets from major 3D applications like 3ds Max, Maya, Softimage, CINEMA 4D, Blender and more, meaning there is no real restrictions to the type of file formats that it supports. With the recent release of Unity 3D 4.3 it also has native 2D

capabilities, supporting sprites and 2D physics, making it a great game engine to use for the development of 2D games.

While the engine supports integration of just about any 3D application, it does, however, suffer in the amount of editing capabilities inside the engine editor. Unity 3D has no real modeling or building features outside of a few primitive shapes so everything will need to be created in a third party 3D application. It does, however, boast a large asset library where a wide variety of assets can be downloaded or purchased (pricing is determined by the asset author).

In terms of programming, Unity 3D supports three programming languages: JavaScript, C# and Boo, which is a python variation. All three languages are fast and can be interconnected. The game's logic runs in the open-source platform "Mono", offering speed and flexibility. Required for the development process a debugger is also included, allowing pausing the game at any time and resuming it step-by-step.

Unity 3D is widely used, utilized by a large community offering help. It is free for noncommercial use and is targeted to all platforms, such as PC, MAC, Android, iOS and web. This game engine targets at offering increased rendering speed, even on machines with low memory and computational power, such as iOS and Android smartphones, and not at creating interactive photorealistic environments, which need a lot of memory and very fast CPU and GPU to render at acceptable speed. Unity 3D was rejected, because it was necessary to have the ability to create photorealistic VEs and render them in real-time.

3.1.2 Torque 3D

Torque 3D is a sophisticated game engine for creating networked games. It includes advanced rendering technology, a Graphical User Interface (GUI) building tool and a World Editor, providing an entire suit of WYSIWYG (What-You-See-Is-What-You-Get) tools to create the game or simulation application.
The programming language used is "TorqueScript", which resembles C/C++. It is targeted for both Windows and MacOS platforms, as well as the web. The main disadvantage of Torque 3D is that it is not free, but it needs to be licensed for $100. For this reason, Torque 3D was, also, rejected.

3.1.3 Unreal Engine 3 – Unreal Development Kit (UDK)

Unreal Development Kit (UDK) is one of the leading game engines currently. It became free on November 2009 for non-commercial use and is used by the world's largest

development studios. The UDK community includes thousands of people from around the world, providing help and advice.

The UDK is a framework used mostly in creating computer games and visualization. UDK consists of different parts, making it act both like a game engine and a 3D authoring environment. It provides the necessary tools to create 3D objects and assign materials on these, import 3D objects such as characters and their animations from 3ds Studio Max and import and use sounds and sound effects. It, also, allows the designed application to seemingly attach to Flash User Interfaces (UI). UDK can also be used to render computer graphics scenes as well as create and respond to events while playing the game.

UDK offers the ability to use both C/C++ and UnrealScript, which provides the developers with a built-in object-oriented programming language that maps the needs of game programming and allows easy manipulation of the actors in a synthetic scene. The main components inside the UDK are: the Unreal Editor which is used to create or import objects and edit VEs handling all the actors and their properties located in the VEs; the Unreal Kismet, which allows for the creation of sequences of events and corresponding actions and the Unreal Matinee which is responsible for the animation of actors or real-time changes in the actors' properties.

3.2 Tools of Unreal Development Kit (UDK)

In this project we used the UDK which, as we already discussed is a framework that is used mostly in creating computer games and visualization. UDK consists of different parts, making it act both like a game engine and a 3D authoring environment. In the following subsections are analyzed the UDK framework and its tools.

3.2.1 Unreal Editor

The Unreal Editor is the tool inside UDK that is used to create and edit VEs. 3D objects, materials, textures, lights, sounds, videos and images can be imported to the Content Browser library and inserted in the VEs through the Unreal Editor. Also, the Unreal Editor can create and assign materials to the 3D objects, as well as animate them (Figure 8).

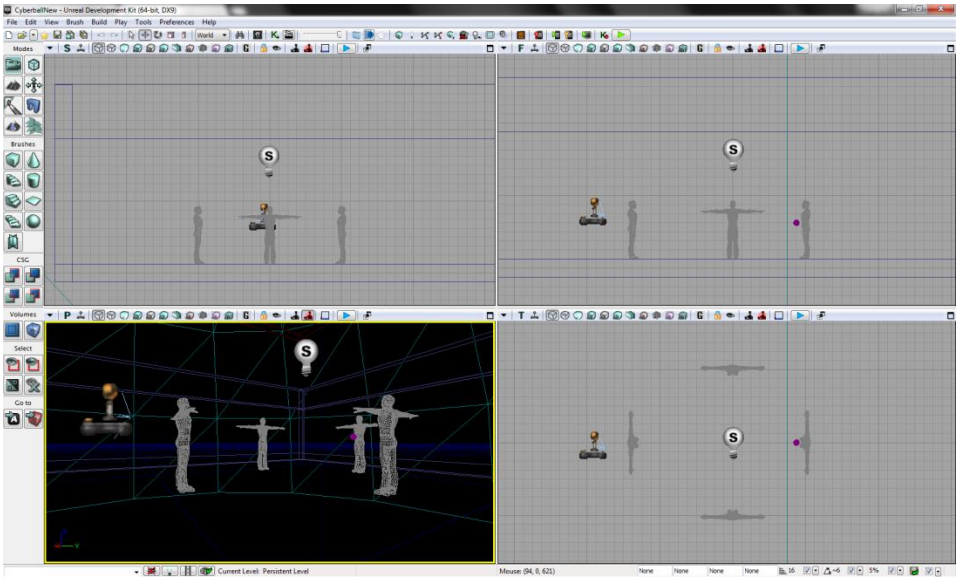


Figure 8: Screenshot from the Unreal Editor.

Everything inside the virtual scene made in the Unreal Editor is considered by UDK to be an Actor, extending from 3D objects to lights. This is in accordance with Unreal Script which is an Object-Oriented Programming language and each item is appointed to a class that extends from Actor. For example, the 3D objects are assigned to the *SkeletalMeshActor* class, the lights be variedly assigned to the *PointLight*, *PointLightToggleable*, *DominantDirectionalLight* classes according to their function, as well as the sounds are assigned to the *Sound* class. All these classes extend from the *Actor* class.

The 3D objects imported into Unreal Editor can be assigned to Static Mesh, used for static objects, or Skeletal Mesh, used for character bodies. The 3D objects in this project are Skeletal Meshes because represent virtual humans. After an object is imported through the Content Browser, we can change its main attributes, such as its clothes, materials and its movements within the Unreal AnimSet Editor (Figure 9). These changes will affect the instances of this object that will be inserted in the virtual scene, unless they are overridden.

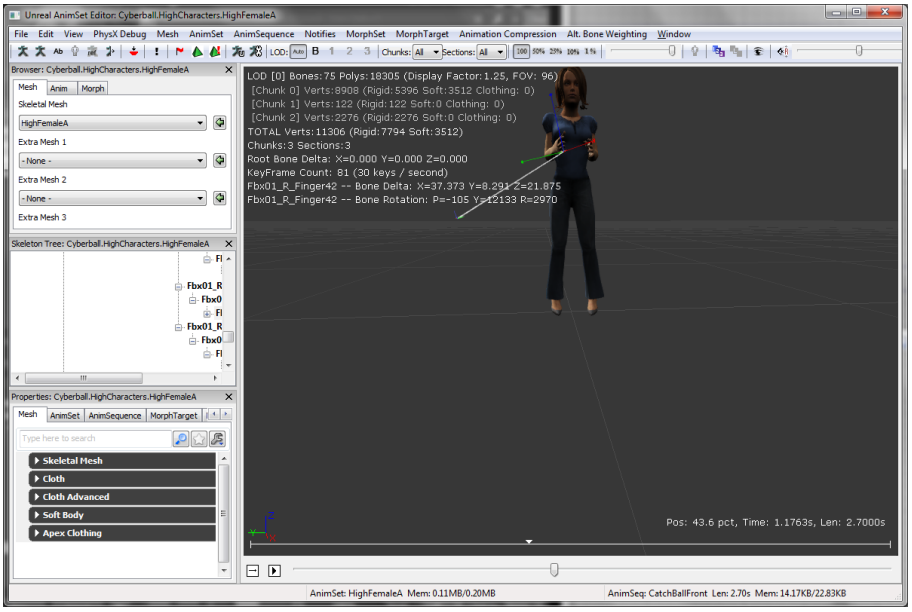


Figure 9: A paradigm of Unreal AnimSet Editor depicting a 3d human.

When an object is inserted in the Unreal Editor through the Content Browser library, an instance of its predefined Actor class is created and the editor offers the option to change the configuration of the specific instance, without affecting the other instances. The options that can be changed include the object's position, draw scale, properties for the lighting system, materials, actor's movements, actor's attachment, collision, components, etc. The Figure 10 displays a paradigm of SkeletalMeshActor, depicting the properties of a 3D human instance that was inserted in the VE. It shows that several properties can be changed, such as Physics and Collision properties, or the Location and Rotation of the object.

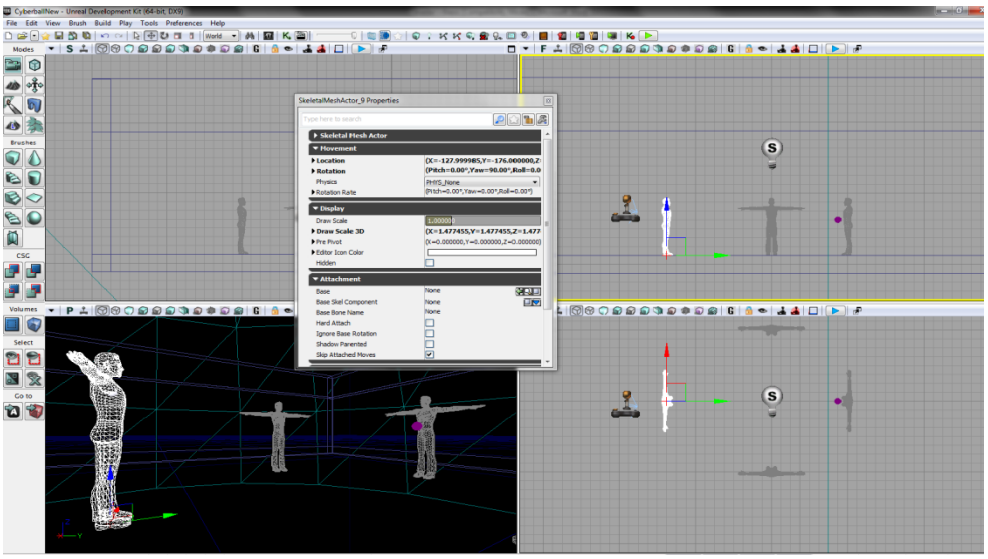


Figure 10: A paradigm of SkeletalMeshActor, depicting the properties of a 3D human instance.

There are many other actor classes that simulate the functions of a variety of other objects, such as triggers, sounds, dynamic meshes that can be moved or rotated, etc. These can be embedded and controlled through the Unreal Editor.

3.2.2 Material Editor

The Material Editor is a useful tool within Unreal Editor which is used to create realistic environment. Particularly, this tool is responsible for the creation and editing of different materials that can be assigned to objects inside the scenes created. The objects are affected by these materials in a variety of ways, mainly in terms of their texture and their interaction with the light. The Material Editor provides the ability to create shaders to be applied to geometry using a node-based graph interface.

Specifically, the Material Editor is node-based; however, its nodes do not represent events or actions, but textures, colors and several processing filters, such as addition of two different textures. It provides the main node which has all the supported properties of the material, such as the diffuse, emissive and specular properties and each property can receive the output from a node or from a sequence of nodes. The Material Editor can be opened by double-clicking any Material asset or through the right-click context menu of a Material asset in the Content Browser. Either of these will open up that particular Material in the Material Editor for editing.

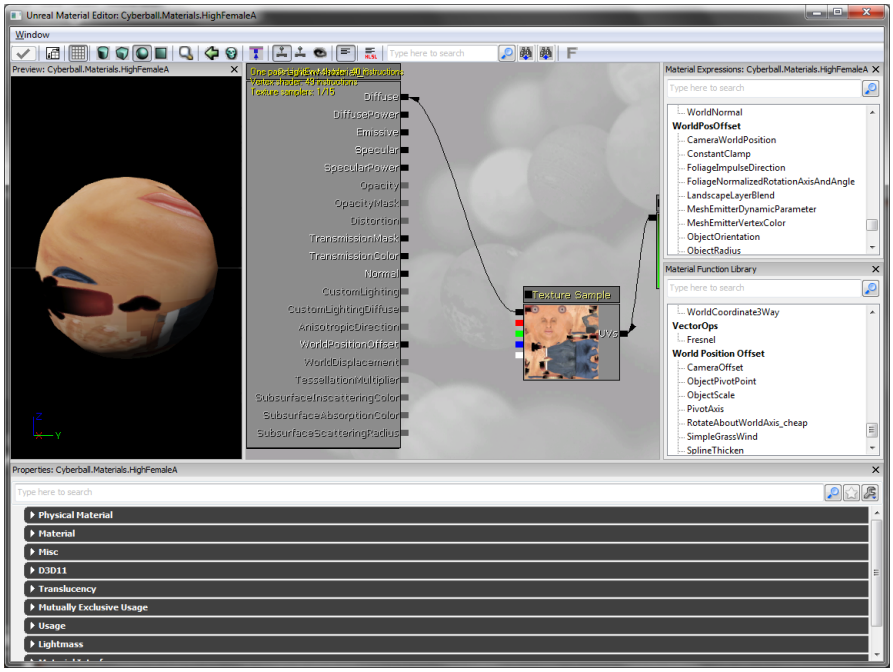


Figure 11: Screenshot from the Material Editor depicting the overall material that was used for a 3d human.

3.2.3 Unreal Kismet

Another useful tool inside the Unreal Editor is the Unreal Kismet which can be described as a graphical system that connects specific events to specific actions. It is responsible for the creation of sequences of events and corresponding actions. Particularly, it is node-based and properties of different nodes can be connected with arrows. However, there are only some predefined events and actions that can be created; more actions can be created through Unreal Script.

It is noteworthy to add that the complete sequence of events and actions applies only to the currently loaded scene and not to other scenes. Along these lines, Unreal Kismet is not efficient in creating general rules that apply to a complete game or application and for this reason the Unreal Script is recommended. As opposed to that, Unreal Kismet is preferred for the connection of specific events and actions.

Several Unreal Script events are generated in order to load the main menu of the game which was used by neuroscientists to fill in the parameters of the game and was loaded on the screen when the game was initiated as well as for the loading of the 'return' menu which was displayed on the screen at the end of the game. In addition, in the Figure

12, we can see the events that were created in Unreal Matinee, as detailed below, for the creation of the player's animations.

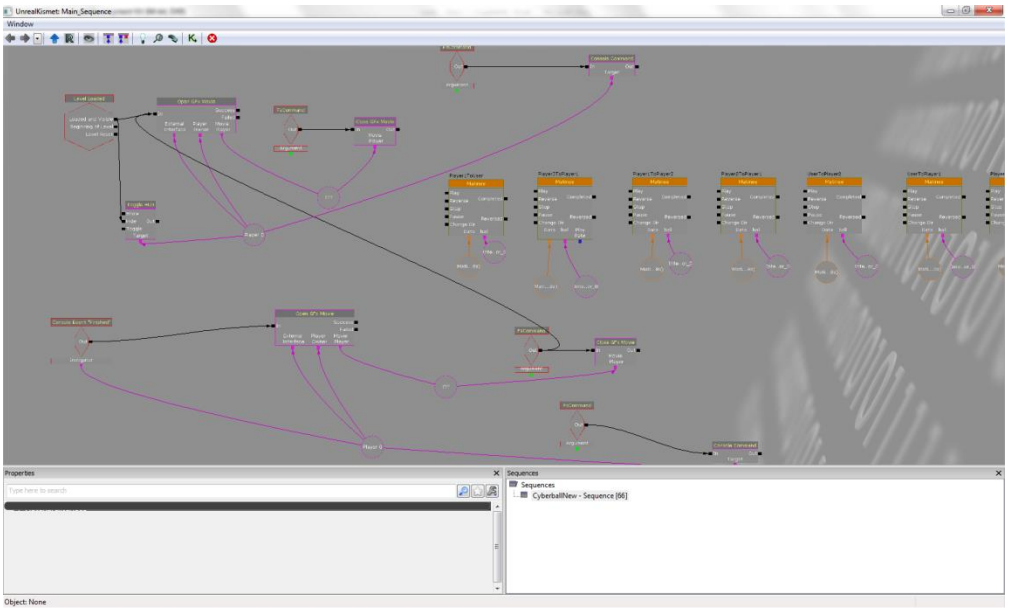


Figure 12: Unreal Script events are generated for the requirements of the game.

3.2.4 Unreal Matinee

The Unreal Matinee is responsible for the animation of the actors or for real-time changes in the actors' properties. It provides the ability to animate the properties of the actors over time, to create either dynamic gameplay or cinematic in-game sequences. The system is based on the use of specialized animation tracks in which key frames can be placed to set the values of certain properties of the actors in the level. The Matinee Editor is similar to the non-linear editors used for video editing, making it familiar to video professionals.

The advantage of working with Matinee is that the actors in the level can be moved around and their properties can be changed as part of previewing the sequence. However, when Matinee exits, all level state will be restored to the way it was when Matinee was entered.

We used the Unreal Matinne to create the animations of the ball as shown in Figure 13. Particularly, we created different Matinee events for each animation of the ball by designing separate Matinee sequences for each different direction of the ball from a player to all other players.

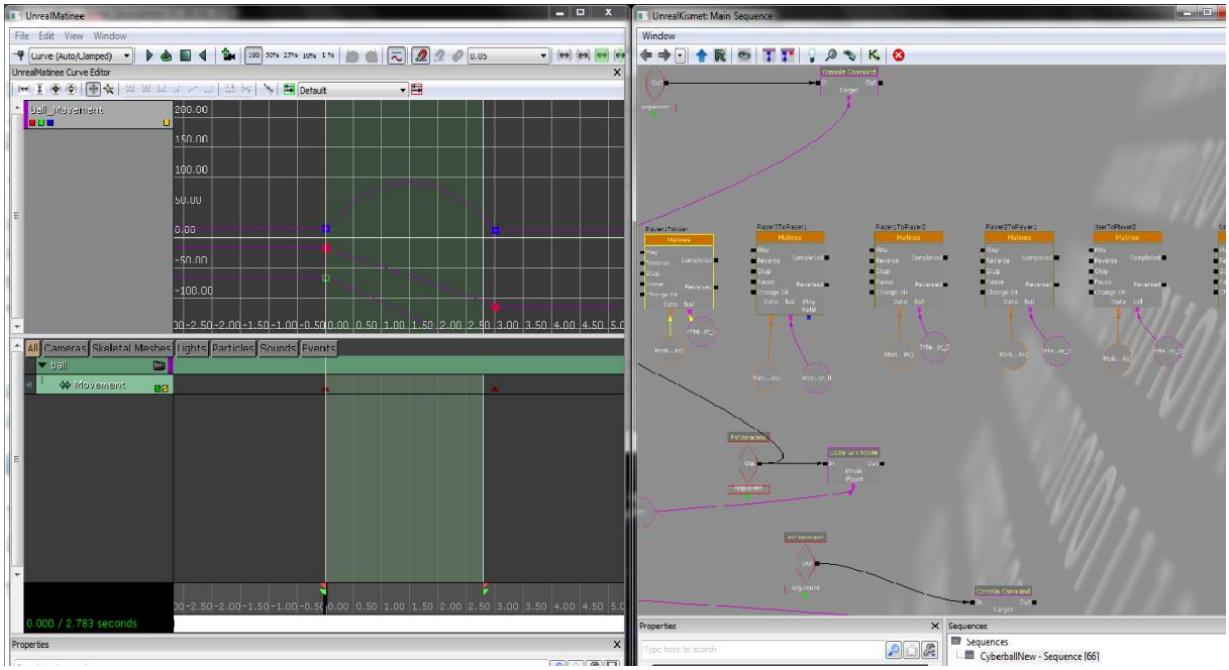


Figure 13: An example shows the creation of a ball animation.

3.2.5 Sound Engine

The behavior of the audio playback in the Unreal Engine is defined within Sound Cues. The Sound Cue Editor is a node-based editor that is used to work with audio and provides the necessary tools in order to create various sound effects. It supports immersive 3D location based sounds and gives complete control over pitch, levels, looping, filtering, modulation and randomization. The audio output of the combination of nodes created in the Sound Cue Editor is saved as a Sound Cue. By default, every Sound Cue's Audio Graph Node contains an output node, which has a speaker symbol on it. The output node's default value for Volume Multiplier is 0.75, and for Pitch Multiplier is 1.00. These values can be altered in the details panel. The volume and pitch settings are used to manage relative Sound Cue volumes. This affects the output of all audio contained within the Sound Cue.

As the rest of the Unreal Editor's tools, the Sound Editor provides a node-based User Interface to import and use several sound cues, change their properties as well as mix them together and channel the resulting output as a new sound effect. An example of the Sound Editor is shown in the Figure 14, which depicts a *SoundCue* which is used in the game. In the left part of the figure, the *SoundCue* residing in the asset library is shown; while in the right part of the figure the Sound Editor window is displayed by depicting one *Sound* node.

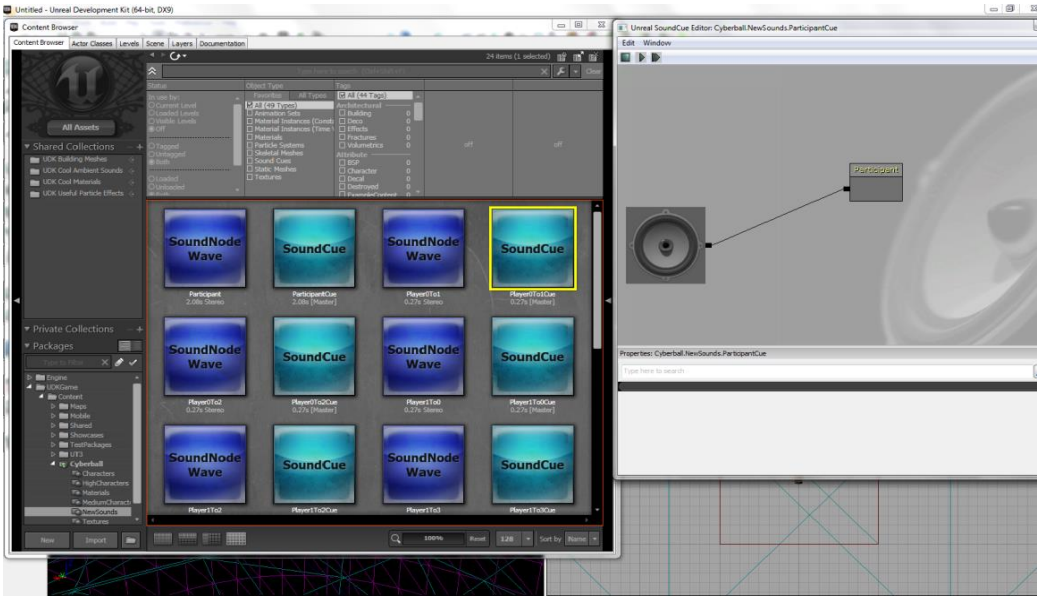


Figure 14: Depicts a *SoundCue.*

3.2.6 Configuration files

Almost every property of the Unreal Editor and its components can be changed through the configuration files, which are bound to their respective UnrealScript class that implements each component. There are two versions of the configuration files, the first is the default, which is used to initialize the component and the other is the compiled and altered version. If, for any reason, the compiled version of a configuration file is corrupted or deleted, a new one is created based on the default one.

Every UnrealScript class that binds to a configuration file can define which of its properties should be saved and / or edited in the respective configuration file. So, when the Unreal Editor loads up and initializes its components, it uses the configuration files to recall their properties. The UnrealScript class bound to a configuration file can save the current state of its properties to the file, by calling the *SaveConfig* function.

An UnrealScript can bind to a configuration file by declaring it in its definition in the UnrealScript file. For example, the following class definition declares that it will bind to the configuration file "*Cyberball*" and the class properties that are defined to be set through configuration will be saved and loaded through that file:

```
class CyberballGamePlayerController extends GamePlayerController DLLBind(TestDLL)
config(Cyberball);
```

Each configuration file stores the information as key – value pairs, with each pair being the declared variables as keys and their respective values as the value of the pair. So, assuming that the *Cyberball* included a variable definition and code as follows:

```
var config int ParticipantNumber;
ParticipantNumber++;
SaveConfig();
```

Then, UDK would produce the following statement in the *Cyberball*:

```
ParticipantNumber = 1;
```

(We suppose that the previous value of ParticipantNumber was zero*).*

3.2.7 Input Manager

The input manager is responsible to handle the communication between the input hardware, such as keyboard, mouse, joystick or button boxes and the application. The input manager examines a configuration file based on DefaultInput.ini and according to it binds each input action, such as the joystick/mouse movement or key press to a specific method designated to perform the selected action. The Unreal Editor comes with a default configuration file including a limited set of predefined bindings between buttons and methods; however, this file can be altered to match the needs of each application.

In order to create a new binding between a button press and the performed action or to change an already defined binding, this change must be reflected in the configuration file. Also, the method defined in the configuration file must exist in the UnrealScript code of the new application being developed.

For example, if we wanted to add or change the actions performed when some number buttons of the keyboard are pressed, we would add or change these lines in the "DefaultInput.ini" configuration file:

```
Left and Right Button Boxes (the buttons respond like pressing numbers 1,2,3,4 in the keyboard)

.Bindings=(Name="one",Command="GBA_ThrowLeft")
.Bindings=(Name="four",Command="GBA_ThrowRight")
```

```
.Bindings=(Name="GBA_ThrowLeft",Command="ThrowLeft");
.Bindings=(Name="GBA_ThrowRight",Command="ThrowRight");
```

In the first two lines, we define that the press of the 1 and 4 buttons of the keyboard corresponds to the game bindable actions named GBA_ThrowLeft and GBA_ThrowRight, respectively. In the next two lines, we define that if a game bindable action named GBA_ThrowLeft or GBA_ThrowRight occurs, then the Input Manager should start the *ThrowLeft or* ThrowRight method respectively, which are located in the application's UnrealScript file. Inside those methods we could effectively develop the application to perform whatever actions are necessary to correspond to the press of these buttons of the keyboard.

3.2.8 DLL Files

UDK provides the option for an UnrealScript class to bind to an external DLL file, by declaring it in the class definition. For example, the following line declares that *CyberballGamePlayerController* class binds to the *TestDLL*:

```
class CyberballGamePlayerController extends GamePlayerController DLLBind(TestDLL)
config(Cyberball);
```

By binding to a DLL file, an UnrealScript class can call the declared in that DLL file methods or functions, which are written in C/C++. This proves to be an easy and efficient way to implement functions that either UDK does not support at all, such as I/O operations, or it would slow down the application, due to the fact that UnrealScript is slow.

A function residing inside the DLL must be first declared in the UnrealScript class file, and then it can be called exactly like it would be if it was an original UnrealScript function. Following the previous example and assuming that the *TestDLL* contained a function called *SaveFiledll*, the code inside the UnrealScript class would be:

```
dllimport final function SaveFiledll(string saveThefile, string text); //Function declaration

SaveFiledll(saveThefile,txt); //Function call
```

3.2.9 Lighting and Rendering Engine

The Unreal Development Kit (UDK) comes along with Gemini, a flexible and highly optimized multi-threaded rendering system, which creates lush computer graphics scenes and provides the power necessary for photorealistic simulations. UDK features a 64-bit color High Dynamic Range (HDR) rendering pipeline. The gamma-corrected, linear color space renderer provides for immaculate color precision while supporting a wide range of post-processing effects such as motion blur, depth of field, bloom, ambient occlusion and user-defined materials.

UDK supports all modern per-pixel lighting and rendering techniques, including normal mapped, parameterized Phong lighting, custom user-controlled per material lighting models including anisotropic effects, virtual displacement mapping, light attenuation functions, precomputed shadow masks and directional light maps. UDK provides volumetric environmental effects that integrate seamlessly into any environment. Camera, volume and opaque object interactions are all handled per-pixel. Worlds created with UDK can easily feature multilayered, global fog height and fog volumes of multiple densities.

It also supports a high-performance texture streaming system. Additionally, UDK's scalability settings ensure that the application will run on a wide range of PC configurations, supporting both Direct3D 9 and Direct3D 11.

The Unreal Development Kit (UDK) includes the Unreal Lightmass, which is an advanced global illumination solver. Unreal Lightmass supports the illumination with a single sun, giving off soft shadows and automatically computing the diffuse interreflection (color bleeding). It also offers a variety of options to optimize the illumination solution. It can provide detailed shadows by using directional light mapping, static shadowing and diffuse normal-mapped lighting. An unlimited number of lights can be pre-computed and stored in a single set of texture maps.

3.2.10 Unreal Lightmass

Unreal Lightmass is an advanced global illumination solver. It uses a refined version of the radiosity algorithm, storing the information in each illuminated 3D object's light map, while providing ray-tracing capabilities, by supporting Billboard reflections, which allows complex reflections even with static and dynamic shadows with minimal CPU overhead.

Unreal Lightmass is provided as part of the Unreal Development Kit (UDK) and it can only work on scenes created through it. Its performance is dependent on the complexity of the scenes created and the types of light emitting sources that exist in the scene. It is optimized to increase the renderer's performance.

Its main features include:

- *Area lights and shadows*: Within Lightmass, all lights are area lights by default. The shape used by Point and Spot light sources is a sphere. The radius of the sphere is defined by *LightSourceRadius* under *LightmassSettings*. Directional light sources use a disk, positioned at the edge of the scene. *Light source size* is one of the two factors controlling shadow softness, as larger light sources will create softer shadows. The other factor is distance from the receiving location to the shadow caster. Area shadows get softer as this distance increases, just like in real life.
- *Diffuse interreflection*: Diffuse Interreflection is by far the most visually important global illumination lighting effect. Light bounces by default with Lightmass, and the *Diffuse* term of each material controls how much light (and what color) bounces in all directions. This effect is sometimes called color bleeding. It's important to remember that diffuse interreflection is incoming light reflecting equally in all directions, which means that it is not affected by the viewing direction or position.
- *Mesh Area Lights from Emissive*: The *emissive* input of any material applied to a static object can be used to create mesh area lights. Mesh area lights are similar to point lights, but they can have arbitrary shape and intensity across the surface of the light. Each positive emissive texel emits light in the hemisphere around the texel's normal based on the intensity of that texel. Each neighboring group of emissive texels will be treated as one mesh area light that emits one color.

- *Translucent shadows*: Light passing through a translucent material that is applied to a static shadow casting mesh will lose some energy, resulting in a translucent shadow.

In this thesis, we present a complete system based on the Unreal Development Kit (UDK) where participant is either excluded or not from a ball tossing game played by three virtual players while immersed in an fMRI scanner detailed in Chapter 4 and 5. A complete neuroscientific experimental scenario has been implemented detailed in Chapter 6.

3.3 UnrealScript

UnrealScript was designed to provide the developers with a powerful, built-in programming language that maps the needs of game programming. The major design goals of UnrealScript are:

• Enabling time, state and network programming, which traditional programming languages do not address but are needed in game programming. C/C++ deals with AI and game logic programming, through events which are dependent on aspects of the object's state. This results in long-length code that is hard to maintain and debug. UnrealScript includes native support for time state and network programming which not only simplifies game programming, but also results in low execution time, due to the native code written in C/C++.

• Programming simplicity, object-orientation and compile-time error checking, helpful attributes met in Java are also met in UnrealScript. More specifically, deriving from Java UnrealScript offers:

- A pointerless environment with automatic garbage collection;
- A simple single-inheritance class graph;
- Strong compile-time type checking;
- A safe client-side execution "sandbox";
- The familiar look and feel of C/C++/Java code.

Often during the implementation, design trade-offs had to be made, choosing between execution speed and development simplicity. Execution speed was then sacrificed, since all the native code in UnrealScript is written in C/C++ and resulted performance outweighs the added complexity. UnrealScript has very slow execution speed compared to C, but since a large portion of the engine's native code is in C only the 10%-20% of code in UnrealScript that is executed when called has low performance.

3.3.1 The Unreal Virtual Machine

The Unreal Virtual Machine consists of several components: The server, the client, the rendering engine, and the engine's support code.

The Unreal server controls all the gameplay and interaction between players and actors (3D objects, lights or sounds that can be inserted in a synthetic scene). A listen server is able to host both a game and a client on the same computer, whereas the dedicated server allows a host to run on the computer without a client. All players connect to this machine and are considered clients.

The gameplay takes place inside a level, containing geometry actors and players. Many levels can be running simultaneously, each being independent and shielded from the other. This helps in cases where pre-rendered levels need to be fast-loaded one after another. Every actor on a map can be either player-controlled or script-controlled. The script controls the actor's movement, behavior and interaction with other actors. Actor's control can change in game from player to script and vice versa.

Time management is done by dividing each time second of gameplay into *Ticks*. Each *Tick* is only limited by CPU power, and typically lasts 1/100th of a second. Functions that manage time are really helpful for gameplay design. Latent functions, such as *Sleep*, *MoveTo* and more cannot be called from within a function but only within a state.

When latent functions are executing in an actor, the actor's state execution does not continue until the latent functions are completed. However, other actors may call functions from the specific actor that handles the latent function. The result is that all functions can be called, even with latent functions pending.

In UnrealScript, every actor is as if executed on its own thread. Windows threads are not efficient in handling thousands at once, so UnrealScript simulates threads instead. This means that 100 spawned actors will be executed independently of each other each *Tick*.

3.3.2 Object Hierarchy

UnrealScript is purely object-oriented and comprises of a well-defined object model with support for high level object-oriented concepts such as serialization and polymorphism. This design differs from the monolithical one that classic games adopted having their major functionality hardcoded and being non-expandable at the object level. Before working with UnrealScript, understanding the object's hierarchy within Unreal is crucial in relation to the programming part (Figure 15).

The main gain from this design type is that object types can be added to Unreal at runtime. This form of extensibility is extremely powerful, as it encourages the Unreal community to create Unreal enhancements that all interoperate. The five main classes one should start with are *Object*, *Actor*, *Pawn*, *Controller* and *Info*.

Object is the parent class of all objects in Unreal. All of the functions in the *Object* class are accessible everywhere, because everything derives from *Object*. *Object* is an abstract base class, in that it doesn't do anything useful. All functionality is provided by subclasses, such as *Texture* (a texture map), *TextBuffer* (a chunk of text), and *Class* (which describes the class of other objects).

Actor (extends *Object*) is the parent class of all standalone game objects in Unreal. The *Actor* class contains all of the functionality needed for an actor to be placed inside a scene, move around, interact with other actors, affect the environment, and complete other useful game-related actions.

Pawn (extends *Actor*) is the parent class of all creatures and players in Unreal which are capable of high-level AI and player controls.

Controller (extends *Actor*) is the class that defines the logic of the pawn. If *Pawn* resembles the body, *Controller* is the brain commanding the body. Timers and executable functions can be called from this type of class.

Info (extends *Actor*) is the class that sets the rules of gameplay. Players joining will be handled in this class, which decides which *Pawn* will be created for the player in the scene and which *Controller* will handle the behavior of the pawn.

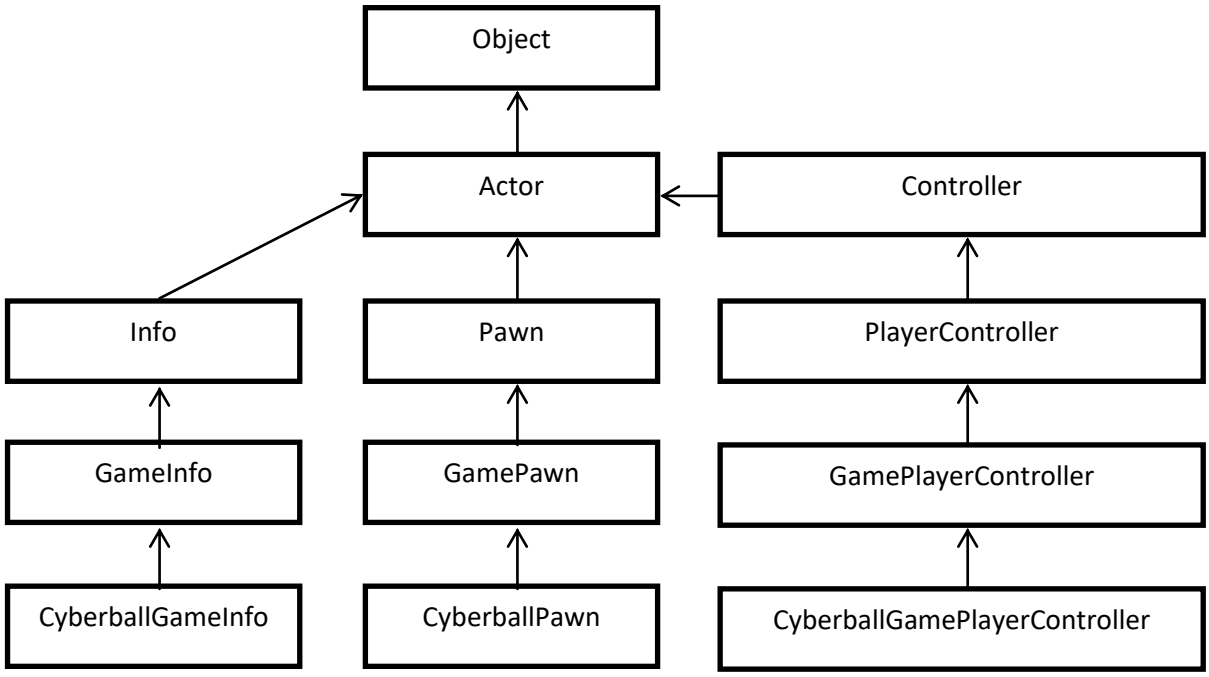


Figure 15: The class hierarchy for the three most important classes that control the application flow.

As already described, the *CyberballGameInfo* class decides how new players entering the scene are treated. The *CyberballPawn* class describes the properties and the behavior of a *Pawn* entering a scene that is handled by *CyberballGameInfo*. Finally, the *CyberballGamePlayerController* class handles the *Pawn* in the scene, according to user input.

3.3.3 Timers

Timers are a mechanism used for scheduling an event to occur. Time management is important both for gameplay issues and for programming tricks. All Actors can have more than one timers implemented as an array of structs. The native code involving timers is written in C++, so using many timers per tick is safe, unless hundreds expire simultaneously. This would require the execution of UnrealScript code for each one of the timers and the heavy computational demands would lead to unwanted delay to the handling of the timers.

```
SetTimer(0.7, false, 'ThrowBall');
```

This line of code defines that after 0.7 seconds, the function *ThrowBall* should be called. The false value passed as an argument means that this timer should not repeat the counting; it will only work once.

3.3.4 States

States are known from hardware engineering where it is common to see finite state machines managing the behaviour of a complex object. The same management is needed in game programming, allowing each actor to behave differently, according to its state. Usually, when implementing states in C/C++ there are many switch cases used, based on the object's state. This method, however, is not efficient, since most applications require many states, resulting to difficulties in developing and maintaining the application.

UnrealScript supports states at the language level. Each actor can include many different states; however, only one can be active at any time. The state the actor is in reflects the actions it wants to perform. *Attacking*, *Wandering*, *Dying* are potential states a *Pawn* may acquire. Each state can have several functions, which can be the same as another state's functions. However, only the functions in the active state can be called. For example, if an application dictates that an action should only be performed in a specific stage, then this stage could be encapsulated in a different state that implements the function corresponding to that action differently than other states.

States provide a simple way to write state-specific functions, so that the same function can be handled in different ways depending on which state the actor is in when the function is called. Within a state, one can write special "state code", using the regular UnrealScript commands plus several special functions known as "latent functions". A latent function is a function that executes slowly (i.e. non-blocking), and may return after a certain amount of "game time" has passed. Time-based programming is enabled which is a major benefit that neither C/C++, nor Java offer. Namely, code can be written in the same way it is conceptualized. For example, a script can support the action of "turn the TV on; show video for 2 seconds; turn the TV off". This can be done with simple, linear code, and the Unreal engine takes care of the details of managing the time-based execution of the code.

3.3.5 Interfaces

UnrealEngine3's UnrealScript has support for interface classes that resembles much of the Java implementation. As with other programming languages, interfaces can only contain function declarations and no function bodies. The implementation for these declared methods must be conducted in the class that actually implements the interface. All function types, as well as events, are allowed. Even delegates can be defined in interfaces.

An interface can only contain declarations which do not affect the memory layout of the class: enums, structs and constants can be declared. Variables cannot be declared for this reason.

3.3.6 Delegates

Delegates are a reference to a function within an instance. Delegates are a combination of two programming concepts, e.g. functions and variables. In a way, delegates are like variables in that they hold a value and can be changed during runtime. In the case of delegates, though, that value is another function declared within a class. Delegates also behave like functions, because they can be executed. It is this combination of variables and functions that makes delegates such a powerful tool under the right circumstances.

3.3.7 Unreal Script Compiler

The UnrealScript compiler is three-pass. Unlike C++, UnrealScript is compiled in three distinct passes. In the first pass, variable, struct, enum, const, state and function declarations are parsed and remembered, e.g. the skeleton of each class is built. In the second pass, the script code is compiled to byte codes. This enables complex script hierarchies with circular dependencies to be completely compiled and linked in two passes, without a separate link phase. The third phase parses and imports default properties for the class using the values specified in the default properties block in the .uc file.

3.3.8 Unrealscript Programming Strategy

UnrealScript is a slow programming language when compared to C/C++. A program in UnrealScript runs about 20x slower than C. However, script programs written are executed only 5-10% of the time with the rest of the 95% being handled in the native code written in C/C++. This means that only the 'interesting' events will be handled in UnrealScript. For example, when writing a projectile script, you typically write a *HitWall*,

Bounce, and *Touch* function describing what to do when key events happen. Thus, 95% of the time, a projectile script isn't executing any code and is just waiting for the physics code to notify it of an event. This is inherently very efficient.

The Unreal log may provide useful information while testing scripts. The UnrealScript runtime often generates warnings in the log that notify the programmer of non-fatal problems that may have occurred.

UnrealScript's object-oriented capabilities should be exploited as much as possible. Creating new functionality by overriding existing functions and states leads to clean code that is easy to modify and easy to integrate with other peoples' work. Traditional C techniques should be avoided, such as writing a switch statement based on the class of an actor or the state because code like this tends to clutter as new classes and states are added or modified.

3.4 Flash Applications as User Interfaces

The Unreal Development Kit (UDK) supports Heads-Up Displays (HUD) that can be constructed in UnrealScript. In order to create a Graphical User Interface to be displayed on the top of the regular viewport such as a menu that displays textboxes, checkboxes etc. requires user response to it, however, this is inefficient. For this reason, UDK provides support to integrate a Flash application inside a scene and project it on top of the surface of a 3D object in the scene, or in the center of the screen.

A Flash application can be ideally used as a User Interface in UDK, because it incorporates the ability to display animated graphics, text or buttons on the screen on top of the scene being rendered. It can also receive user input and provide feedback according to it. A Flash application consists of many frames placed in the main timeline, an example shown in Figure 16. The flow of the frames being displayed can be changed through ActionScript - Flash’s scripting language, thus allowing controlling which frame will be displayed next and when that will happen. Each frame can have its own set of graphics, texts, movie clips and buttons and a script controlling the behavior of the frame’s components.

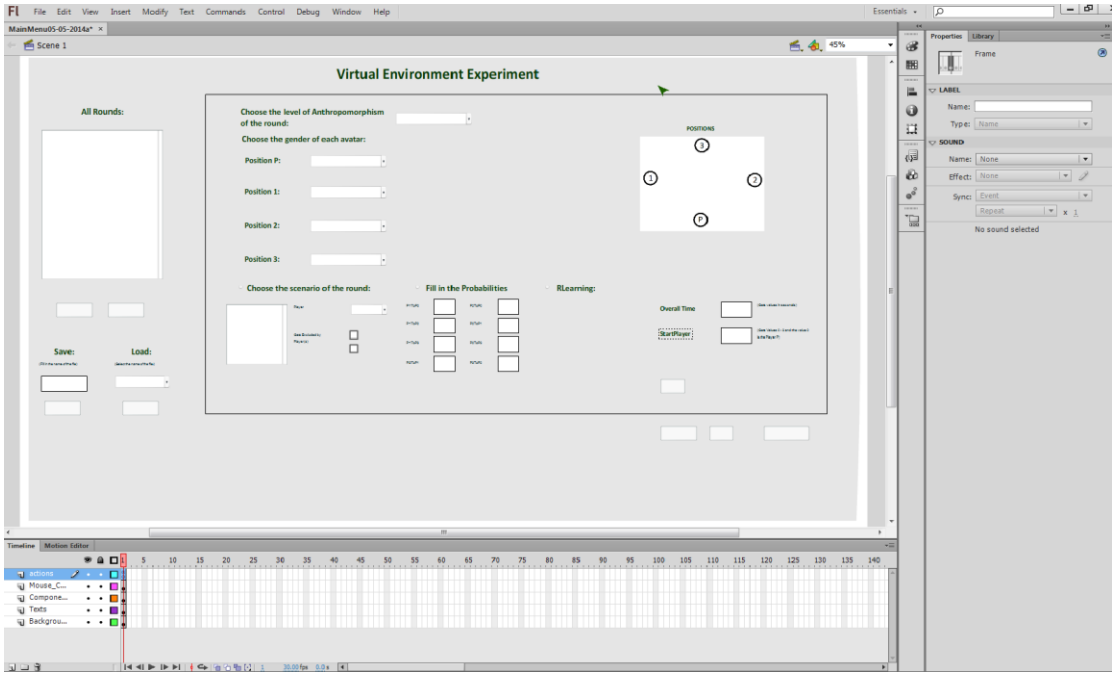


Figure 16: The Flash authoring environment is presented with a completed Flash User Interface.

The graphics, the buttons, the checkboxes, the comboboxes and the input texts inserted into the stage are shown. The components shown can be adjusted by user input through ActionScript, which is a scripting programming language with a superset of the syntax and semantics of the more widely known JavaScript and is suited to the development of Flash applications. More details about Action Script can be found in section 3.4.2.

3.4.1 Authoring Environment

In order to create a Flash application, a Flash authoring environment is necessary. There are many different Flash authoring environments available; however, the most powerful is Adobe Flash Professional CS5.5. Although it is not free, a 30-day trial version is available for download.

A freshly created Flash application is equipped with an empty stage and an empty timeline. Objects, such as movie clips, graphics, buttons, text, sounds or other Flash components, can be inserted into the application's library, or directly into the scene in the currently selected frame. Various different frames can be created, carrying different components inserted into each frame and the control of the application can be handled through ActionScript.

When the Flash application is fully developed and working, the authoring environment can compile the assets and the ActionScript comprising the application into an executable file in SWF format. Such files can be directly executed by a Flash player and also this is the format that UDK supports.

3.4.2 ActionScript 2.0

Although Flash Professional provides the tools to create applications running in all versions of ActionScript (up to 3.0) and of Flash Player (up to 10.3), UDK currently only supports the integration of Flash applications with ActionScript 2.0 (AS2) and Flash Player 8.

ActionScript is a scripting programming language and it is a dialect of ECMAScript, meaning it has a superset of the syntax and semantics of the more widely known JavaScript. It is suited to the development of Flash applications.

The language itself is open-source in that its specification is offered free of charge and both an open source compiler and open source virtual machine are available. It is often possible to save time by scripting something rather than animating it, which usually also enables a higher level of flexibility when editing.

ActionScript 2.0 primitive data types

The primitive data types supported by ActionScript 2.0 are:

- *String*: A list of characters such as "Hello World".
- *Number*: Any Numeric value.
- *Boolean*: A simple binary storage that can only be "true" or "false".
- *Object*: Object is the data type all complex data types inherit from. It allows for the grouping of methods, functions, parameters, and other objects.

ActionScript 2.0 complex data types

There are additional "complex" data types. These are more processor and memory intensive and consist of many "simple" data types. For AS2, some of these data types are:

- *MovieClip* - An ActionScript creation that allows easy usage of visible objects.
- *TextField* - A simple dynamic or input text field. Inherits the MovieClip type.

- *Button* - A simple button with 4 frames (states): Up, Over, Down and Hit. Inherits the MovieClip type.
- *Date* - Allows access to information about a specific point in time.
- *Array* - Allows linear storage of data.
- *XML* - An XML object
- *XMLNode* - An XML node
- *LoadVars* - Load Variables objects allows for the storing and send of HTTP POST and HTTP GET variables.
- *Sound*
- *NetStream*
- *NetConnection*
- *MovieClipLoader*
- *EventListener*

3.4.3 Connection between User Interface and Application

The integration of a Flash application inside a scene in UDK requires that it should first be compiled into an SWF file and imported inside the UDK asset library. Afterwards, either UnrealScript or Unreal Kismet can initiate the Flash application, interact with it, hide it or instruct it to stop playing.

While a Flash application is playing inside a scene, UnrealScript can initiate a call of an ActionScript function and vice versa. This feature allows full interaction between the Flash interface and the application. Consequently, it is easy and efficient to create an application that initiates a Flash interface whenever it is required and then receive the user's response and order it to stop playing.

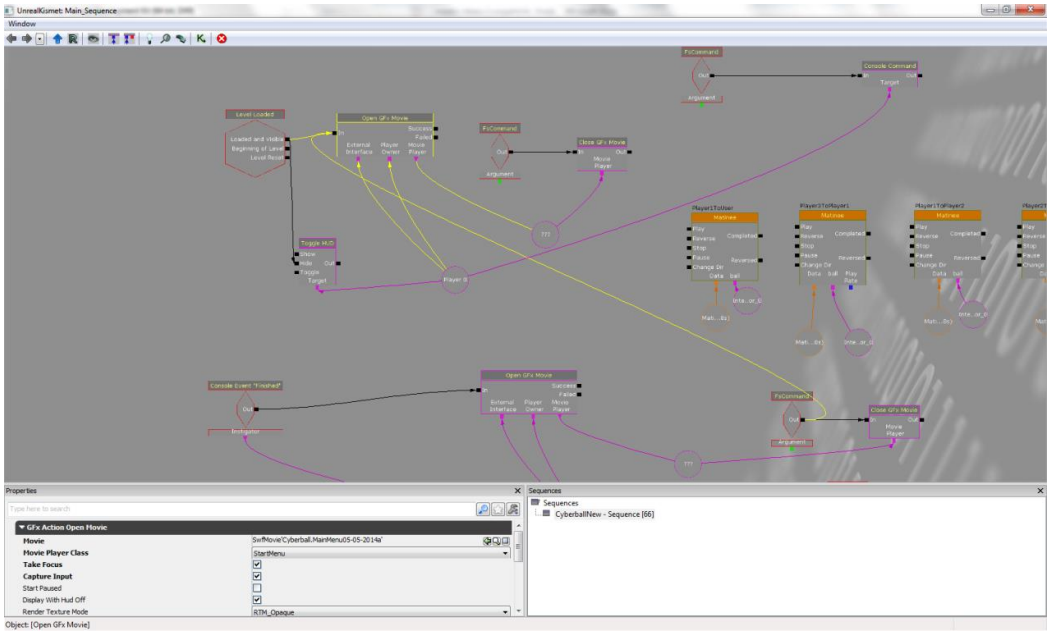


Figure 17: The actions *Open GFx Movie and* Close GFx Movie.

The action *Open GFx Movie* performs the required operations to start the Flash Movie, which is inserted as an argument in its properties, as shown in Figure 17. Additional settings may include whether a movie can capture user input as well as where to project this movie, either on the screen or on an Actor's surface.

Also, Unreal Kismet provides the action *Close GFx Movie* which handles the termination of the selected Flash application. In the Figure 17 the action *Close GFx Movie* firing up when a flash command is executed through Unrealscript.

3.5 Summary

In this chapter, the technical requirements of this interactive 3D gaming system were introduced. In addition, the architecture of the application that was developed for the experiments of this work was presented, along with the inherent architecture of the Unreal Development Kit (UDK) used to develop it.

4 Chapter 4 – Implementation

The Cyberball game is a ball-toss game that can be used for research on ostracism, social exclusion or rejection. It has also been used to study discrimination and prejudice. The Cyberball was originally intended as a simulation of ostracism in the context of a research program **(see Williams 1997, 2001; Williams, Forgas & von Hippel 2005; Williams & Zadro 2005).** During the first Cyberball experiments, participants were recruited to log on to an online experiment where they played a virtual ball-tossing game with two other participants who had logged on from somewhere else in the world.

In this project the **Cyberball3D+** game was played by three virtual players and the human subject and many scenarios were evaluated by the neuroscientists, some fair and other unfair to the subject (simulating social exclusion) or to the other avatars (simulating empathy for social exclusion). The human subject played several rounds of the game. Each round included varied scenarios simulating situations of social exclusion or empathy as well as varied levels of anthropomorphism of the avatars. A typical paradigm of a game playing consists of six rounds and each of these rounds is played twice but not consecutively and used a scene from the **five basic scenes**, lasting 1.3 minutes. The first round simulates the inclusion of all players and uses a low level of anthropomorphism; the second round simulates the inclusion of all players as the first round but uses a high level of anthropomorphism for all players. The third and the fourth rounds of the game simulate the exclusion of the participant in the fMRI scanner and use low and high level of anthropomorphism respectively. In addition, the fifth and the sixth rounds of the game simulate the exclusion of the programmed players and use low and high level of anthropomorphism respectively.

In this chapter the designing and implementation of the whole project is described analytically.

4.1 Scenarios

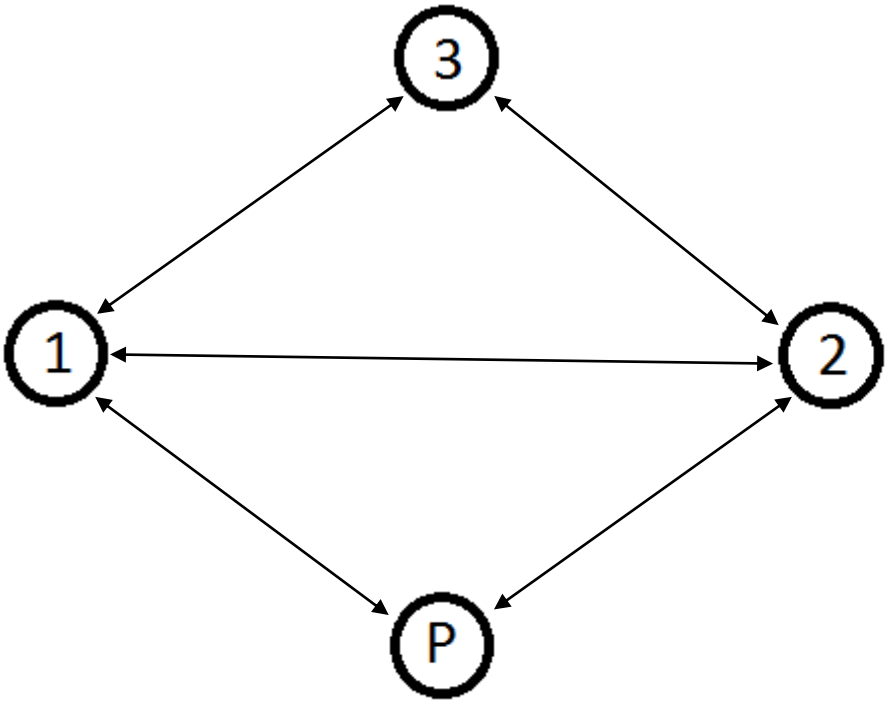
Many scenarios were evaluated by the neuroscientists, some fair and other unfair to the participant of the experiment (simulating social exclusion) or to the other avatars (simulating empathy for social exclusion). The human subject played several rounds of the game. Each round included varied scenarios simulating situations of social exclusion or empathy.

Particularly, for the purposes of the Cyberball3D++ game we implemented three versions of the game. The three versions were the five basic scenarios, the probabilities and the Reinforcement Learning version.

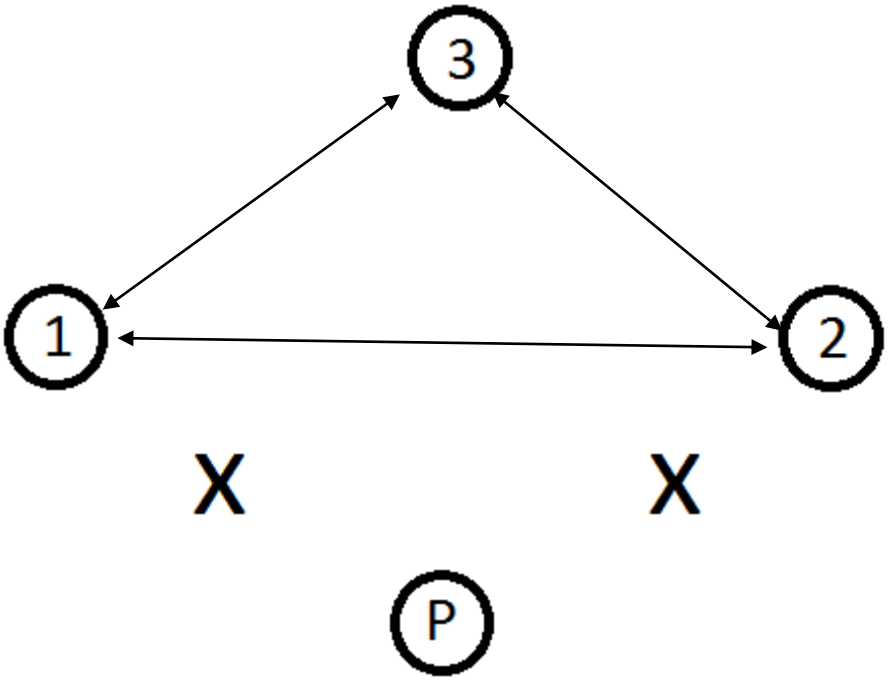
4.1.1 Five Basic Scenes

In the **five basic scenes** either no-one player is excluded or the participant/Player3 gets excluded by Player 1 or Player 2 or both of them. In addition, in this version of the game the participant and the Player 3 are not able to throw the ball to each other.

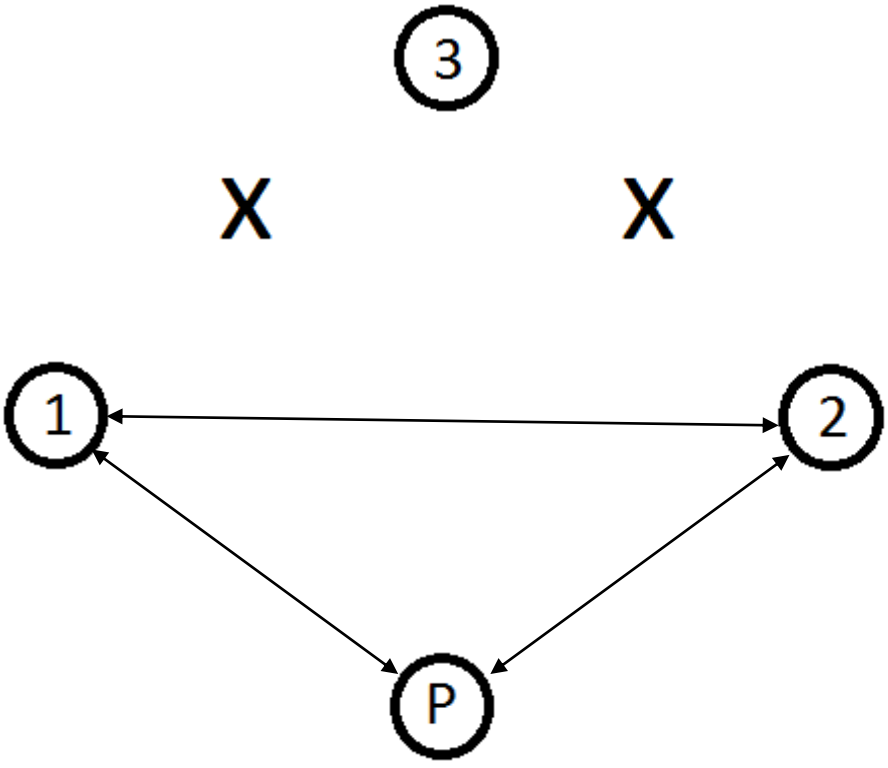
Scene1: *No – one player is excluded*



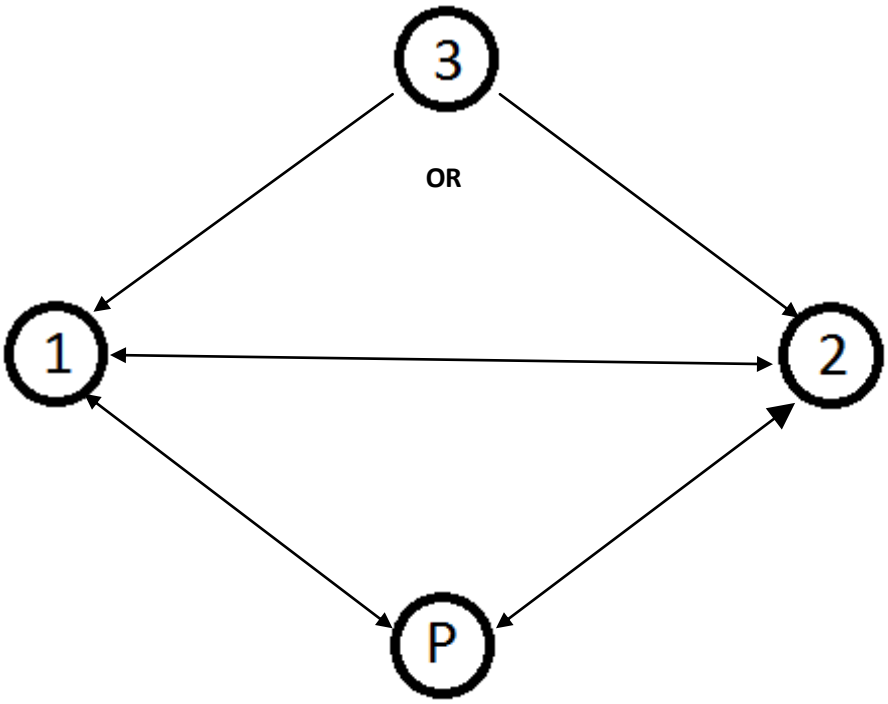
Scene2: *The user (Player P) gets excluded by Player 1 and Player 2*



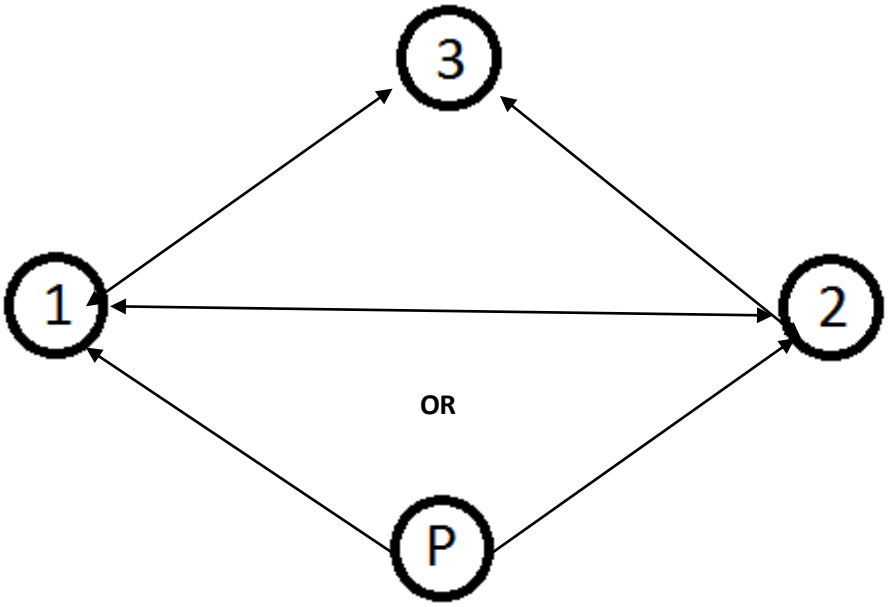
Scene3: *The Player 3 gets excluded by Player 1 and Player 2*



Scene4: *The Player 3 gets excluded by Player 1* ***or*** *Player 2*



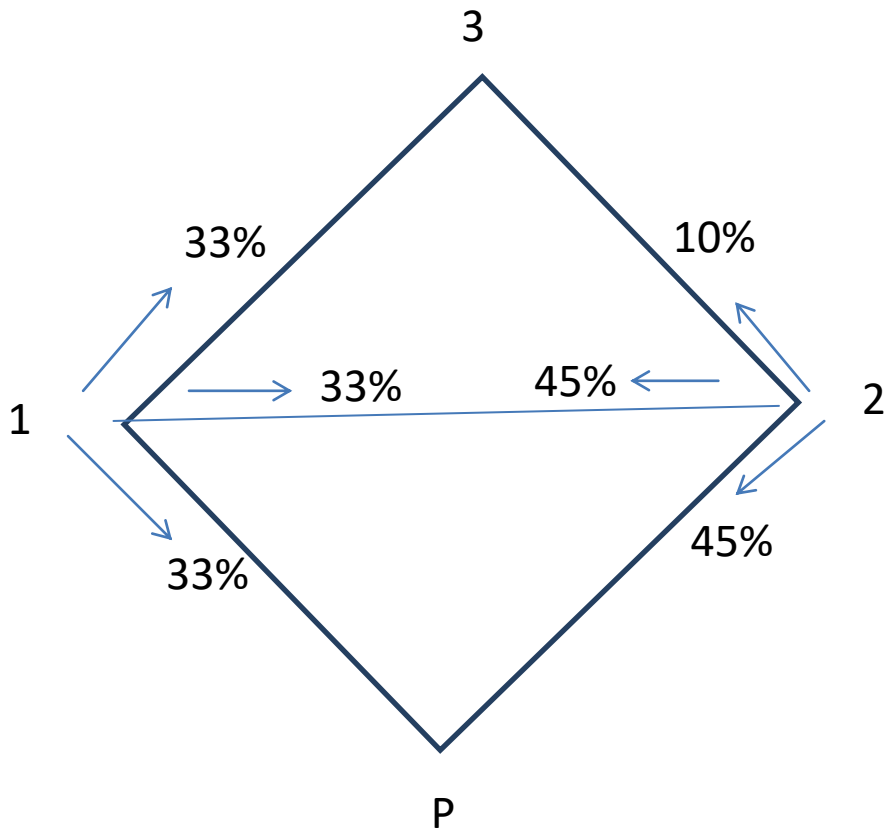
Scene5: *The user (Player P) gets excluded by Player 1* ***or*** *Player 2*



4.1.2 Scenarios Based on Probabilities

In this version, the neuroscientists using the main menu of the game are able to fill in the percentages of the tosses that each player makes to each one of the other players. For example, the neuroscientists can set the probabilities that the Player 1 will pass the ball to the Players 2, 3 and the participant independently of one another, the probabilities that the Player 2 will pass the ball to the Players 1, 3 and the participant, as well as the probabilities that the Player 3 will pass the ball to the Players 1 and 2. In addition, the participant and the Player 3 are not able to throw the ball to each other.

In the graph below we can see a typical paradigm of this version. The neuroscientists using these percentages want to investigate if the participant (P) reduces or increases the number of passes to player 2. The first behaviour indicates an 'altruistic punishment' (other-oriented motive) where the participant (P) is not passing the ball to player 2 because the player 2 is excluding the player 3, while the second behaviour indicates a 'direct reciprocity' (self-oriented motive) where the participant (P) is passing the ball to player 2 because the player 2 includes the participant (P) during the game.



4.1.3 Reinforcement Learning in Cyberball3D+

An **extension of the Cyberball3D+** game was to program players to be dynamically intelligent. For this reason, we implemented the third version of the Cyberball3D+ game. Particularly, we implemented a more sophisticated version of Cyberball3D+ including state of the art Reinforcement Learning methods in Games and incorporating a form of learning affecting players' behavior in Cyberball3D+. This sophisticated version of Cyberball3D+ made the **game more competitive and intelligent** integrating rules and scores as well as creating intelligent opponents (computer players) modeling their decision making behavior.

Two potential rules were the ability of each player to throw the ball to any of the players of the game and also the inclusion of a reward system whereby points were given to or removed from a player depending on his/her ball throws. Particularly, if a player threw the ball to the player that had the highest number of receiving tosses he/she lost a point while if a player threw the ball to the player that had the lowest number of receiving tosses he/she won a point. In addition, if a player threw the ball to a player that had a medium number of receiving tosses he/she did not lose or win points. The player that obtained the highest score of points he/she won the game. The score of the participant was displayed on the screen during the rounds of the game.

Participants were informed that they were able to throw the ball to any player in the game and also that there was a reward system. Although they were not informed on how they gain or lose points, they were able to figure it out on their own, if they paid attention on the pointing system since the score appear on the screen. The **purpose of this approach** was to make the players to understand that they have to feel empathy for the excluded players throwing the ball to them if they want to win the game.

We created intelligent opponents (computer players) using a Reinforcement Learning Algorithm. The algorithm that we used was the Q-Learning algorithm. The **Q–learning method** is a Reinforcement Learning algorithm that updates the value of a state-action pair after the action has been taken in the state and an immediate reward has been received **(Watkins & Dayan 1992)**. Values of state-action pairs, Q(s, α) are learned since the resulting policy is more easily recoverable than learning the values of states alone, V(s). **Q-learning algorithm** will converge to an optimal value function under conditions of sufficiently visiting each state-action pair, but often requires many learning episodes to do so.

We assumed that the Cyberball3D+ game has thirty six states, three actions that can be taken and a reward system that is described above. The thirty six potential states are the **Sk** where the factor **k** belongs to the range from '1' to '36'. The situation of the environment for each state **Sk** is described in the table below. The **key C** indicates who player has the ball, the **keys H, M and L** indicate if the player has the highest, medium or lowest number of receiving tosses.

TABLE I: The states of the environment in the Reinforcement learning version of the game.

State	Participant (P)	Player1	Player2	Player3
S**1**	C	H	M	L
S**2**	C	H	L	M
S**3**	C	H	L	L
S**4**	C	M	H	L
S**5**	C	L	H	M
S**6**	C	L	H	L
S**7**	C	M	L	H
S**8**	C	L	M	H
S**9**	C	L	L	H
S**10**	H	C	M	L
S**11**	H	C	L	M
S**12**	H	C	L	L
S**13**	M	C	H	L
S**14**	L	C	H	M
S**15**	L	C	H	L
S**16**	M	C	L	H
S**17**	L	C	M	H
S**18**	L	C	L	H
S**19**	H	M	C	L
S**20**	H	L	C	M
S**21**	H	L	C	L
S**22**	M	H	C	L
S**23**	L	H	C	M
S**24**	L	H	C	L
S**25**	M	L	C	H

S**26**	L	M	C	H
S**27**	L	L	C	H
S**28**	H	M	L	C
S**29**	H	L	M	C
S**30**	H	L	L	C
S**31**	M	H	L	C
S**32**	L	H	M	C
S**33**	L	H	L	C
S**34**	M	L	H	C
S**35**	L	M	H	C
S**36**	L	L	H	C

The three actions that could be taken in a state were the throws of the ball to any of the rest of the players of the game (the game consisted of four players).

In the Q-Learning algorithm when an action α is taken in state s, the value of a state-action pair, or Q-value, is updated as $Q(s, \alpha) = Q(s, \alpha) + a(r + \gamma V(s') - Q(s, \alpha))$

Where $\alpha \in [0, 1]$ is the learning rate, r is reward that is observed, γ is the discount factor, s' is the next state, and $V(s') = \max_{a'} Q(s', \alpha')$.

We defined the value of the learning rate 'a' to be equal to 1 because our environment is full deterministic and it has a finite number of steps. In addition, we defined the value of the discount factor '*γ*' to be equal to 1 due to our game is played to a specified period, which is determined by neuroscientists.

The best using of the Reinforcement learning system in our project was to train first the opponents (virtual players) with the rules and the reward that we have already implemented in order to learn the best strategy to win the game and in the second phase insert the trained opponents inside our game in order to explore the behavior and the brain activation of the participant when he was trying understand and learn the environment. However, we decided **the learning of the environment** from the opponents (virtual players) and the participant to be done **at the same time** because the system is a fully deterministic environment.

The intelligent opponent (computer player) **acquired the necessary knowledge** about which action was the best choice for a given state after some learning episodes on each state-action pair, by **observing the points** that he gained/lost and also the overall number of receiving tosses of each player.

4.2 Creating the 3D Virtual Scenes

The 3D Cyberball3D+ game was developed with the Unreal Development Kit (UDK) allowing one user in the fMRI scanner to play with three players modeled in three levels of anthropomorphism while undergoing an fMRI scan examining the neural basis of empathy for social exclusion as an effect of level of anthropomorphism.

The UDK is a framework used mostly in creating computer games and visualization. UDK consists of different parts, making it act both like a game engine and a 3D authoring environment. It provides the necessary tools to create 3D objects and assign materials on these, import 3D objects such as characters and their animations from 3ds Studio Max and import and use sounds and sound effects. It, also, allows the designed application to seemingly attach to Flash User Interfaces (UI). UDK can also be used to render computer graphics scenes as well as create and respond to events while playing the game. UDK offers the ability to use both C/C++ and UnrealScript, which provides the developers with a built-in object-oriented programming language that maps the needs of game programming and allows easy manipulation of the actors in a synthetic scene. The main components inside the UDK are: The Unreal Editor which is used to create or import objects and edit VEs handling all the actors and their properties located in the VEs; the Unreal Kismet, which allows for the creation of sequences of events and corresponding actions and the Unreal Matinee which is responsible for the animation of actors or real-time changes in the actors' properties.

3ds Studio Max modeling software and the Adobe Flash Professional platform were also employed. 3ds Studio Max provides the necessary tools to create 3D objects such as virtual characters and assign materials on these. Additionally, it provides the tools to create animation on 3D objects by applying a Physique or skin modifier. Also, 3ds Studio Max exports the 3D objects in FBX format supported by UDK. 3D characters of varied anthropomorphism were modeled and animated for catching and throwing the ball (Figures 28, 31 and 34). Motion capture data were not utilized and the movement was as natural as possible. The derived 3D objects and animations were imported in UDK, by selecting the import option in the asset library of the Unreal Editor.

The Adobe Flash Professional is a Flash authoring environment. A Flash application can be used as a User Interface in UDK, because it incorporates the ability to display animated graphics, text or buttons on the screen on top of the scene being rendered. It can also receive user input and provide feedback according to it. A Flash application consists of many frames, placed in the main timeline. The flow of the frames being displayed can be

changed through ActionScript - Flash's scripting language. Each frame can have its own set of graphics, texts, movie clips and buttons and a script controlling the behavior of the frame's components. The integration of a Flash application in a UDK scene requires that it should first be compiled into an SWF file and then imported in UDK's asset library. Afterwards, either UnrealScript or Unreal Kismet can initiate the Flash application, interact with it, hide it or instruct it to stop playing. While a Flash application is playing in a scene, UnrealScript can initiate a call of an ActionScript function and vice versa. We used the Adobe Flash Professional to create many User Interfaces such as the initial and return menu as well as the interface that depicted the scores of the players in the Reinforcement Learning version of the game.

In order to create the 3D scene that the application would render, several steps were required, from creating the actual 3D objects placed in the scenes, to importing them inside UDK and placing them in a synthetic scene, as well as creating and assigning the appropriate materials and apply animations to these objects. These steps will be further explained below.

4.2.1 Creating Visual Content

The synthetic scene of the Cyberball3D+ game consists of 4 players and one ball. Particularly, the four players are positioned on the scene so as to form a rhombus, e.g. they are placed on the vertices of a rhombus. Player 1 is placed on the left side, Player 2 on the right and directly opposite of the participant in the fMRI scanner is Player 3. Three levels of anthropomorphism (low, medium and high) were employed and in each round of the game the appropriate 3D characters of the scene (Figures 45, 46 and 47) were displayed based on the parameters selected by the neuroscientists as entered in the initial menu. The 3D characters of the 'low level' of anthropomorphism (Figure 28) consist of human form of the face and the body but their gender is not distinguishable. This was considered as the lowest fidelity 3D character. While, the gender of the 3D characters of the 'medium level' of anthropomorphism (Figure 31) is distinguishable, however, they do not have hair or beard and they wear uniform clothing. Finally, the 3D characters of the 'high level' of anthropomorphism (Figure 34) consist of high fidelity human characteristics and wear distinguished clothes. The low-level 3D characters and their animations were modeled in 3ds Studio Max. The 3D characters of medium and high level of anthropomorphism were downloaded from 3D model's repositories; however, the animations of the ball-throwing were created in 3ds Studio Max.

Especially, the animations of the 3D characters such as the ball-throwing and ball-catching to all directions were modeled inside the 3ds Studio Max only once using the low level 3D character and were saved in a .bip format. Then, each different animation was imported to all characters of all levels of the anthropomorphism through motion editor of 3ds Studio Max, separately and was exported in an .fbx format as is the format that UDK supports. In the next state of the implementation of the game the derived 3D objects and animations were imported in UDK in an .fbx format by selecting the import option in the asset library of the Unreal Editor. The modeling of the characters of all levels of the anthropomorphism will be more explained and depicted in figures analytically below.

A. "Low level' of anthropomorphism

The 3D characters of the 'low level' of anthropomorphism as described above consist of human form of the face and the body but their gender is not distinguishable. For the modeling of the lowest fidelity 3D character were used two sketches that were depicted the front and the right view of a human silhouette, respectively. These sketches were imported and positioned on the scene of 3ds Studio Max so as to form a vertical angle. Using the Editable Poly Modifier of the 3ds Studio Max we started to create small polygons and connect them to each other based on these two sketches until a human silhouette be modeled.

Figure 18 shows two viewpoints of the 3ds Max scene that depicts the two sketches and a part of the final model of the low level 3d character. In this figure, the human body is depicted without the head, the hands and the foot. In the next steps, were modeled the rest parts of the body as shown in Figures 19, 20, 21, 22, 23.

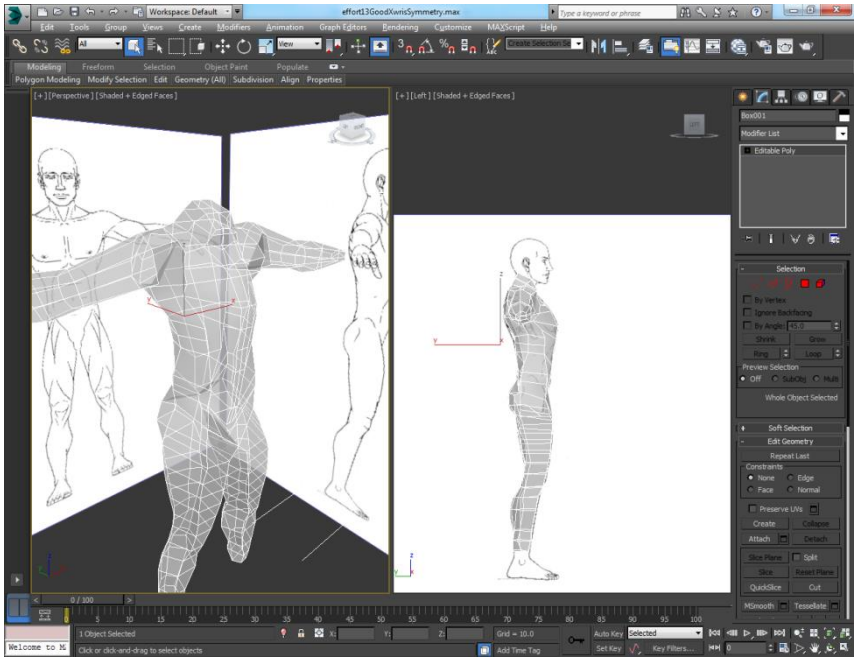


Figure 18: A 3d representation from the process of the low level character modeling.

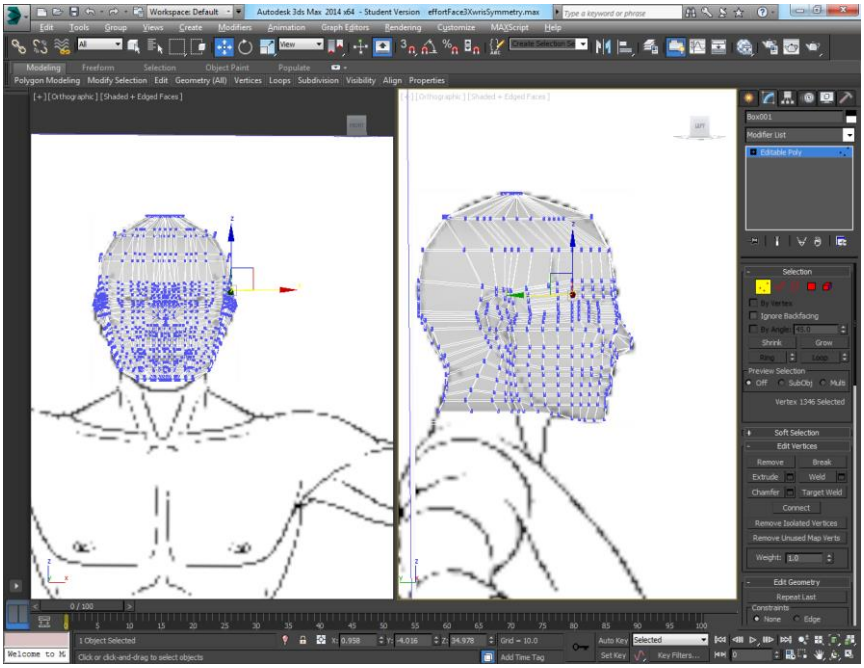


Figure 19: A 3d representation from the process of the modeling of the head of the low level character.

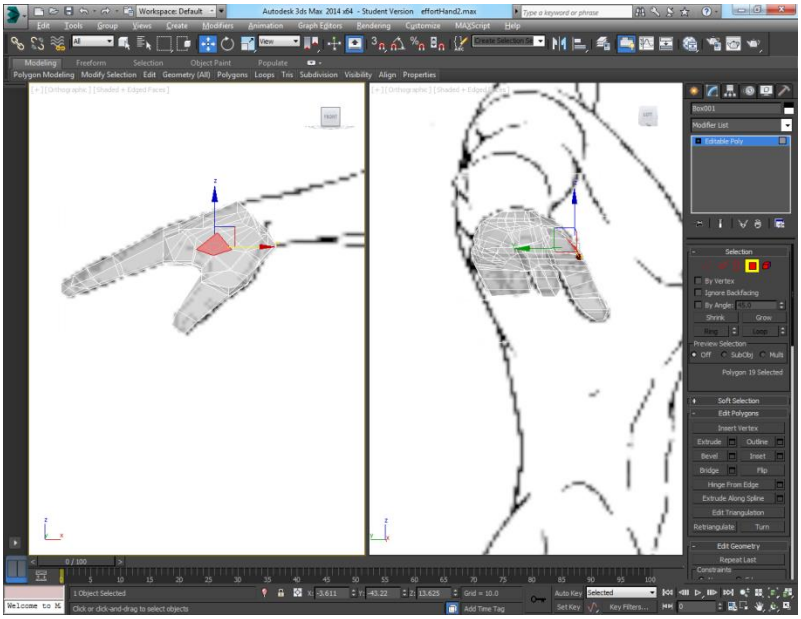


Figure 20: A 3d representation from the process of the modeling of the hand of the low level character.

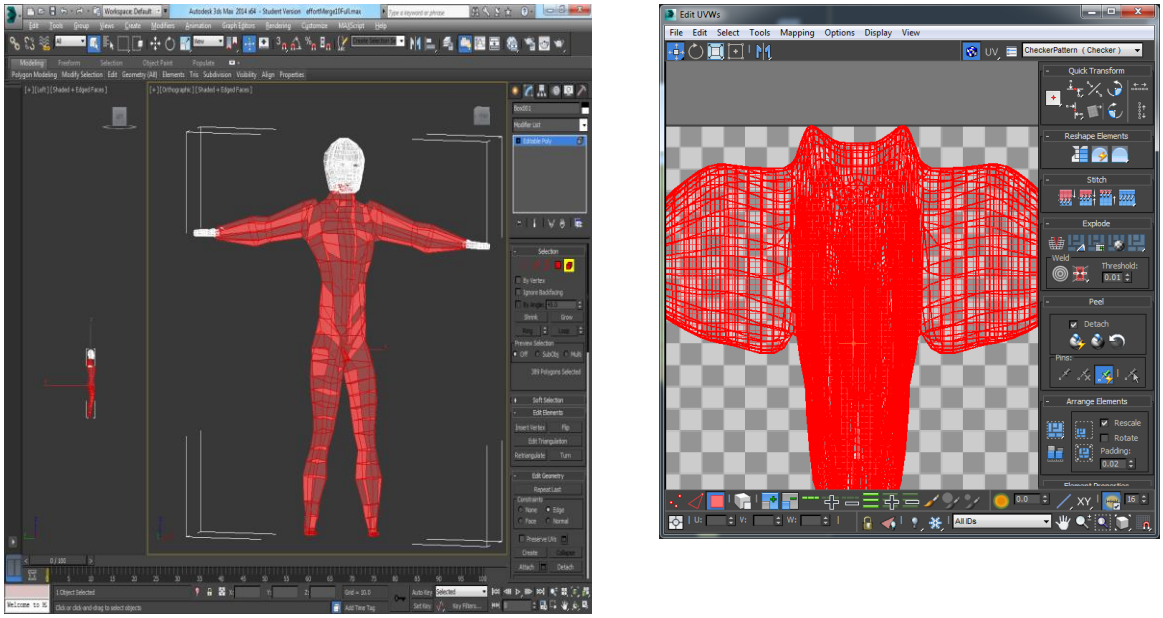


Figure 21: The left image represents the low level character without UVW modifing, while the right image shows a snapshot during the application of the Unwrap UVW modifier to the sub-object selections of the character.

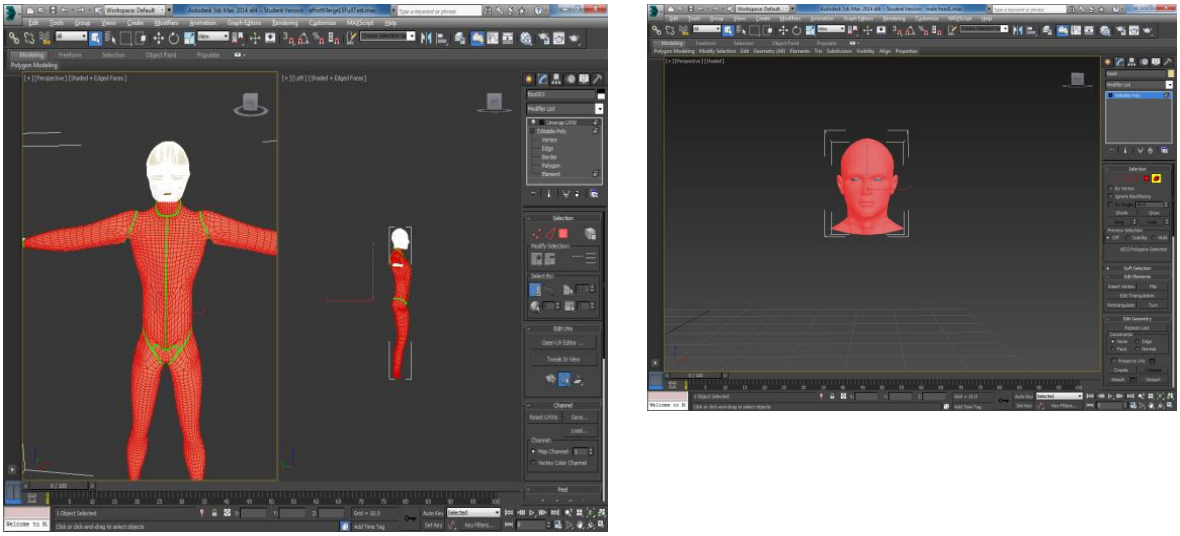


Figure 22: The left image represents the result of the UVW modifier. The right image shows the 3d object that finaly were used for the head of the low level character.

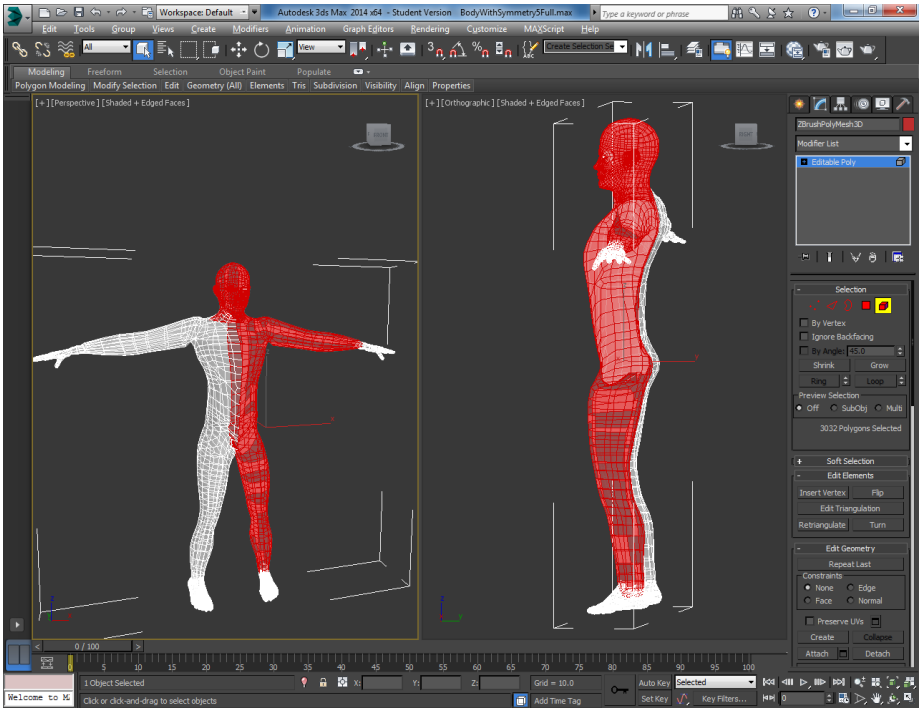


Figure 23: The screenshot represents the final 3D model of the low level 3D character.

After the modeling of the 3D character using the Editable Poly, a Unwrap UVW modifying is necessary to assign mapping (texture) coordinates to objects and sub-object selections. The Unwrap UVW modifier is used to assign planar maps to sub-object selections and to edit the UVW coordinates of those selections. Existing UVW coordinates on an object can be unwrapped and edited as well. Maps can be adjusted to the proper fit on a Mesh, Patch, Polygon, HSDS, or NURBS model. Also, the Unwrap UVW

modifier can be used as a self-contained UVW mapper and UVW coordinate editor, or in conjunction with the UVW Map modifier.

Although, the modeling of the body of the character is done, is required the application of a Skin or Physique Modifier for the modeling of the animations of the character. Particularly, it is necessary to associate the character mesh with the biped's skeletal parts. This is accomplished through a process called skinning. Just as the human skin is moved by the motion of our bones and muscles, the character is animated by means of a skinning modifier that deforms the mesh according to the rotation and position of the objects in the biped hierarchy. After the biped is correctly associated with the mesh (skinned), each part of the biped acts just like a bone inside a body.

At the time the mesh is skinned, vertices on the mesh are associated with one or more bones via a weighting system. A capsule-shaped envelope is created around each bone based on the bone's size as shown in Figure 24. Vertices that fall within a bone's envelope receive some weight from it, meaning they are moved by it. A vertex that falls within the capsules of more than one bone receives appropriate weights from each bone.

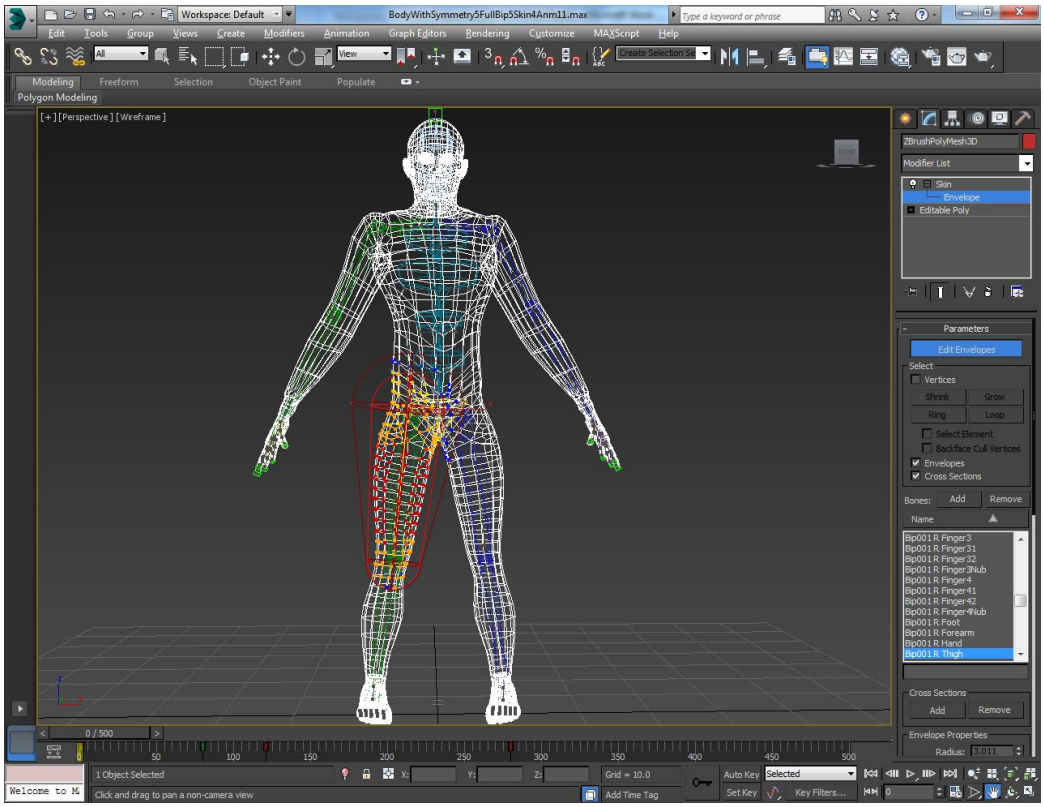


Figure 24: The representation of the applying of the Skin Modifier and the capsule-shaped envelope that was created around of a thigh bone.

When the Skin or Physique Modifier applied to the character, then next step for the modeling of the animations of the character is to create the movements using the *Auto-Key* and the *Animation Timeline.* The animation timeline is the numbered bar just below the animation views and the numbered increments represent the number of frames set for an animation. Using the Auto Key Mode, when a change to objects' position, rotation, and scale made, the 3ds Max creates a key storing the new value for the changed parameter at the current time. The 3ds Max automatically fills in the frames between the different keys.

For the purposes of the Cyberball3D++ game we implemented the ball-throwing and the ball-catching animations to all directions such as the catching/throwing of the tossing ball from/to the left or right or front player.

The animations were modeled only once using the low level 3D character and were saved in a .bip format (Figure 25). Then, for the modeling of the animations of the rest 3D characters the .bip animations were loaded through the Motion Editor and its loading action for a .bip file (Figure 26). Finally, all characters and their animations separately were exported in an .fbx format as is the format that UDK supports (Figure 27).

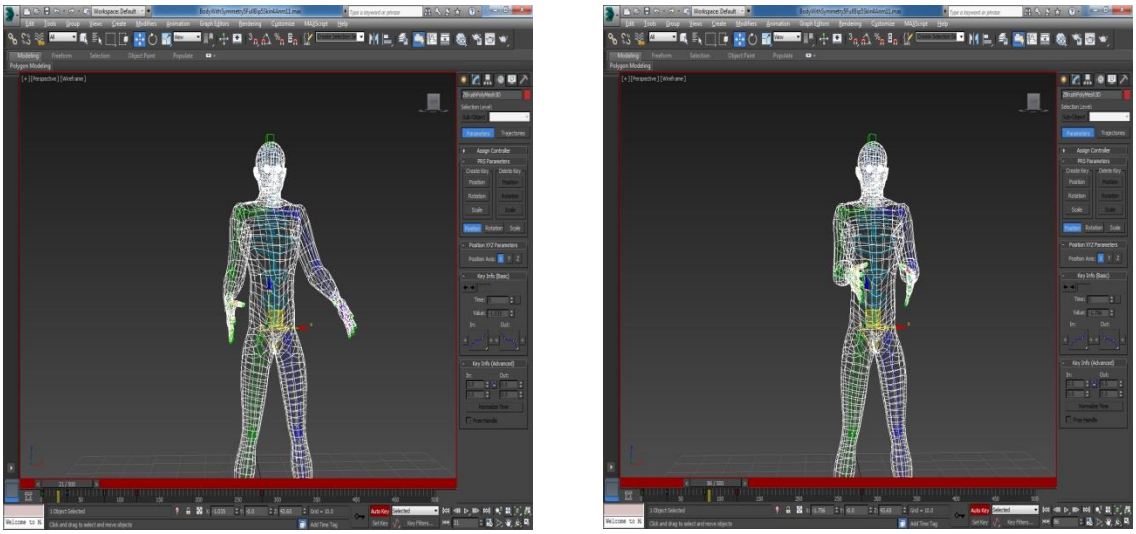


Figure 25: The screenshots represent a paradigm of the modeling of an animation.

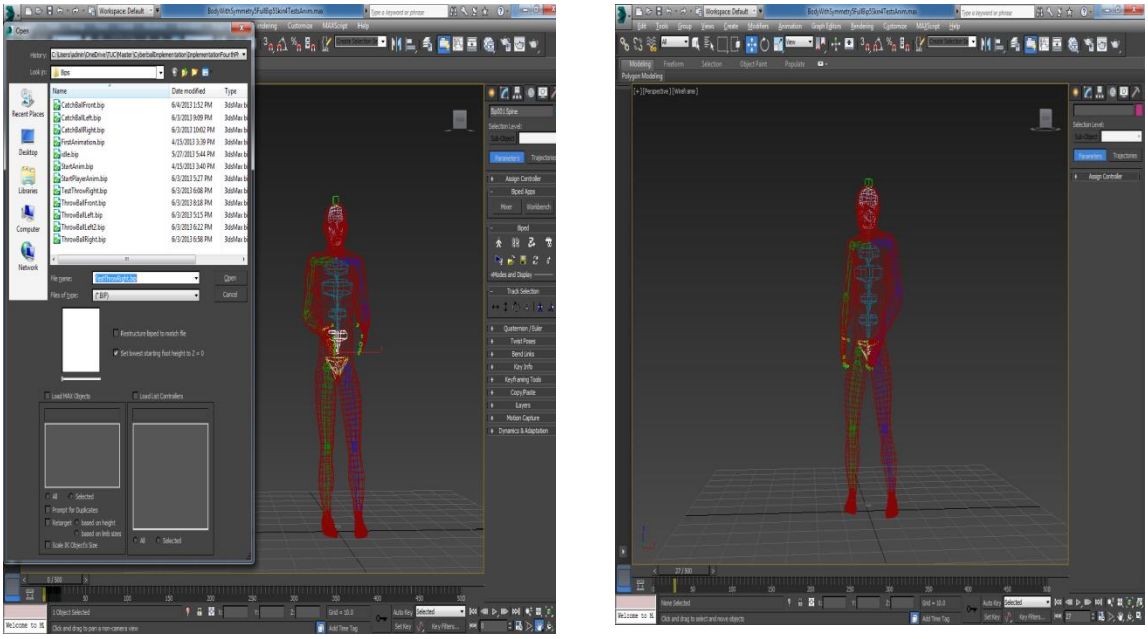


Figure 26: The screenshots depict the loading of a .bip file.

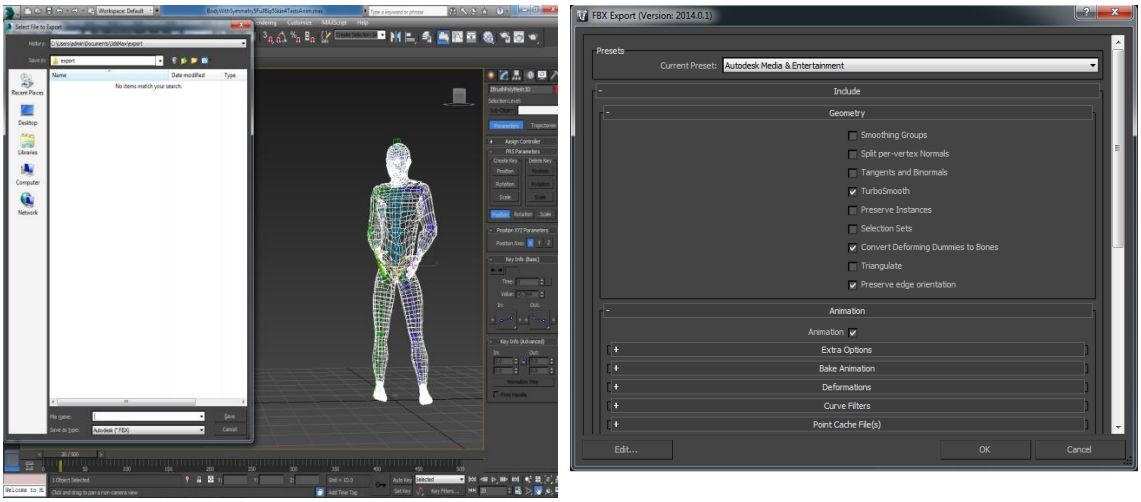


Figure 27: The screenshots depict the exporting of a low level character as well as an unique animation.

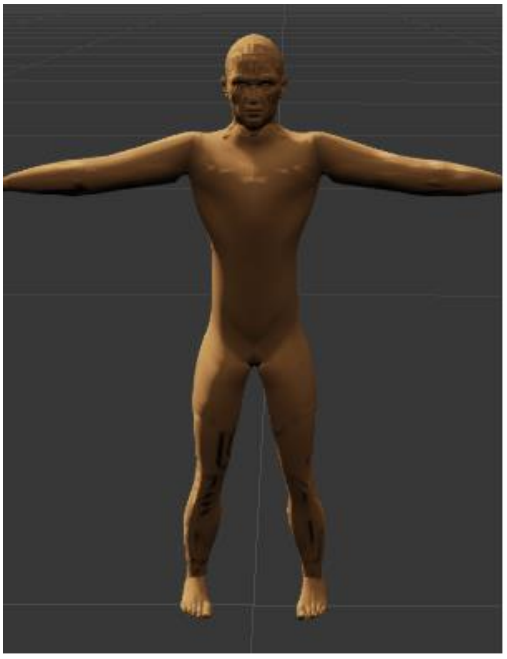


Figure 28: 3D representation of the character of the "low level" anthropomorphism.

B. "Medium level' of anthropomorphism

As described above, while, the gender of the 3D characters of the 'medium level' of anthropomorphism is distinguishable, however, they do not have hair or beard and they wear uniform clothing. The 3D characters of medium level of anthropomorphism were downloaded from 3D model's repositories (Figure 29); however, the animations of the ball-throwing and the ball-catching were modeled in 3ds Studio Max loading the .bip files that are created using the lowest fidelity 3D character (Figure 30).

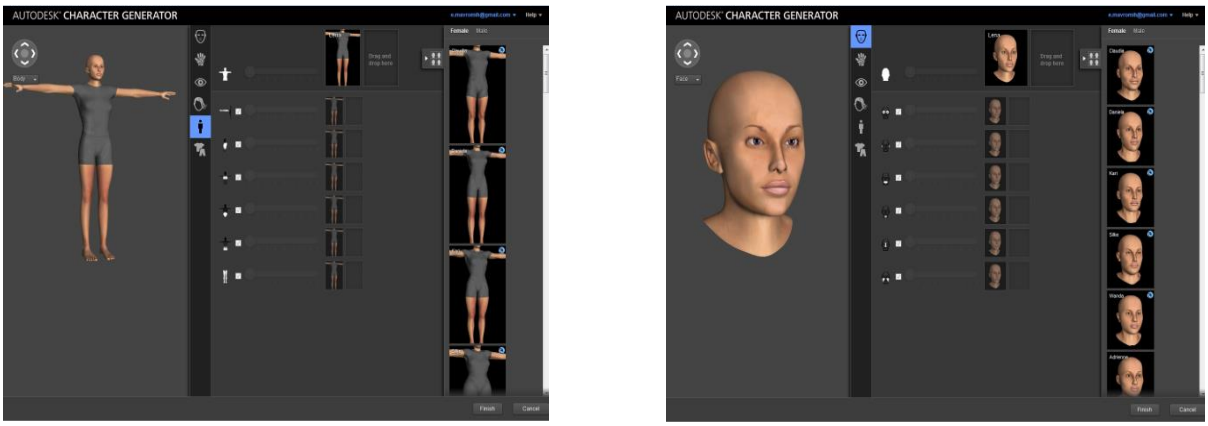


Figure 29: Screenshots that represent the process of generate a 3d model using the 3D model's repositories.

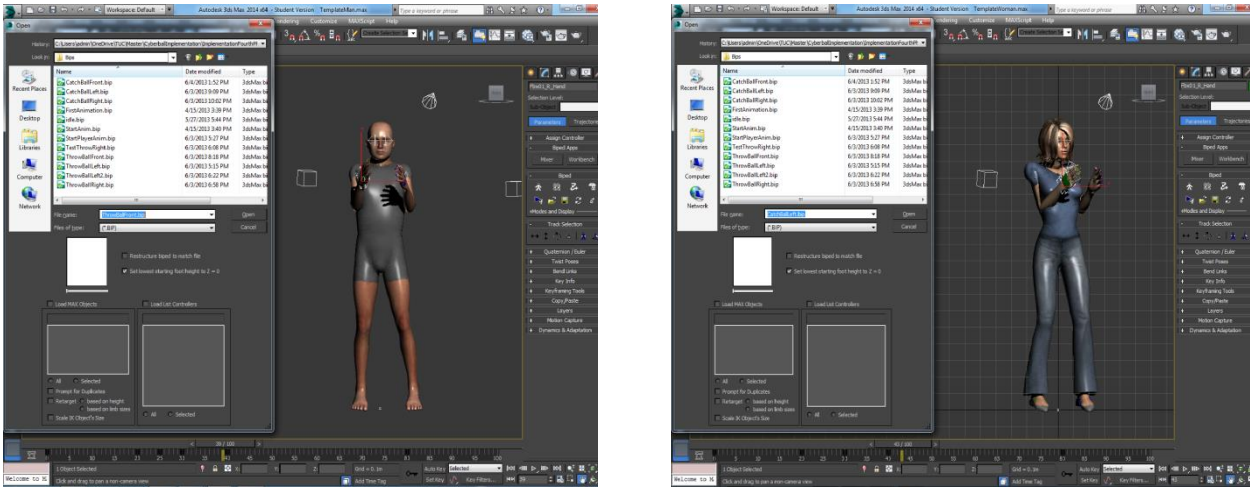
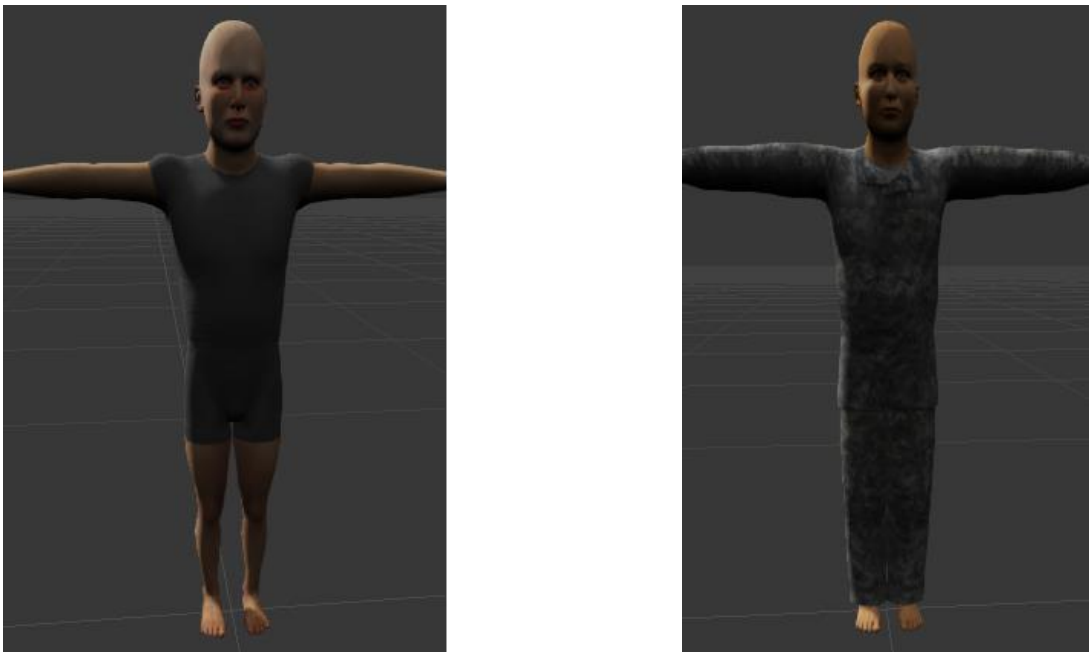


Figure 30: These images decpict two paradigms of loading two different animations using the .bip files.



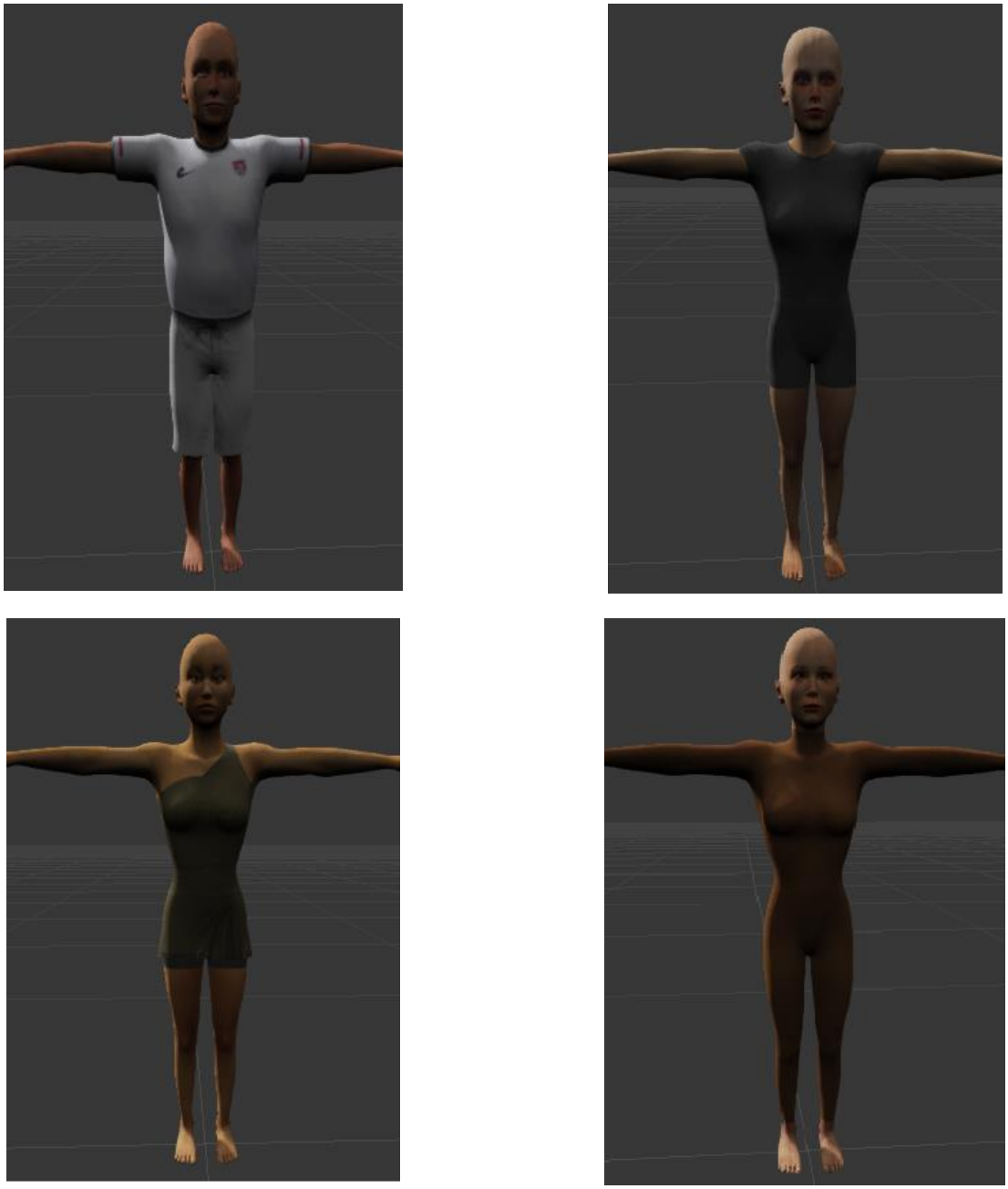


Figure 31: The six images above represent the 3d characters of the medium level of anthropomorphism that were used in the game.

C. "High level' of anthropomorphism

Finally, the 3D characters of the 'high level' of anthropomorphism consist of high fidelity human characteristics and wear distinguished clothes. The process of the modeling of these characters and their animations was the same as the process that was following for the medium level anthropomorphism characters.

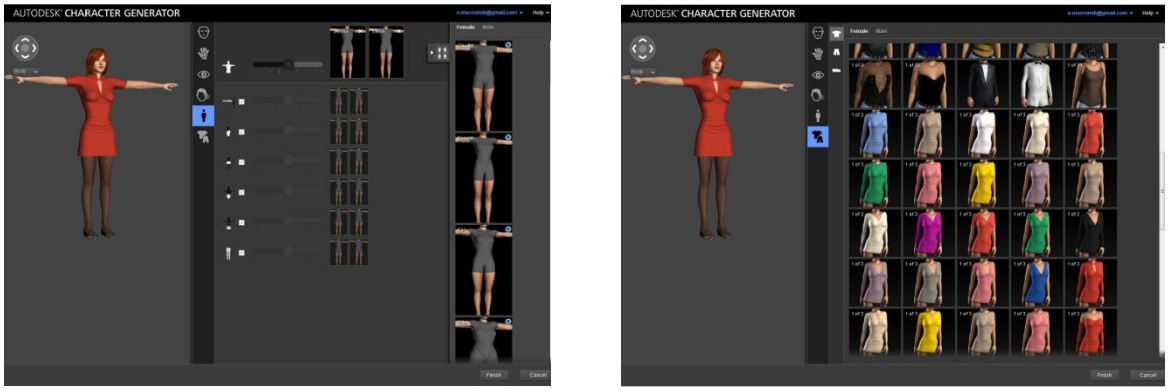


Figure 32: Screenshots that represent the process of generate a 3d model using the 3D model's repositories.

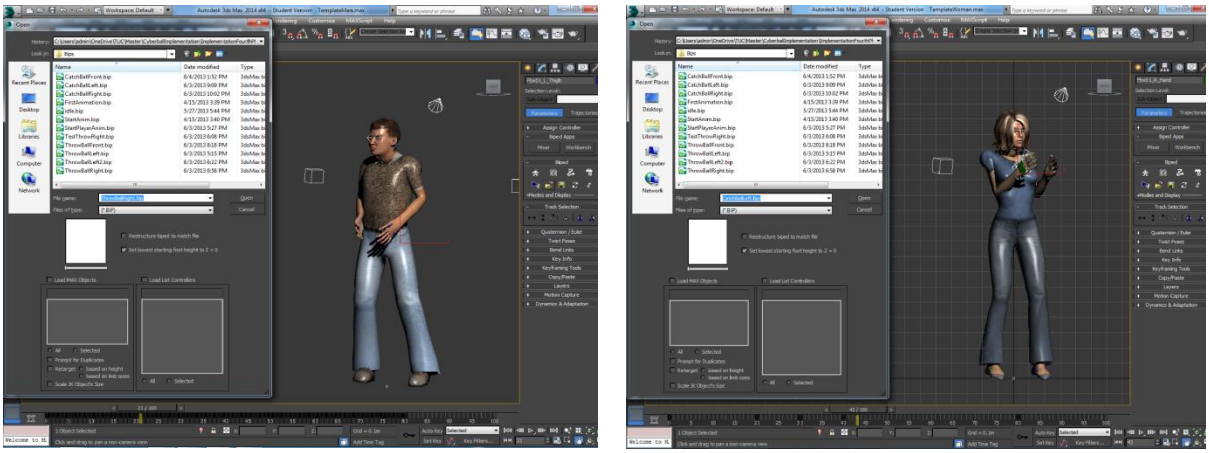


Figure 33: These images decpict two paradigms of loading two different animations using the .bip files.

Figure 34: The six images above represent the 3d characters of the high level of anthropomorphism that were used in the game.

D. Modeling of the ball

The ball was downloaded from a 3d model repository but the changes of the model were made through the Edit Poly modifier (Figure 35). The 3d model was exported in a .fbx format and imported in UDK by selecting the import option in the asset library of the Unreal Editor. The movements of the ball such as the tossing or catching of the ball from or to all other players were modeled using the Matinee tool of the Unreal Editor (Figures 37 and 38). Particularly, we created different Matinee events for each animation of the ball by designing separate Matinee sequences for each different direction of movement of the ball from a player to all other players (Figure 36).

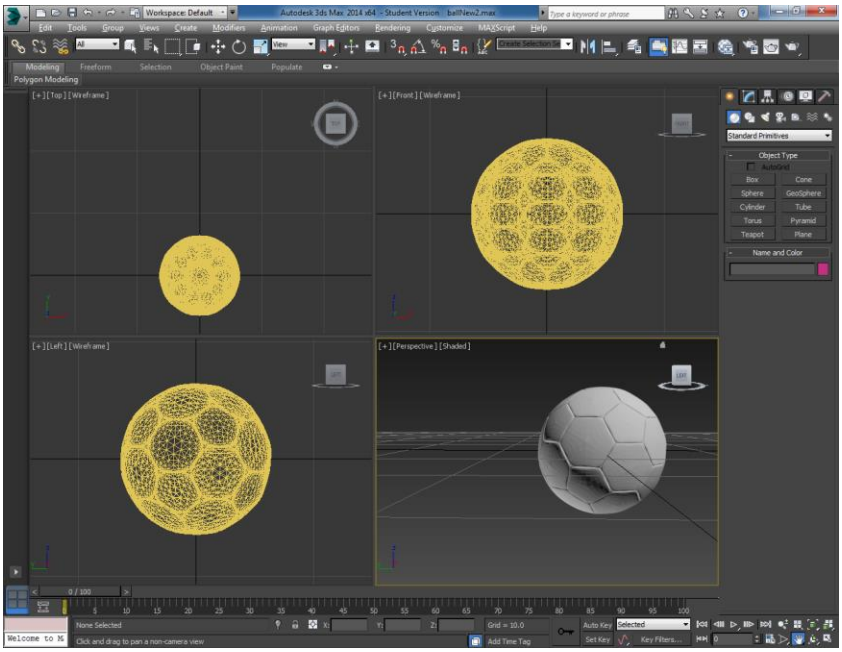


Figure 35: 3d representation of the modeling of the ball inside the 3ds Max.

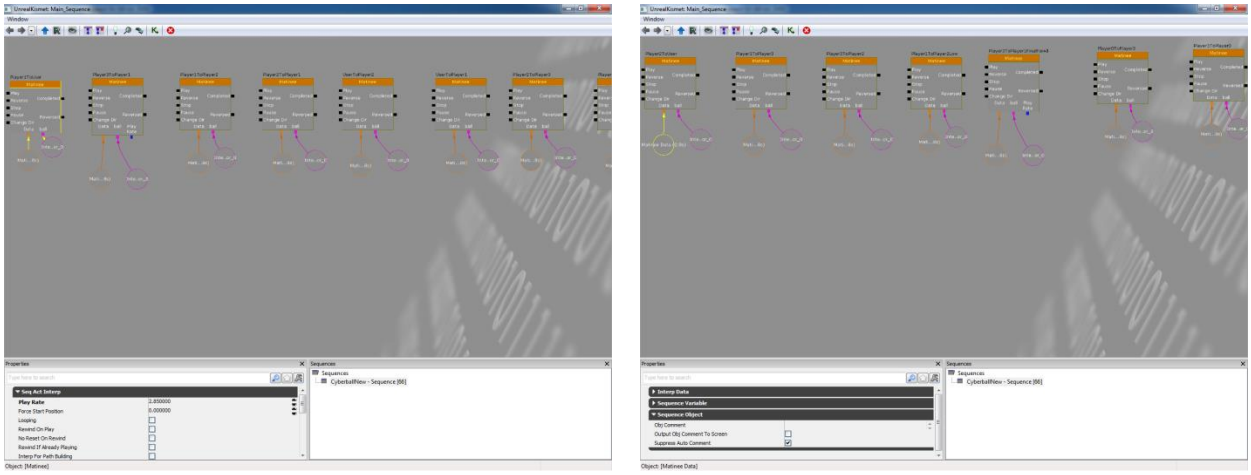


Figure 36: The images depict all Matinee events that created for each animation of the ball.

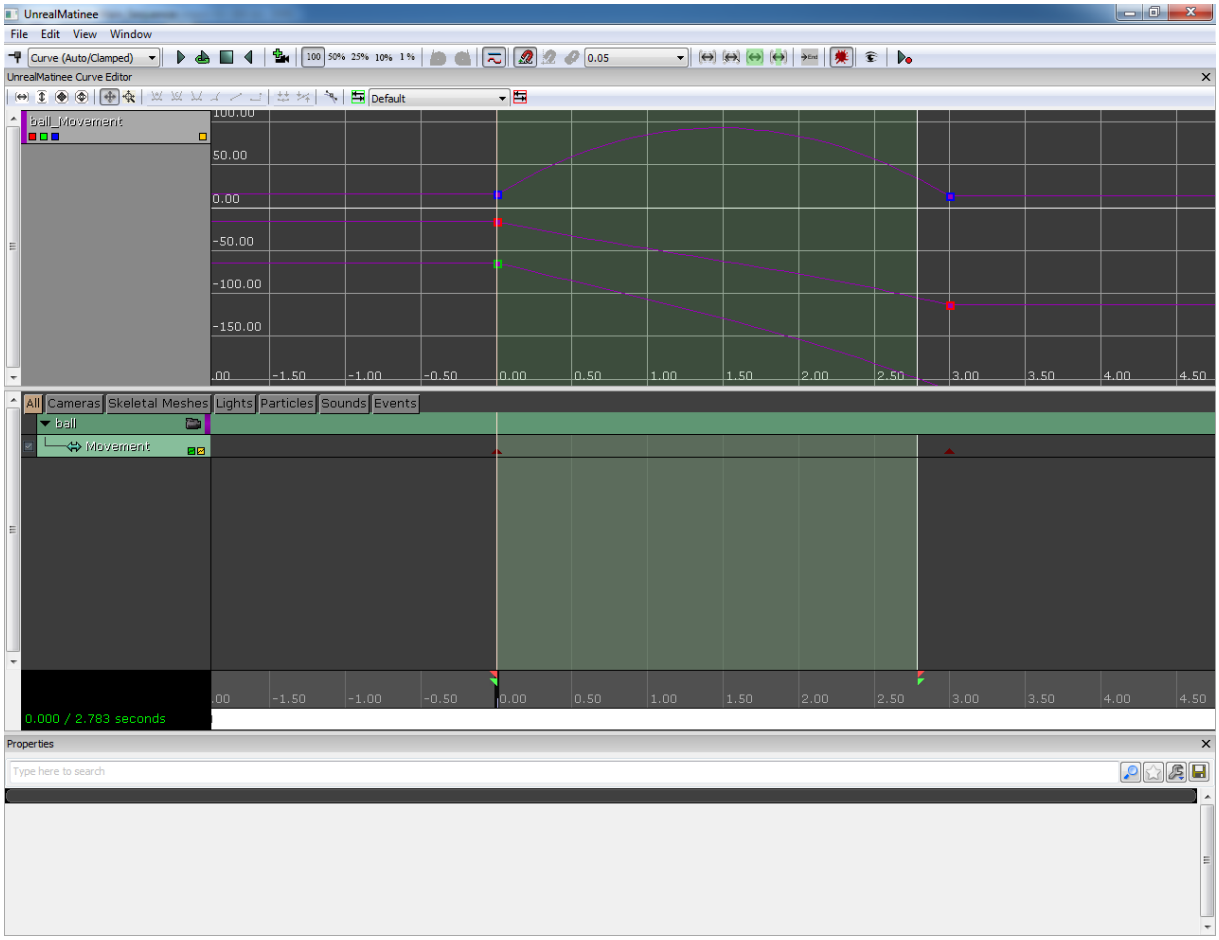


Figure 37: An example of creating a ball animation using the Unreal Matinee Editor.

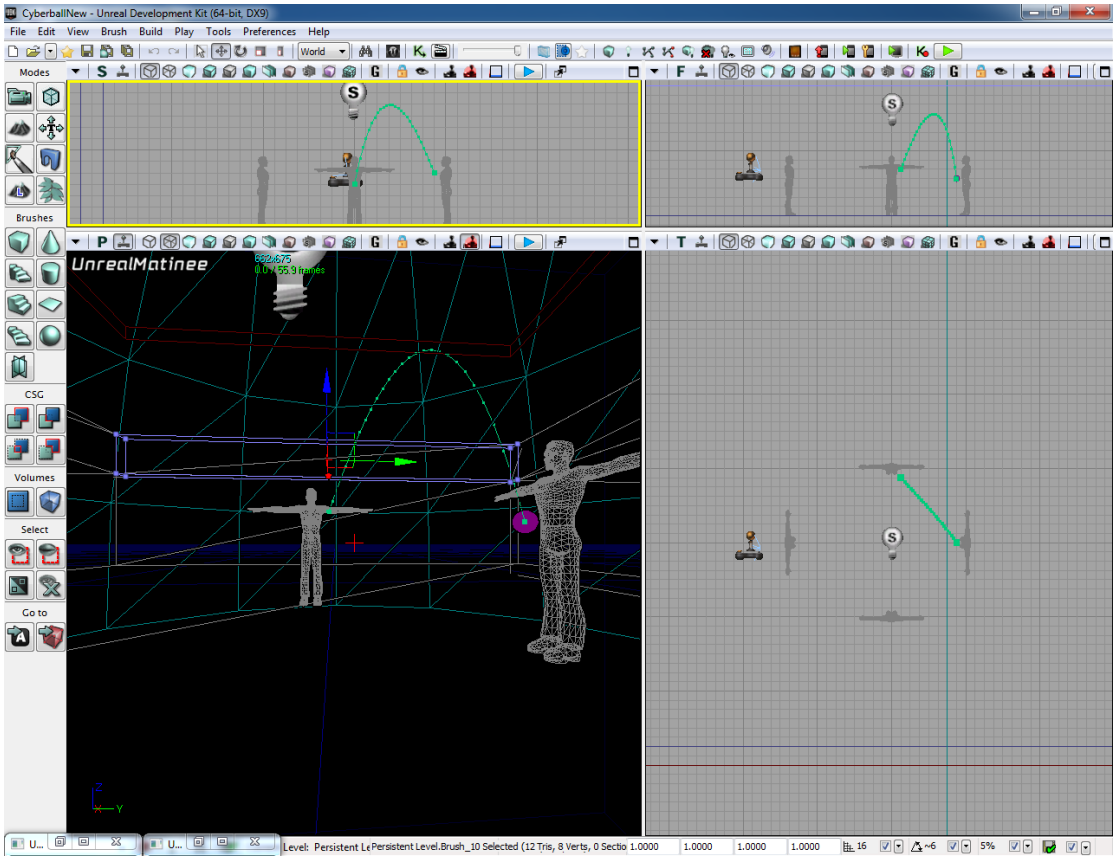


Figure 38: 3d representation of a ball animation inside the scene of the Unreal Editor.

4.2.2 Setting up the Scene in UDK

Inside UDK, the created 3D objects such as the 3D characters and the ball were imported, by selecting the import option in the Asset library of the Unreal Editor. UDK reads the file containing the exported 3D objects and recreates the geometry of the objects. It also creates the different material slots for each part of the 3D object and initializes the UV channels of the object. The 3d characters were imported as Skeletal Meshes while the ball as Static Mesh. In addition, the animations of each character as well as the sounds that were used during the game were imported inside the Content Browser. Particularly, the AnimSet Editor of each Skeletal Mesh was using to import all animations in an .fbx format for each human character, separately.

In the (Figures 39, 40, 41, 42 and 43), were represented the Content Browser that contains all 3d models, the importing option inside the Content Browser as well as the AnimSet Editor, the Material Editor, and the Sound Cue Editor. Furthermore, The Content Browser is the primary area of the Unreal Editor for creating, importing, organizing, viewing, and modifying content assets within Unreal Editor. It also provides the ability to manage content folders and perform other useful operations on assets, such as renaming, moving,

copying, and viewing references. The Content Browser can search for and interact with all assets in the game.

In addition, the AnimSet Editor is responsible for the changing of the main attributes of a Skeletal Mesh, such as its clothes, materials and its movements. These changes will affect the instances of this object that will be inserted in the virtual scene, unless they are overridden. The Material Editor Tool is responsible for creating and editing the different materials that can be assigned to objects inside the scenes created, while the Sound Cue Editor is a node-based editor that is used to work with audio and provides the necessary tools, in order to create various sound effects.

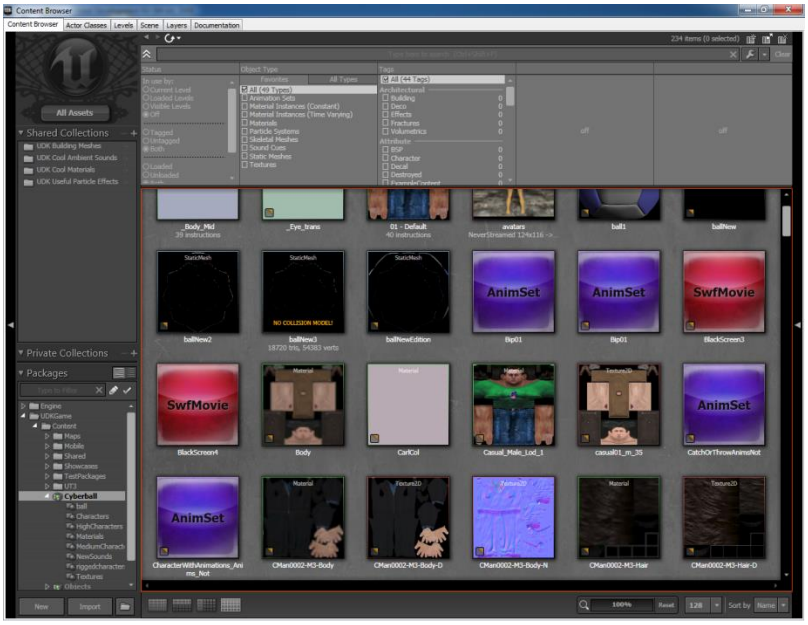


Figure 39: The Content Browser.

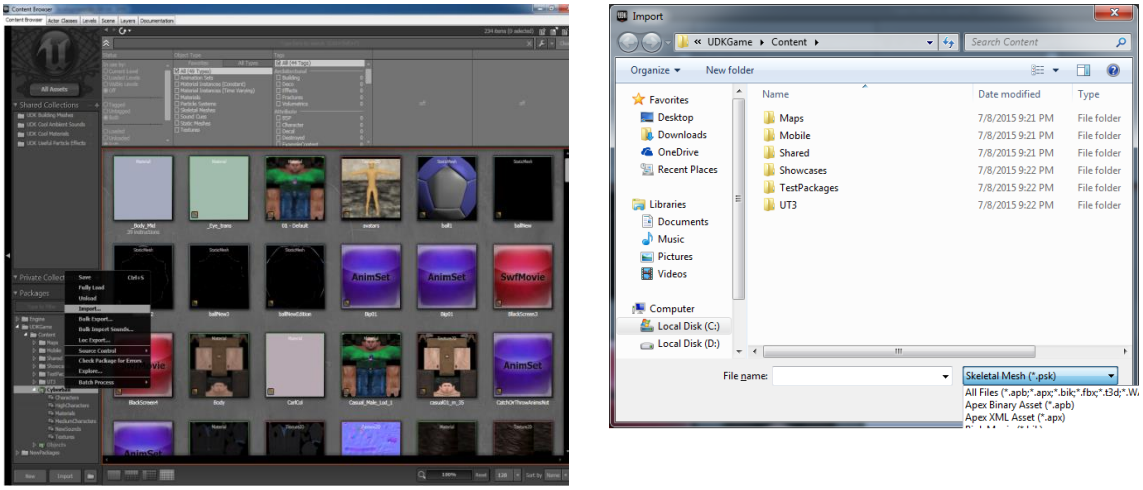


Figure 40: The importing option inside the Content Browser.

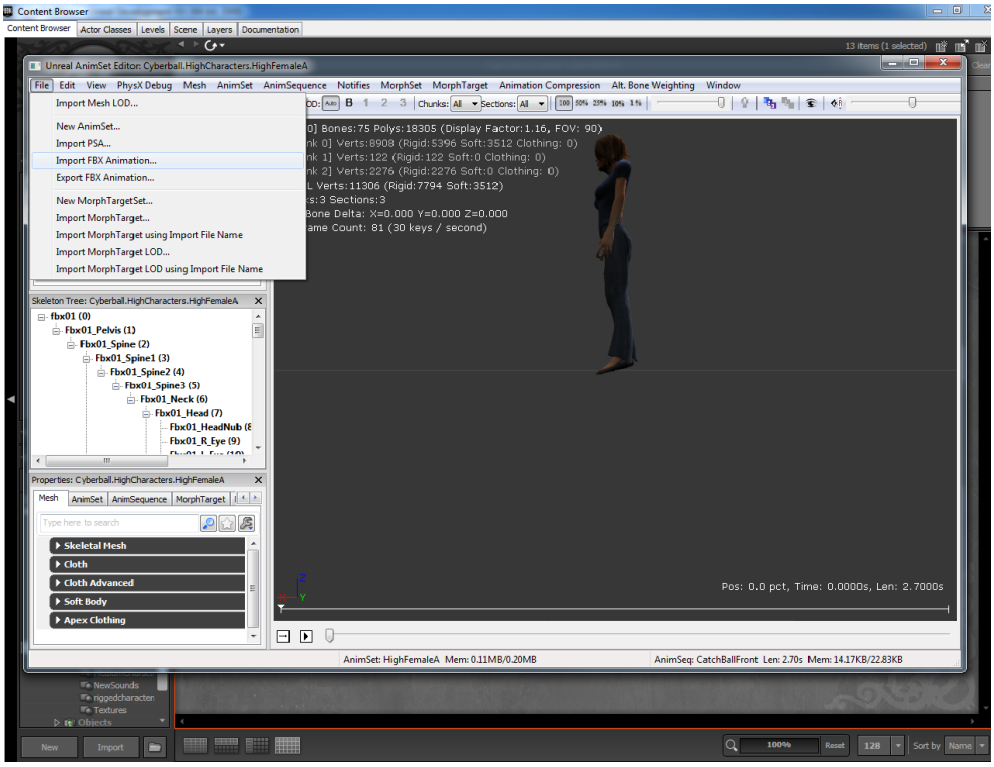


Figure 41: The importing option of a fbx animation inside the AnimSet Editor.

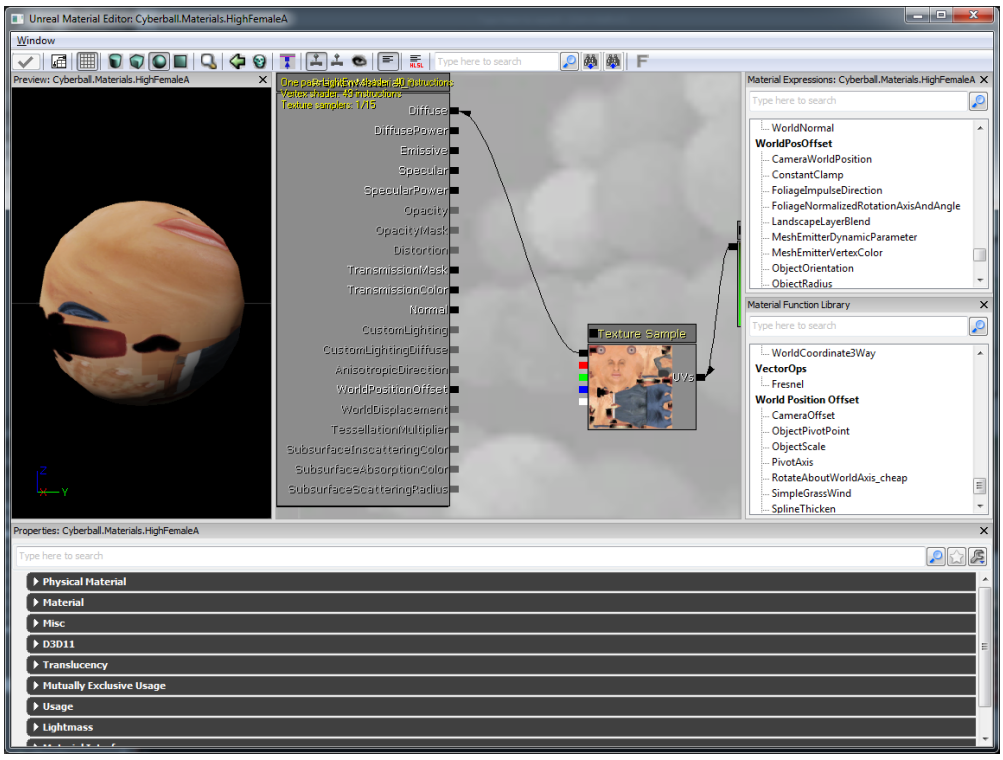


Figure 42: Material Editor depicting the overall material that was used for a 3d model.

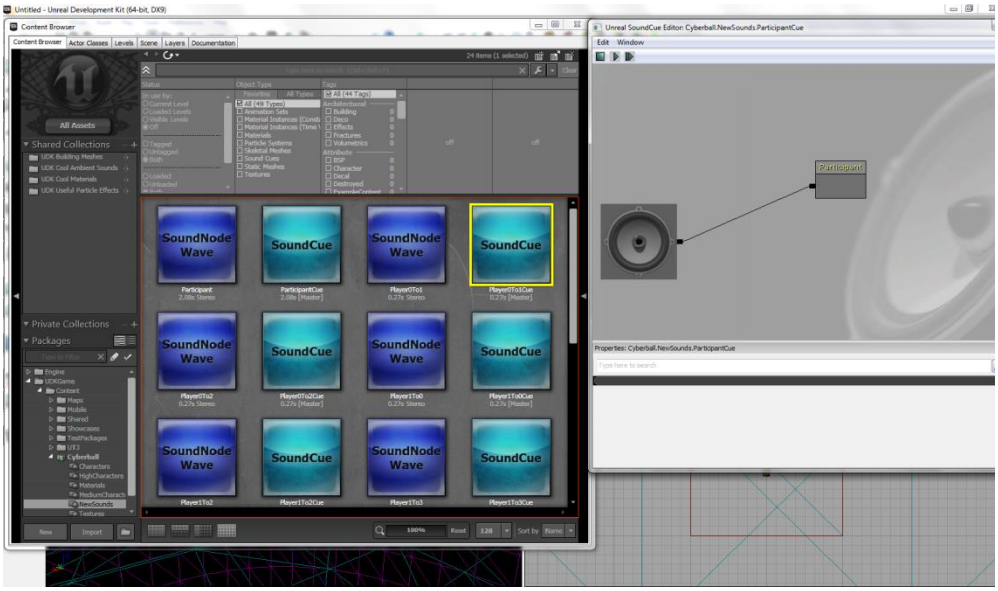


Figure 43: One Sound node inside the Sound Cue Editor.

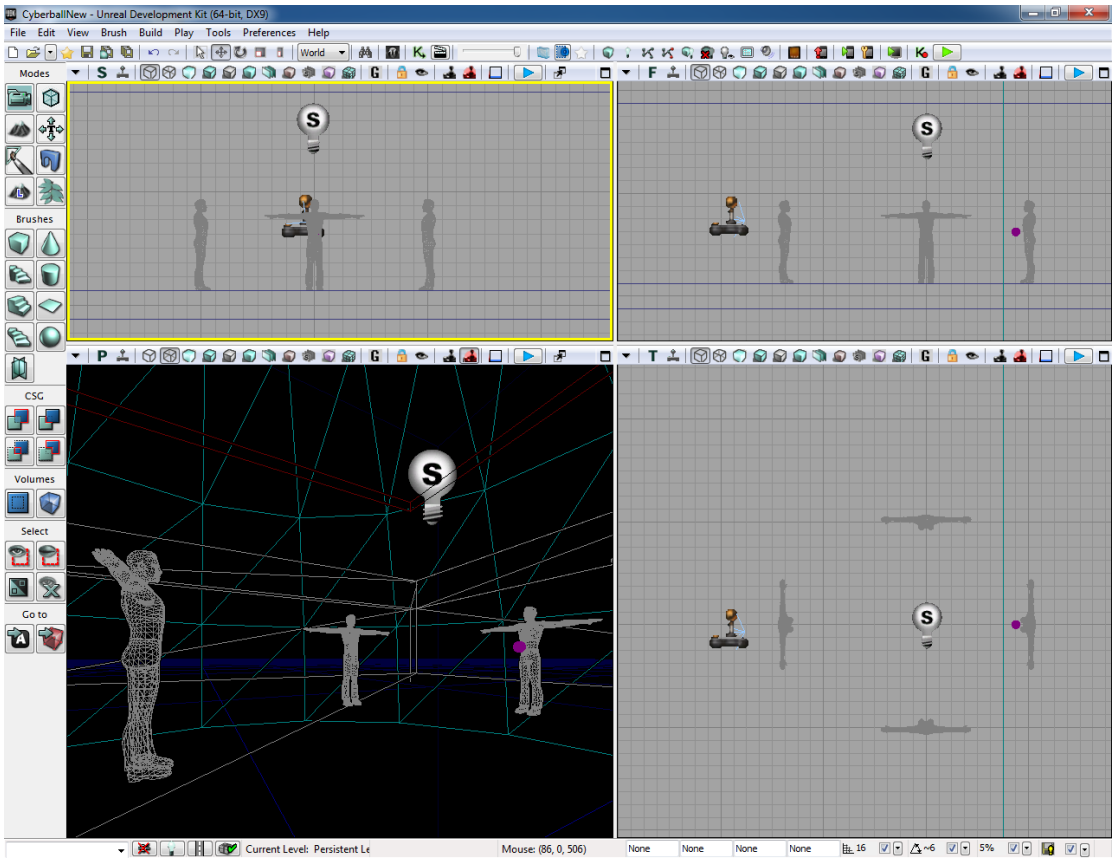


Figure 44: The final scene of the game inside the Unreal Editor.

The synthetic scene of the Cyberball3D+ game consists of 4 players and one ball, inside the Unreal Editor (Figure 44). Particularly, the four players are positioned on the scene so as to form a rhombus, e.g. they are placed on the vertices of a rhombus. Player 1 is placed on the left side, Player 2 on the right and directly opposite of the participant in the fMRI

scanner is Player 3. Three levels of anthropomorphism (low, medium and high) were employed and in each round of the game the appropriate 3D characters of the scene (Figures 45, 46 and 47) are displayed based on the parameters selected by the neuroscientists as entered in the initial menu.

In addition, a *PointLight* was utilized in the center of the scene, simulating an artificial light.

Figure 45: Cyberball3D+ game of low level of anthropomorphism.

Figure 46: Cyberball3D+ game of medium level of anthropomorphism.

Figure 47: Cyberball3D+ game of high level of anthropomorphism.

4.3 Unrealscript Classes

The core of the development of the complete application required for the fMRI experiments was implemented in UnrealScript. Several classes were created in UnrealScript which would handle the aspects of the application's rendering, navigation and interaction with the synthetic scenes and the other 3d models. The most important classes created will be presented here.

CyberballGameInfo: This was the class of the application that defined the main properties, such as the Pawn that would be used in the application and the Controller that would handle the Pawn. This class extended the GameInfo class, as can be seen in its declaration:

```
class CyberballGameInfo extends GameInfo;
```

This class was needed to define the default Pawn and the Controller for the Pawn that would be used in the experiments, when a new 3D scene was loaded. It defines the game being played: the game rules, scoring, which actors are allowed to exist in this game type, and who may enter the game. While this class is the public interface, much of its functionality is delegated to several classes to allow easy modification of specific game components. A *CyberballGameInfo* actor is instantiated when the level is initialized for gameplay (in C++ UGameEngine::LoadMap()). The class of this actor is determined by the DefaultGame entry in the game's .ini file (in the Engine.Engine section), which was set to be

CyberballGameInfo. The GameType used can be overridden in the class's script event SetGameType(), called on the game class picked by the above process.

The experiment flow was controlled from the *Controller* of the *Pawn*, so the code needed in this class was to define the default *Pawn* and the default *Controller* of the *Pawn* as well as the class that was responsible for the User Interfaces, in the default properties block of the class as follows:

```
DefaultProperties
{

        bDelayedStart = false;
        HUDType = class'ScoreHUD';
        DefaultPawnClass = class 'CyberballPawn';
        PlayerControllerClass = class 'CyberballGamePlayerController';

}
```

CyberballPawn: This class defined the *Pawn* that would be used in the application. When a scene was started, a new *CyberballPawn* was instantiated, as instructed from the *CyberballGameInfo* class. It extended from the *Pawn* class and defined the main properties of the used *Pawn*. The class declaration was the following:

```
class CyberballPawn extends Pawn;
```

The only pawn used in the experiments was that belonging to the participant and it was assumed to be a simple camera navigating inside the synthetic scenes. Also, UDK supports the ability for *Pawns* to jump, swim, fly, climb ladders, etc. Such features were not required; therefore, these abilities were disabled in the *default properties* of the *Pawn* class. The settings used for the *Pawn* in its *default properties* block can be seen below:

```
DefaultProperties
{
        bCanBeDamaged=false  //this pawn cannot receive any damage.
        bCanCrouch=false             //this pawn cannot crouch
        bCanFly=false                //this pawn cannot fly
        bCanJump=false               //this pawn cannot jump
        bJumpCapable=false   //this pawn is not capable of jumping
        bCanSwim=false               //this pawn cannot swim
        bCanTeleport=false   //this pawn cannot teleport

}
```

CyberballGamePlayerController: This class was used as the *Controller* of the *CyberballPawn* and took control of the created *Pawn* when a scene was started, as instructed by *CyberballGameInfo*. It was the class that handled all aspects of the application and in which all computations were taking place, such as the interactions with the other players. It extended from *GamePlayerController* and it was bound to *Cyberball* for saving its properties between different rounds of the game and to *TestDLL*, in order to create entries in the log files. The declaration of the class was the following:

```
class CyberballGamePlayerController extends GamePlayerController DLLBind(TestDLL)
config(Cyberball);
```

The *CyberballGamePlayerController* class contained all the functions and code necessary to control the state of the experiment and the events that should occur at specific time points. All of these will be described in different parts in the following subsections. In the *default properties* block of this class, the default *Player Input* class, which is responsible for the translation between button presses from the user and actions inside the virtual scene, was defined, as follows:

```
DefaultProperties
{
        InputClass=class'CyberballPlayerInput';
}
```

4.4 Handling User Input

The core development of the complete application as described above was implemented in UnrealScript. Several classes were created in UnrealScript which handles the aspects of the application's interaction with the synthetic scenes. One of the main requirements of the application was that the synthetic scenes were to be interactively manipulated and that the application was required to react to user input throwing the ball to the players. In order to achieve this, the buttons corresponding to the physical button boxes placed in the fMRI scanner and utilized for interacting with the scene were registered with their respective commands. A Current Designs 932 set up was utilized (Figure 59). Two buttons from a four-button interface are used to throw the ball left or right respectively. The

remaining two buttons of the 4-button-boxes are not used. The two main methods that handled the participant's actions were ThrowLeft and ThrowRight. When the participant pressed a button to throw the ball left or right the Controller class executed the ThrowLeft or ThrowRight method, respectively. These methods check first if the participant has the ball and then act to throw the ball to either Player 1 or 2. In addition, a more action was added to the configuration file that was responsible for the termination of the current round displaying the main menu when the button escape of the keyboard is pressed by the neuroscientists.

So, the following entries were added in the "DefaultInput.ini" configuration file, which is responsible of defining the key bindings for the *Input Manager* to handle:

```
.Bindings=(Name="one",Command="GBA_ThrowLeft")

.Bindings=(Name="four",Command="GBA_ThrowRight")

.Bindings=(Name="Escape",Command="GBA_EscapeFunction")
```

Then, in the same configuration file, the commands bound to the buttons were further assigned to a specific method that would handle them in the *CyberballGamePlayerController* class:

```
.Bindings=(Name="GBA_ThrowLeft",Command="ThrowLeft");

.Bindings=(Name="GBA_ThrowRight",Command="ThrowRight");

.Bindings=(Name="GBA_EscapeFunction",Command="EscapeFunction");
```

Each one of the methods *ThrowLeft*, *ThrowRight* and *EscapeFunction* assigned to each button resides in the *CyberballGamePlayerController* class and handles the specific action that it is assigned to.

```
exec function ThrowLeft(){
        if(currentPosition == 0 && ballIsAttached == 1){
                Values(0,1,'ThrowBallLeft','CatchBallRight',5);
                SetTimer(0.7,false,'ThrowBall');
        }
}

exec function ThrowRight(){
        if(currentPosition == 0 && ballIsAttached == 1){
                Values(0,2,'ThrowBallRight','CatchBallLeft',4);
```

```
                SetTimer(0.7,false,'ThrowBall');
        }
}

exec function EscapeFunction(){

        consolecommand("open CyberballNew");
}
```

The functions *ThrowLeft* and *ThrowRight* check first whether the participant had the ball and then acted to throw the ball to Player 1 and 2 activating the animations 'ThrowBallLeft'/'CatchBallRight' and 'ThrowBallRight'/'CatchBallLeft, respectively. In addition, there was a timer that counted the time that had to be passed before the function 'ThrowBall' was executed. Furthermore, the function *EscapeFunction* was responsible to send a console command to the Game Controller to terminate the current round and open the CyberballNew scene again.

4.5 Logging of Player's Actions

The application is recording every action by the players in a separate log file dedicated to each experimental round of the game in order to be able to understand and analyze the data gathered from each experiment. For each participant, a different log file is created for each round which follows the following naming convention: log(Participant Number)_r(Number of Round).csv. Each ball tossing occurring by all players is being recorded in the log file. The first data column records the time stamp when a player throws a ball, the second column which player throws the ball and the third column which player catches the ball. The time was measured in milliseconds starting just at the start of the game of the experiment assumed to be time point 0.

In order to implement the log file operations, a .dll file was bound to the Controller class, providing the necessary methods to record each log entry. This was implemented because UDK's support for I/O operations is limited avoiding extreme overhead for the application since UnrealScript is very slow and inefficient for such operations. Whenever a new log entry was recorded in the log file, the controller could simply call the C/C++ function residing in the .dll file and let it perform the operation.

In order also, to make a function residing in the .dll visible in an UnrealScript class, it had to be declared in that class with a *dllimport* and a *final* modifier. The functions that were in the .dll file and were included in the controller class were the following:

```
dllimport final function initStartTime();

dllimport final function int ReturnCurrentTime();

dllimport final function logWriteEvent(int pn, int round, int ThisPlayer, int ToPlayer, string msg);

dllimport final function logWriteSyncPulse(int pn, int round, int ThisPlayer, int ToPlayer, string msg);

dllimport final function logHeaders(int pn, int round, string Lev,  string levelAnt, string GeA, string genderA, string GeB, string genderB, string GeC, string genderC, string GeD, string genderD);

dllimport final function logWriteTossesAndScores(int pn, int round, int tossesFromPlayerA, int tossesFromPlayerB, int tossesFromPlayerC, int tossesFromPlayerD, int scoreFromPlayerA, int scoreFromPlayerB, int scoreFromPlayerC, int scoreFromPlayerD);

dllimport final function logWriteEventScorePlayer(int pn, int round, int CurrentPlayer, int score);

dllimport final function SaveFiledll(string saveThefile, string text);

dllimport final function string LoadFiledll(string loadThefile);

dllimport final function string LoadAllFilesdll();
```

The function *startTime* was executed when the game was started while the function ReturnCurrentTime was executed when a new round of the game was started. The .dll file had two global variables named *startTime* and *Timenow* respectively, which were set to be of *struct timeb* type, located in <sys/timeb.h>. So, the functions were designed to be the following:

```
__declspec(dllexport) void initStartTime()
{
	ftime(&startTime);
	FILE* fp = 0;

	fopen_s(&fp, "startTime.jc", "w");

	fprintf(fp, "%ld\n", startTime.time);
```

```
	fprintf(fp, "%d", startTime.millitm);

	fclose(fp);
}

__declspec(dllexport) int ReturnCurrentTime()
{
	int currenttime;
	ftime(&Timenow);

	currenttime = Timenow.time; //Seconds of current time

	return(currenttime);

}
```

```
__declspec(dllexport) void logWriteEvent(int pn, int round, int ThisPlayer, int ToPlayer,
wchar_t* msg){

	double n = Timenow.time*1000 + Timenow.millitm;
	double s = startTime.time*1000 + startTime.millitm;

	double diff = n - s;

	FILE* fp = 0;
	char filename[100] = "";

	int rd = round+1;

	sprintf_s(filename, "%s_%d_r%d%s", "log", pn, rd, ".csv");

	fopen_s(&fp, filename, "a");
	if(fp == NULL)
		return;

	fprintf(fp, "%.0lf,", diff);

	fprintf(fp, "%d,%d,",ThisPlayer, ToPlayer);

	const size_t newsize = 500;
	size_t convertedChars = 0;

	size_t origsize = wcslen(msg) + 1;
	char message[newsize];
	wcstombs_s(&convertedChars, message, origsize, msg, _TRUNCATE);
	fprintf(fp, "%s\n", message);

	fclose(fp);
}
```

```
__declspec(dllexport) void logWriteSyncPulse(int pn, int round, int ThisPlayer, int ToPlayer,
wchar_t* msg){

	double n = Timenow.time*1000 + Timenow.millitm;
	double s = startTime.time*1000 + startTime.millitm;

	double diff = n - s;

	FILE* fp = 0;
	char filename[100] = "";

	int rd = round+1;

	sprintf_s(filename, "%s_%d_r%d%s", "log", pn, rd, ".csv");

	fopen_s(&fp, filename, "a");
	if(fp == NULL)
		return;

	fprintf(fp, "%.0lf,Sync Pulse,", diff);

	fprintf(fp, "%d,%d,",ThisPlayer, ToPlayer);

	const size_t newsize = 500;
	size_t convertedChars = 0;

	size_t origsize = wcslen(msg) + 1;
	char message[newsize];
	wcstombs_s(&convertedChars, message, origsize, msg, _TRUNCATE);
	fprintf(fp, "%s\n", message);

	fclose(fp);
}
```

When a ball tossing occurs by a player the functions *logWriteEvent* and *logWriteSyncPulse* were executed in order to record the data in the log files. As described above, the first data column records the time stamp when a player throws a ball, the second column which player throws the ball and the third column which player catches the ball. The time was measured in milliseconds starting just at the start of the game of the experiment assumed to be time point 0.

In addition, the function *LogHeaders* was called at the starting of each round of the game recording some parameters of the round such as the level of the anthropomorphism of all players as well as the gender of each player. While the functions

logWriteTossesAndScores and *logWriteEventScorePlayer* were used in the ReInforcement Learning version of the game to record the scores of each player.

Finally, the functions *SaveFiledll, LoadFiledll* and *LoadAllFilesdll* were used for loading and saving the parameters of each round of the game and will be described in the fifth section.

4.6 Time Limits Control

The game consists of a number of rounds and each round is played at preordained time defined by the neuroscientists in the initial menu. The application was timed perfectly in order to be synchronized with the brain images acquired by the scanner.

The Controller class was developed in order to control the time limits and react so as not to exceed them. At the end of each ball tossing, the Controller class calls a C/C++ function residing in the .dll file to check if the time limit was exceeded or calculate the remaining time. The implementation code that checks the time limit is described below:

```
curTime = ReturnCurrentTime();
timenow = curTime - initTime ;
newversiontime = timenow + 3;
remaintime = overallnum - timenow;

if(newversiontime < overallnum){
        SetTimer(0.8,false,'ThrowBall');
}
else{
    CurrentRound++;
        if(CurrentRound==CountOfRounds){
                returnMenuForScores = 1;
                SetTimer(remaintime,false,'Finished');
        }
        else{

        SetTimer(remaintime,false,'NextRound');
        SetTimer(remaintime,false,'SetBlackScreen');
                InTheNextRound = true;
                rmtime = remaintime + 5.5;
                SetTimer(rmtime,false,'GameBegin');
        }

}
```

As we can see, at the end of each ball tossing, the Controller class calls the ReturnCurrentTime function to get the current time, and then calculates the time that is passed from the starting of the current round and finally calculates the time that remains until the end of the round. If there is enough time for the next ball tossing then the current round is continuing to be played else the Controller class checks if there is other round that has to be played by loading it. If all rounds of the games were played, then the game will be finished.

4.7 Time Synchronization with fMRI Scanner

The experiments were conducted inside an fMRI scanner and the application was required to be perfectly synchronized with the scanner, in order to be able at a later stage of the game to identify the exact action of the application associated with each brain image acquired by the scanner. The application was instructed to send a specific sound sync pulse to be recorded from the PC that was dedicated to recording the spike signals sent from the synchronization box of the scanner, as well as the heart rate data and pupillometry of the participant. The sound sync pulses were directed through the left audio channel of the application, leaving the right audio channel available to the participant.

The required synchronization between the fMRI scanner and the application was achieved by sending such sound sync pulses to be recorded as an analog spike signal, whenever a ball tossing occurred in the application while at the same time recording that action and the exact time it happened in the log file. The log file describes the state of the application associated to a specific brain image taking into account the sound sync pulse sent last before receiving that image.

For each different ball tossing based on the direction of the movement of the ball was sending different sound sync pulse through the left audio channel of the application, in order to be easy distinguishable by the neuroscientists which player throws the ball and which player catches the ball. When a ball tossing occurred, the Controller class sent a sync pulse through the left audio channel in order to be recorded by the spike application and then recorded the sync pulse in the log file, as described below:

```
ClientPlaySound(AllPlayersData[currentplayer].AnimsData[ToPlayerCount].sound);

logWriteSyncPulse(ParticipantNumber, CurrentRound, currentplayer, nextplayer, "");
```

In addition, at the start of each round of the game the *Controller* class also sent sound sync pulses through the both audio channels, in order to be recorded by the spike application and inform the participant that will be started a new round of the game, as we can see on the code below:

```
ClientPlaySound(ParticipantSound);
ClientPlaySound(RoundSound);
```

4.8 Summary

This chapter describes in detail the implementation of this interactive 3D gaming framework. Specifically, examples and source code samples were demonstrated, in relation to analyze how the application met the requirements imposed by the expert psychiatrists in order to incorporate a formal neuroscientific protocol for the fMRI scanner.

5 Chapter 5 – UI Implementation

Although UDK includes preliminary support for User Interfaces (UI), the ability to embed Flash User Interfaces was appropriate in relation to the requirements of the experiments, since the UIs were required to be interactive and to be displayed transparently on top of the rendered scene. These requirements could only be fulfilled with the use of embedded Flash UI applications. The Flash application is imported as .SWF file in UDK and loaded and displayed in a scene by connecting the Open GFx Movie action to the Level Loaded and Visible event, which is automatically generated and activated by UDK, when the virtual scene becomes visible. The Open GFx Movie action is provided by UDK and it accepts an imported Flash UI as an argument, which it loads and displays it on the center of a screen or on a specified surface.

Each Flash UI was designed so as to be displayed on top of the currently rendered virtual scene as shown in Figure 48. The participant's ability to navigate and look around the virtual scene was disabled every time a Flash UI was displayed, so that the input could be captured by the Flash UI itself.

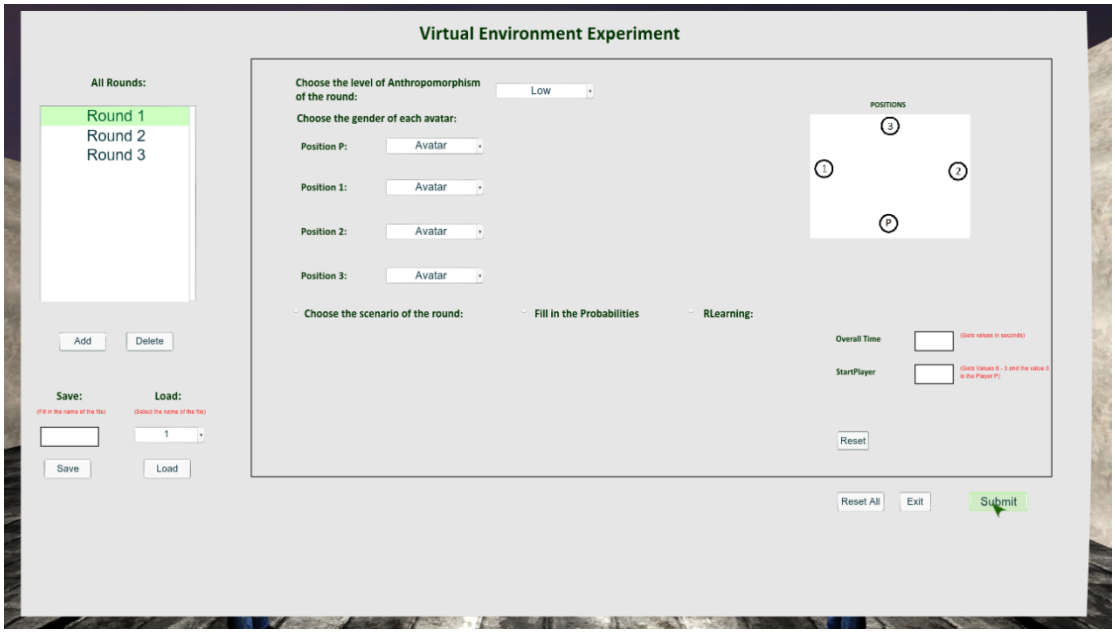


Figure 48: A Flash UI menu being displayed on top of the currently rendered virtual scene.

5.1 Application Menus as Flash UIs

For the purposes of the Cyberball3D++ game we implemented the User Interfaces of the game such as the "initial" and "return" menu as well as the interface that depicted the scores of the players in the **Reinforcement Learning** version of the game using the Adobe Flash Professional.

5.1.1 Initial Menu

The "initial" menu of the game (Figures 49, 50, 51, 52, 53 and 54) was used by neuroscientists to define the parameters of the game. It was displayed on the top of the screen when the game was initiated and the neuroscientists had the ability to select the level of anthropomorphism of all avatars, the gender of each avatar and the scenario of each round as well as define which player initially would have the ball at the start of each round. In addition, they filled in the duration limit for each round.

Analytically, the neuroscientists had the ability to select one of the three versions of the game. The three versions were the **five basic scenarios**, the **probabilities** and the **Reinforcement Learning** version. Particularly, in the version of the **five basic scenarios** the neuroscientists had the ability to select either the first scenario in which no player is excluded or the second and third scenario in which programmed Players 1 or 2 (left or right player) exclude the participant and Player 3 (player opposite the participant) from the game, respectively. Finally, they were able to select the fourth or fifth scenario in which either the Player 1 or 2 exclude the Player 3 and the participant from the game, respectively.

In the **probabilities** version of the game the neuroscientists were able to fill in the probability of the tosses that each player made to each one of the other players. For example the neuroscientists could set the probabilities that Player 1 would pass the ball to Player 2, 3 and the participant independently of one another, the probabilities that Player 2 would pass the ball to Player 1, 3 and the participant, as well as the probabilities that Player 3 would pass the ball to Player 1 and 2.

Finally, in the **Reinforcement Learning** version of the game the neuroscientists were able only to select the checkbox for this version.

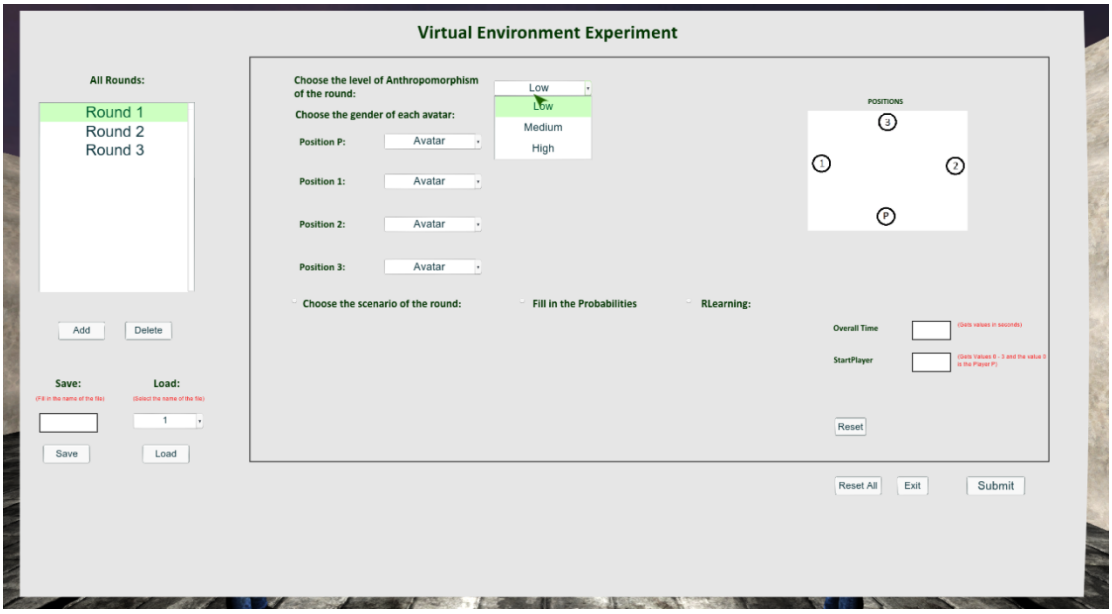


Figure 49: The selection of the level of anthropomorphism of all avatars.

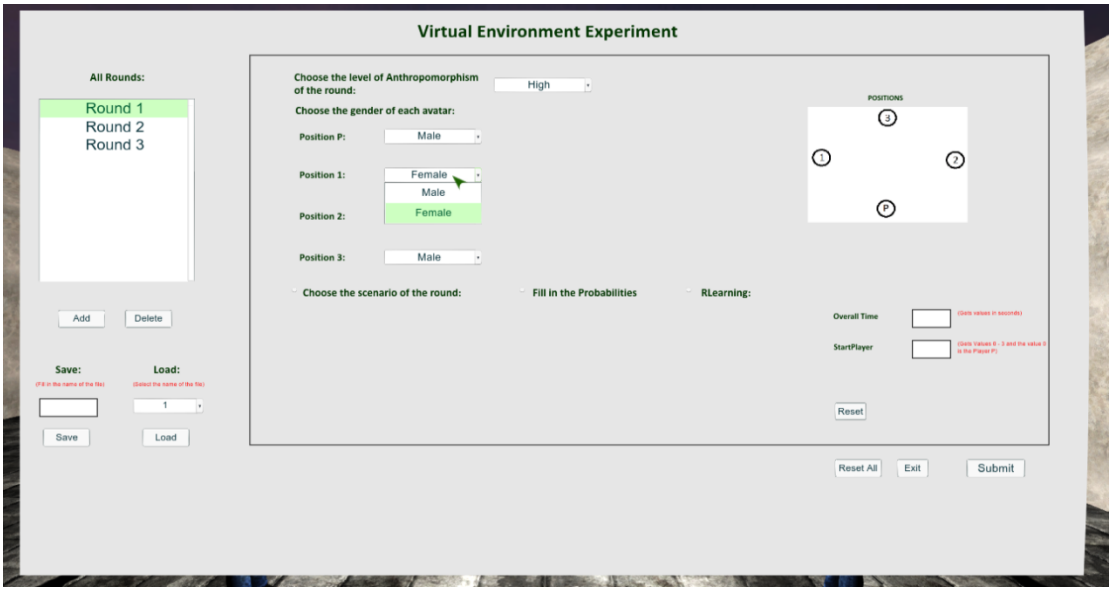
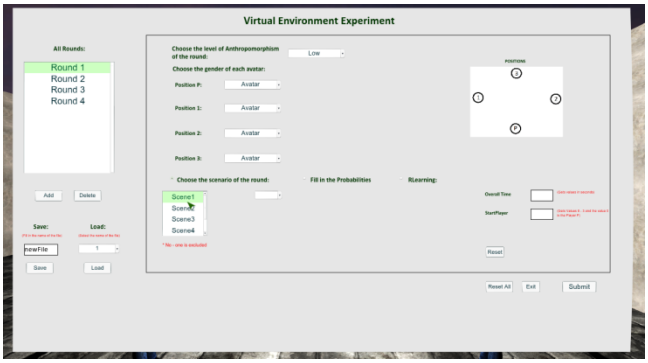


Figure 50: The selection of the gender of the player 1.



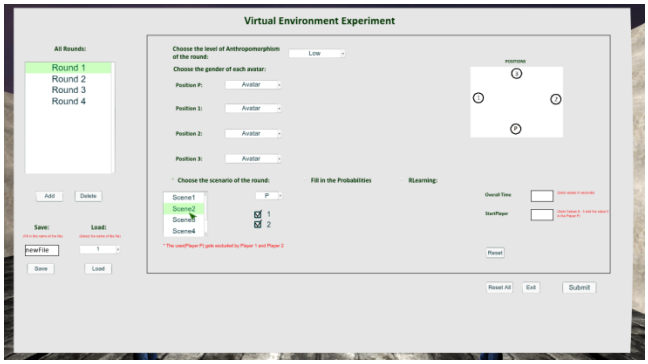


Figure 51: Two examples show the selection of the scenario from the five basic scenarios version of the game.

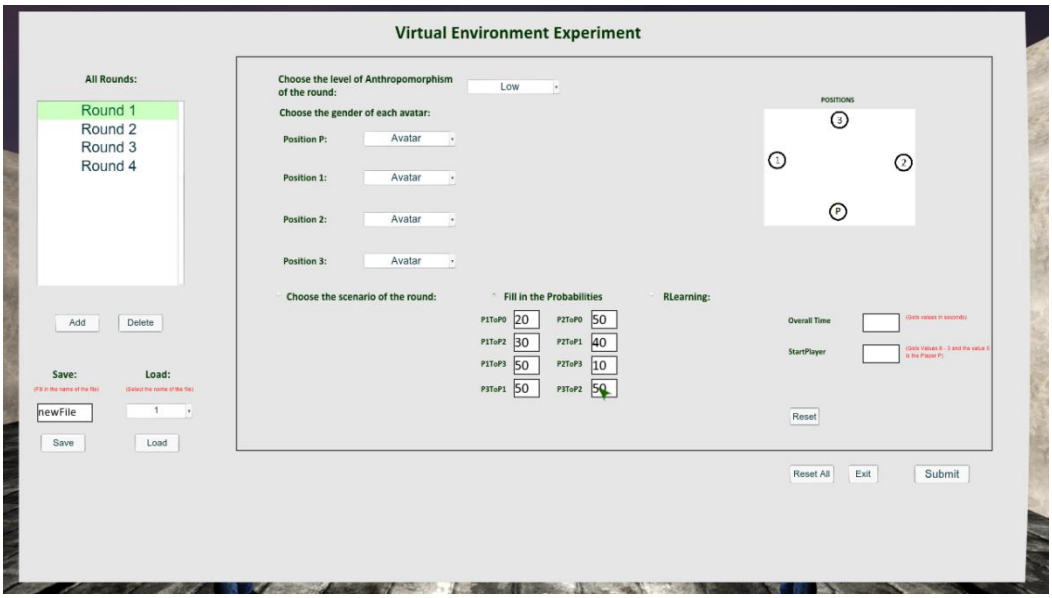


Figure 52: The filling of the probabilities in the second version of the game.

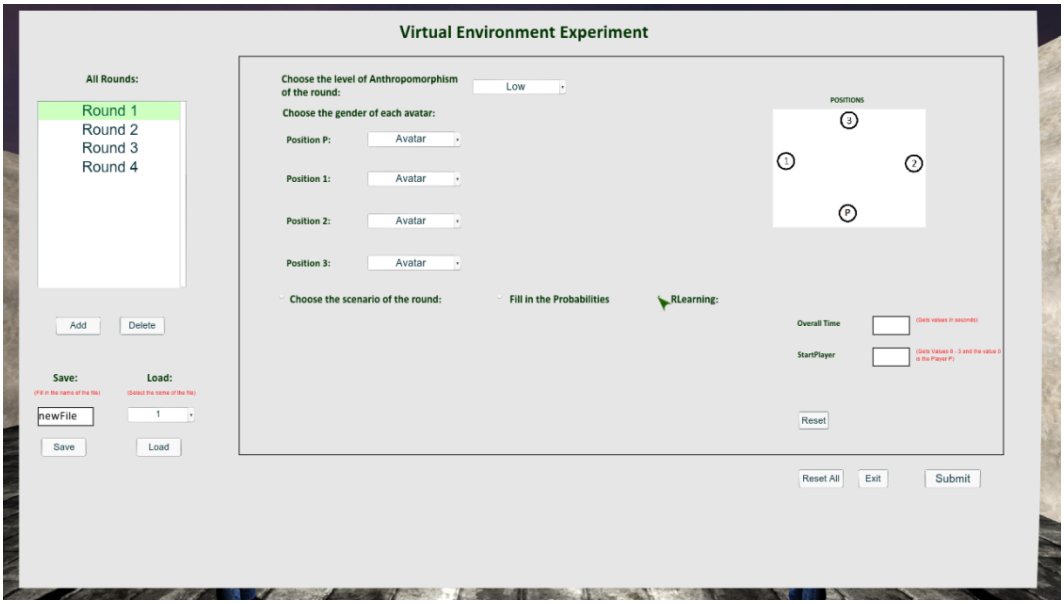


Figure 53: the selection of the third version of the game.

Figure 54: The left screenshot depicts the filling of two parameters of the game, while the right screenshot displays an error message.

5.1.2 Return Menu

The "return" menu was displayed on the top of the screen at the end of the game depicting a question to the neuroscientists if they want to continue the game or finish it.

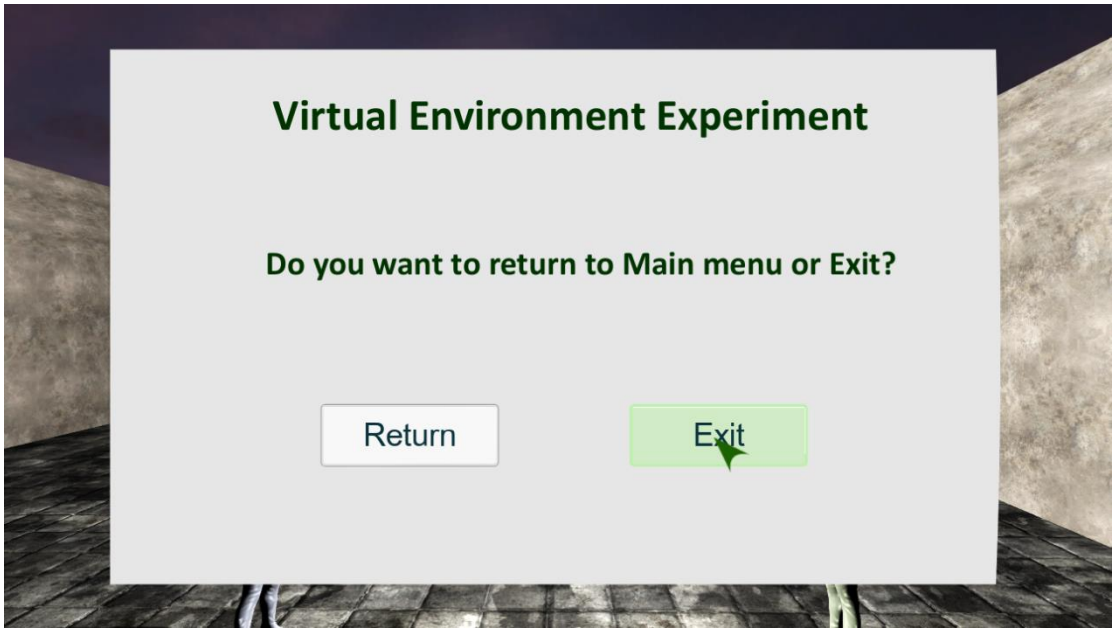


Figure 55: The return menu that was depicted on the screen at the end of the game.

5.1.3 Score Menu

The "score" menu was displayed on the upper left of the screen over the whole duration of the game in the **Reinforcement Learning** version of the game and depicted the scores of the players.

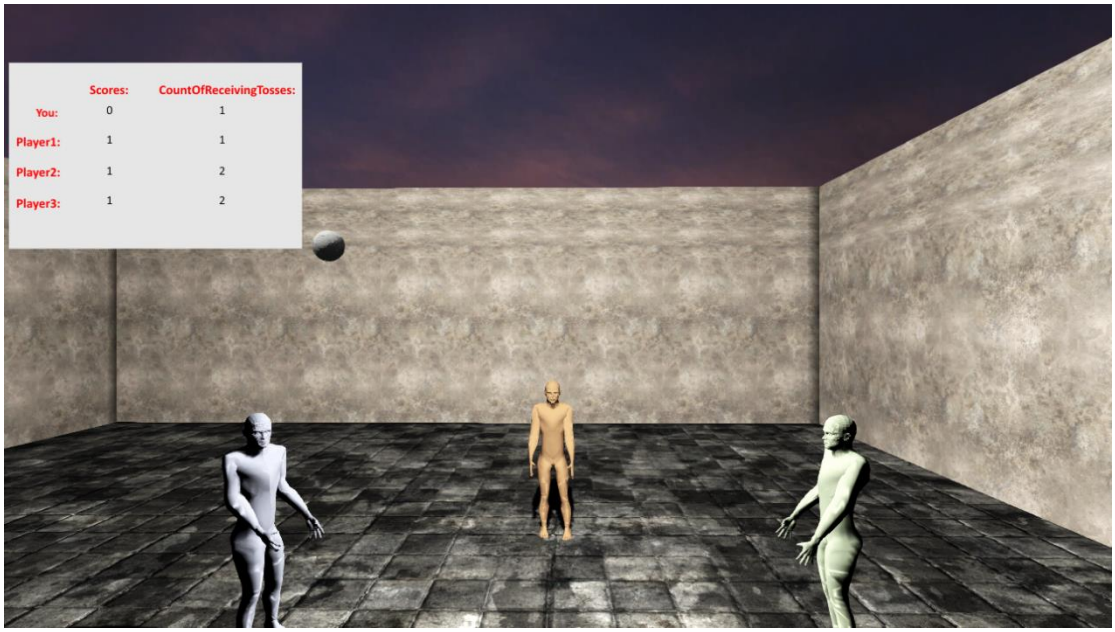


Figure 56: A screenshot shows the scores of the players during the game.

5.2 DLL Files for saving and loading UIs Parameters

Additionally, the neuroscientists using the "initial" menu had the ability to save the rounds and the parameters of each round in a file through a C/C++ function residing in a .dll file which is called by the Controller class as well as load them by selecting the name of the file. The functions those were responsible for the saving and loading of the parameters of the game were *SaveFiledll*, *LoadFiledll* and *LoadAllFilesdll*. As also described above, in order to make a function residing in the .dll visible in an UnrealScript class, it had to be declared in that class with a *dllimport* and a *final* modifier. So, these functions that were in the .dll file were included in the controller as we can see below:

```
dllimport final function SaveFiledll(string saveThefile, string text);
dllimport final function string LoadFiledll(string loadThefile);
dllimport final function string LoadAllFilesdll();
```

The parameters of the game such as the level of the anthropomorphism of all avatars, the gender of each avatar, the scenario and the definition of the starter player as well as the duration limit of each round are saved and loaded as a string joining them using the semicolon (‘,’) delimiter. The function *SaveFiledll* saves the string that contains the parameters of the game in a file. The string with the parameters of the game and the name of the file are passed as parameters in the function. In the same way, the functions

LoadFiledll and LoadAllFilesdll load the parameters of the game and the saved files returning the string with the parameters of the game and the string with the names of all saved files, respectively. So, the functions that were in the .dll file were designed to be the following:

```
__declspec(dllexport) void SaveFiledll(wchar_t* saveThefile, wchar_t* text)
{
	FILE* fp = 0;
	FILE* fpa = 0;

	// the name of the file
	size_t origsize = wcslen(saveThefile) + 1;
	const size_t newsize = 1000;
	size_t convertedChars = 0;
	char savefile[newsize];
	wcstombs_s(&convertedChars, savefile, origsize, saveThefile, _TRUNCATE);


	fopen_s(&fp, savefile, "w");
	if(fp == NULL)
		return;

	//the string that will be saved in the file
	origsize = wcslen(text) + 1;
	convertedChars = 0;
	char txt[newsize];
	wcstombs_s(&convertedChars, txt, origsize, text, _TRUNCATE);

	fprintf(fp, "%s", txt);

	fclose(fp);

	origsize = wcslen(saveThefile) + 1;
	convertedChars = 0;
    char savefile2[newsize];
   wcstombs_s(&convertedChars, savefile2, origsize, saveThefile, _TRUNCATE);

	//Save the name of the file in the allfiles file
	fopen_s(&fpa, "allfiles", "a");
	if(fpa == NULL)
		return;

	fprintf(fpa, "%s", savefile2);
	fprintf(fpa, "%s", ",");

	fclose(fpa);
}

__declspec(dllexport) wchar_t* LoadFiledll(wchar_t* loadThefile)
{
```

```
	FILE* fp = 0;

	const size_t newsize = 1000;
	char txt[newsize];

	// the name of the file
	size_t origsize = wcslen(loadThefile) + 1;
	size_t convertedChars = 0;
	char loadfile[newsize];
	wcstombs_s(&convertedChars, loadfile, origsize, loadThefile, _TRUNCATE);

	const size_t cSize = 1000;
	wchar_t wc[cSize];


	fopen_s(&fp, loadfile, "r");
	if(fp == NULL)
		return wc;

	fgets(txt,1000,fp);

	fclose(fp);

	mbstowcs (wc, txt, cSize);

	return wc;
}

__declspec(dllexport) wchar_t* LoadAllFilesdll()
{
	FILE* fp = 0;

	const size_t newsize = 1000;
	char txt[newsize];

	const size_t cSize = 1000;
	wchar_t wc[cSize];

	fopen_s(&fp, "allfiles", "r");
	if(fp == NULL)
		return wc;

	fgets(txt,1000,fp);

	fclose(fp);

	mbstowcs (wc, txt, cSize);

	return wc;
}
```

The Figures 57 and 58 depict two examples of the loading and saving of the parameters of the game. In the example of the loading of the parameters of the game the neuroscientists are able to select the file that they want by using the combobox that contains all the files that have been saved and press the 'load' button. In addition, in the example of the saving of the parameters the neuroscientists have the ability to typing the name of the file as they want and save it using the 'save' button.

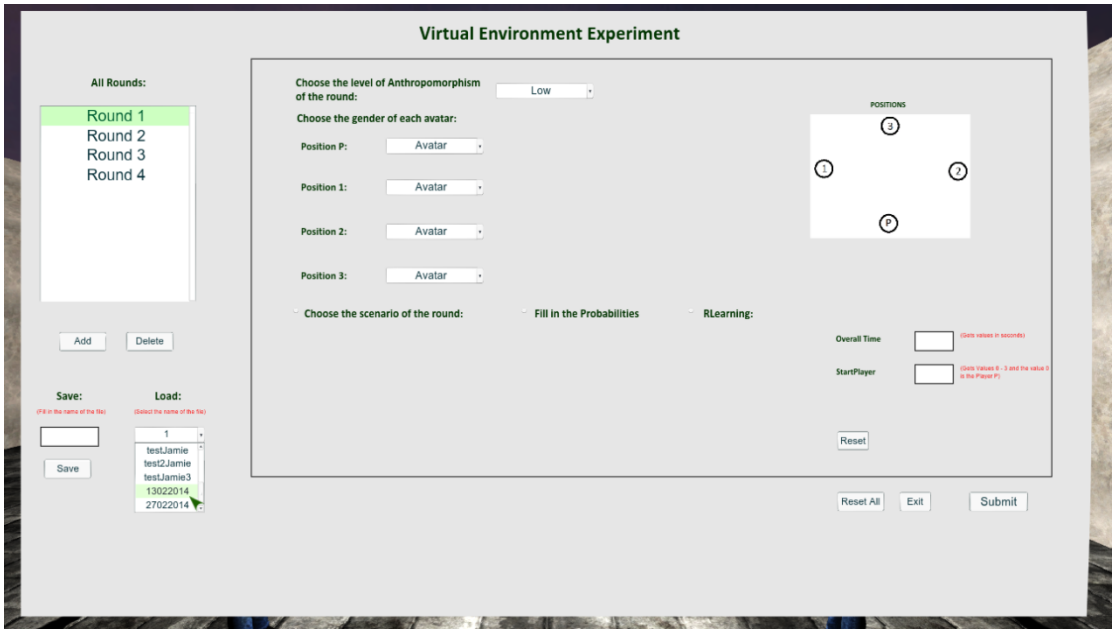


Figure 57: The loading of the parameters of the game by selecting the name of a file.

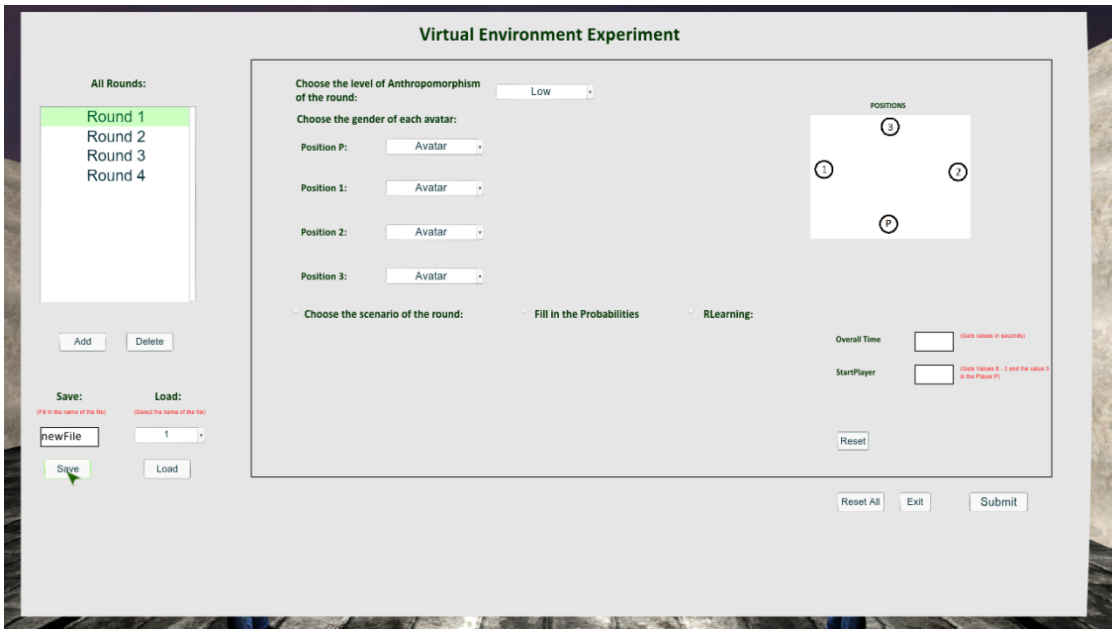


Figure 58: The saving of the parameters of the game by pressing the 'save' button.

5.3 Creating UIs using both of Unrealscript and Actionscript

As mentioned in the Section 3, the integration of a Flash application inside a scene in UDK requires that it should first be compiled into an SWF file and then imported inside the UDK asset library. Afterwards, either UnrealScript or Unreal Kismet can initiate the Flash application, interact with it, hide it or instruct it to stop playing.

While a Flash application is playing inside a scene, UnrealScript can initiate a call of an ActionScript function and vice versa. This feature allows full interaction between the Flash interface and the application. Consequently, it is easy and efficient to create an application that initiates a Flash interface whenever it is required and then receive the user's response and order it to stop playing.

The most User Interfaces that were used in this project such as the 'initial' menu and the 'return' menu were initiated through the Unreal Kismet; however the User Interface that depicted the scores of the players in the **Reinforcement Learning** version of the game was initiated through UnrealScript. Nevertheless, all User Interfaces that were used in this project could initiate a call of an UnrealScript function and vice versa.

In particular, the pressing of the 'Submit', 'Load' and 'Save' button in the 'initial' menu initiated the call of the *StartGame*, *LoadFile* and *SaveFile* UnrealScript function, respectively, passing the necessary parameters. The initiation of the call of the UnrealScript functions inside Actionscript can be seen below:

```
ExternalInterface.call("StartGame",countofrounds,overallNumber,startPlayer,Scenario,ExcPlayer,countOfExc,ExcPlayerByOne,ExcPlayerByTwo,levelAnthrop,avatarP,avatar1,avatar2,avatar3,Modes,Probabilities0,Probabilities1,Probabilities2,Probabilities3,Probabilities4,Probabilities5,Probabilities6,Probabilities7);

ExternalInterface.call("SaveFile",SaveFile.text,countofrounds,overallNumber,startPlayer,Scenario,ExcPlayer,countOfExc,ExcPlayerByOne,ExcPlayerByTwo,levelAnthrop,avatarP,avatar1,avatar2,avatar3,Modes,Probabilities0,Probabilities1,Probabilities2,Probabilities3,Probabilities4,Probabilities5,Probabilities6,Probabilities7);

ExternalInterface.call("LoadAllFile");

ExternalInterface.call("LoadFile",CurrentFile);
```

In the **Reinforcement Learning** version of the game the User Interface that depicted the scores of the players was initiated through the *CyberballGameInfo* class of UnrealScript

declaring the HUDType object to be an instance of the ScoreHUD class. In addition the *HudMovie* object of the *ScoreHUD* was declared to be an instance of the *ScoreMovie* class as well as the *ScoreMovie* class set to its MovieInfo object to be a .swf movie that was created using Adobe Flash Professional. As we can see below, the *TickHUD* function of the *ScoreMovie* class was used to get ticks in real time in order to update the interface in real time as well via *PostRender* function of the *ScoreHUD* class.

The *TickHUD* function in every tick updated the variables scorA, scorB, scorC, scorD, tosA, tosB, tosC, and tosA as well as set the visibility of the interface. These variables were used to depict the scores and the overall number of receiving tosses of each player. In addition, the interface was visible only in the **Reinforcement Learning** version of the game so in every tick the *TickHUD* function checks if the game was in this mode in order to change the visibility of this to true. So, the functions that were in the Unrealscript classes were designed to be the following:

```
class CyberballGameInfo extends GameInfo;

DefaultProperties
{
        HUDType = class'ScoreHUD';

}
```

```
class ScoreHUD extends HUD;

var ScoreMovie HudMovie;

simulated event PostBeginPlay()
{
        // Grab all the normal initialization for the HUD class.
         super.PostBeginPlay();

         //Create a new instance of our custom GFX HUD class.
         HudMovie = new class'ScoreMovie';

         //Set it to realtime updating.
         HudMovie.SetTimingMode(TM_Real);

         //Calls an initialization function inside the custom GFX HUD class
         HudMovie.Init();

}
```

```
event PostRender()
{
	//Call all the other PostRender stuff from GFxMovie
	super.PostRender();

    // As long as we have a HUD, we call the TickHUD function on every tick.
	if (HudMovie != none){
		HudMovie.TickHUD();
	}

}
```

```
class ScoreMovie extends GFxMoviePlayer;

DefaultProperties
{
	MovieInfo=SwfMovie'Cyberball.Scores';
}

function Init( optional LocalPlayer LocPlay )
{
 //Gets all the other intialization stuff we need.
 super.Init (LocPlay);

 //Starts the GFx Movie that's attached to this script (IE: our HUD).
 Start();

 //Advances the frame to the first one.
 Advance(0.f);
 SetViewport(0,0,350,350);
}
```

```
function TickHUD(){
	local WorldInfo myworldinfo;
	local CyberballGameInfo cyberinfo;

	local String scorA, scorB, scorC, scorD;
	local String tosA, tosB, tosC, tosD;

	myworldinfo = class 'WorldInfo'.static.GetWorldInfo();
	cyberinfo=CyberballGameInfo(myworldinfo.Game);

	SetModeVis();

	scorA = cyberinfo.MyController.scs[0];
	scorB = cyberinfo.MyController.scs[1];
	scorC = cyberinfo.MyController.scs[2];
	scorD = cyberinfo.MyController.scs[3];

	tosA = cyberinfo.MyController.toss[0];
```

```
        tosB = cyberinfo.MyController.toss[1];
        tosC = cyberinfo.MyController.toss[2];
        tosD = cyberinfo.MyController.toss[3];

         SetVariableString ("_root.ScoreYou.text",scorA);
         SetVariableString ("_root.Score1.text",scorB);
         SetVariableString ("_root.Score2.text",scorC);
         SetVariableString ("_root.Score3.text",scorD);

         SetVariableString ("_root.TossesYou.text",tosA);
         SetVariableString ("_root.Tosses1.text",tosB);
         SetVariableString ("_root.Tosses2.text",tosC);
         SetVariableString ("_root.Tosses3.text",tosD);

}
```

```
function SetModeVis(){
        local WorldInfo myworldinfo;
        local CyberballGameInfo cyberinfo;

        myworldinfo = class 'WorldInfo'.static.GetWorldInfo();
        cyberinfo=CyberballGameInfo(myworldinfo.Game);

         if(cyberinfo.MyController.Mode == "2"){
                MyActionScriptVoidTrue();
         }

         else if(cyberinfo.MyController.Mode != "2"){
                MyActionScriptVoidFalse();
         }

         if(cyberinfo.MyController.returnMenuForScores == 1){
                MyActionScriptVoidFalse();
         }
}
```

```
function MyActionScriptVoidTrue(){
        ActionScriptVoid("_root.SetVisibleTrue");
}

function MyActionScriptVoidFalse(){
        ActionScriptVoid("_root.SetVisibleFalse");
}
```

Furthermore, from the side of ActionScript were used the functions SetVariableString and ActionScriptVoid to set values to the TextBoxes objects of the interface as well as the visibility of its.

5.4 Summary

This chapter describes the implementation of the User Interfaces (UI) that we used in this interactive 3D gaming framework. The steps taken to create the individual UIs as Flash applications and embed them in the complete systems were presented as well. In addition, the challenges concerning the creation of interactive synthetic worlds and associated UIs as displayed in the fMRI scanner were described as well as the solutions to overcome them were explained.

6 Chapter 6 – Experiments

6.1 Materials

This experiment was designed to explore the changes in regional brain activity associated with the pain of social exclusion as well as with feelings such as the empathy felt when people observe other people get socially excluded. From a computer graphics point of view this experiment investigated whether the level of anthropomorphism of the avatars may affect game playing as well as fMRI data acquired at the same time the game playing occurs. The goal was to discover the neural circuitry that supports such feelings as well as, for the first time, devised behavioral fidelity metrics of character believability and emotional engagement based on neural activity. Two groups of experiments were conducted, using the 'Five Basic Scenes' version of the Cyberball in the first group and a combination of the 'Five Basic Scenes' and the Reinforcement learning versions in the second group.

6.1.1 Participants

10 healthy adult volunteers (eight female, two male mean age 41.5 years, range 29-65) underwent functional echo planar imaging at Brighton and Sussex Medical School, Sussex, United Kingdom in the first group of the experiments as well as 12 healthy adult volunteers separated into two groups of six persons (nine male, three female mean age 29.5 years, range 25-34) participated in the second stage of the experiments at Technical University of Crete.

6.1.2 Apparatus

In the first stage of the experiments the VEs were presented at VGA resolution on the screen of an fMRI, with a Field-of-View comprising 50 degrees diagonal. A Current Designs 932 (Figure 59) set up was utilized for interacting with the scene. Two buttons from a four-button interface were used by participants to throw the ball left or right respectively. The remaining two buttons of the 4-button-boxes were not used. The viewpoint was set in the middle and front of the synthetic scene. Rotation and navigation around the scene were disabled so that participants would not interact with the other virtual avatars only through the playing of the game. The application ran on a standard PC connected to the screen of the fMRI as Clone-Mode. The experiments were conducted inside an fMRI, which recorded the

brain images data. Another standard PC was used to record the spikes sent from the fMRI's synchronization box, as well as pulse oximetry and sound sync pulses from the application.

In the second stage of the experiments the VEs were presented at a standard PC of the Laboratory of Distributed Multimedia Information Systems and Applications of the Technical University of Crete. The arrows buttons of the computer keyboard were used by participants to throw the ball to the virtual players. As the first stage of the experiments the viewpoint of the VE was set in the middle and front of the synthetic scene as well as rotation and navigation around the scene were disabled so that participants would not interact with the other virtual avatars only through the playing of the game.

Figure 59: Photo of the Current Designs 932 response pad used in the experiments.

6.1.3 Visual Content

The synthetic scenes of the Cyberball3D+ game consisted of 4 players and one ball. The four players were positioned on the scene so as to form a rhombus, e.g. they were placed on the vertices of a rhombus. Player 1 was placed on the left side, Player 2 on the right and directly opposite of the participant was Player 3. Three levels of anthropomorphism (low, medium and high) were employed and in each round of the game the appropriate 3D characters of the scene were displayed based on the parameters selected by the neuroscientists as entered in the initial menu. In addition, three versions of the game were implemented simulating social exclusion or empathy situations in all level of anthropomorphism.

The 3D characters of the 'low level' of anthropomorphism consist of human form of the face and the body but their gender is not distinguishable. This was considered as the lowest fidelity 3D character. While, the gender of the 3D characters of the 'medium level' of anthropomorphism is distinguishable, however, they do not have hair or beard and they

wear uniform clothing. Finally, the 3D characters of the 'high level' of anthropomorphism consist of high fidelity human characteristics and wear distinguished clothes.

6.2 Experimental Procedure

6.2.1 Five Basic Scenes

In a block design the human subjects participated in several rounds of the Cyberball3D+ task and these rounds were combinations of low or high anthropomorphism, inclusion of all avatars, exclusion of human subject or exclusion of other player, simulating social exclusion or empathy for social exclusion. A typical paradigm of an experiment was to consist of six rounds and each of these rounds to be played twice but not consecutively, lasting 1.3 minutes. The first round simulated the inclusion of all players **(Scene 1)** and used a low level of anthropomorphism; the second round simulated the inclusion of all players **(Scene 1)** as the first round but used a high level of anthropomorphism for all players. The third and the fourth rounds of the game simulated the exclusion of the participant in the scanner **(Scene 2)** and used low and high level of anthropomorphism respectively. In addition, the fifth and the sixth rounds of the game simulated the exclusion of the programmed players **(Scene 3)** and used low and high level of anthropomorphism respectively. The task took 20 minutes. Between each round there was a 5-second break displaying a black screen. The order of rounds was counterbalanced amongst participants.

Two buttons from a 4 - button interface were used by participants to throw the ball left or right respectively (Figure 59). User interactions were synchronized to the fMRI scanner by using trigger information. A frequency modulated audio signal was generated at prescribed times within the experimental phase. The audio signal was fed into a biometric recorder, which also recorded heartbeat, scanning synchronization etc. A log was generated marking the exact time the sync pulses were sent to the biometric recorder as well as logs for user interactions.

Neuroscientists used a dedicated user interface to select the level of anthropomorphism of all avatars, the gender of each avatar and the fairness of the game. Particularly, before the game begun, the neuroscientists used a dedicated user interface to fill in the overall number of rounds of the game (Figure 48). Moreover, the neuroscientists selected the level of anthropomorphism of all avatars, the gender of each avatar, the level of fairness of the round represented by the selected players to be excluded from the game and the duration time of each round. When the researchers filled in all parameters of the game,

the game began and the participants in the fMRI scanner interactively played Cyberball3D+ by using the button boxes.

The fMRI scanner acquired brain images at strictly specified timings while the participants followed the experimental protocol, performing the tasks assigned to them while being immersed in the VEs such as throwing the ball to a player. Meanwhile, participants' physiological measures were acquired, such as heart rate and heart pulse oximetry. In addition, this neuroscientific protocol implemented maintained the imposed time limits and was completely synchronized with the fMRI scanner.

Participants followed a formal neuroscientific experimental protocol for fMRI Cyberball3D+ task for which healthy controls were recruited to an existing study of abnormal skin sensations. This study included structural imaging and DTI. They undertook an fMRI emotional processing task exploring effects of skin and infestation related images. They also complete questionnaire measures of Alexithymia (TAS), Anxiety (BAI), Interoceptive Sensibility (Porges Body Perception Questionnaire), Empathy (BEES) and hypermobility (Beighton score). In addition, the participants underwent laboratory testing for Interoceptive Sensitivity (heartbeat detection and mental tracking tasks) and the Rubber Hand Illusion. During the completion of the Cyberball3D+ task they underwent simultaneous heart rate recording and pupillometry.

The 3D Cyberball3D+ paradigm conducted investigated two hypotheses. The first hypothesis was that the emotional response of inclusion and exclusion of self and other would be modulated by the level of anthropomorphism of the players. The second hypothesis was that the exclusion of others would activate similar networks (social pain matrix) to watching exclusion of self, therefore, eliciting empathy.

6.2.2 Five Basic Scenes and Reinforcement Learning

The Reinforcement Learning as described in a previous section is a type of Machine Learning, and thereby also a branch of Artificial Intelligence. It allows machines and software agents to automatically determine the best behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal. Reinforcement Learning allows the machine or software agent to learn its behavior based on feedback from the environment. This behavior can be learnt once and for all, or keep on adapting as time goes by.

In this work we used the Reinforcement learning methods to implement a more sophisticated version of Cyberball and incorporating a form of learning affecting players'

behavior. This sophisticated version of Cyberball made the game more competitive and intelligent integrating rules and scores as well as creating intelligent agents modeling their decision making behavior.

We created two potential rules that were the ability of each player to throw the ball to any of the rest of the players of the game and the inclusion of a reward system whereby points were given to or removed from a player depending on his/her ball throws. In particular, if a player threw the ball to the player that had the highest number of receiving tosses he/she lost a point while if a player threw the ball to the player that had the lowest number of receiving tosses he/she won a point. In addition, if a player threw the ball to a player that had a medium number of receiving tosses he/she did not lose or win points. The player that obtained the highest score of points he/she won the game. The score of the participant was displayed on the screen during the rounds of the game.

Participants were informed that they were able to throw the ball to any player in the game and also that there was a reward system. Although they were not informed on how they gain or lose points, they were able to figure it out on their own, if they paid attention on the pointing system since the score appear on the screen. The purpose of this approach was to make the players to understand that they had to feel empathy for the excluded players throwing the ball to them if they wanted to win the game.

In a block design the human subjects participated in two rounds of the Cyberball3D+ task and these rounds were combinations of low or high anthropomorphism as well as the using of the Scene4 from the 'Five Basic Scenes' version or the Reinforcement learning version. Particularly, the participants divided into two groups of six persons and each group of the participants played the same rounds of the game but in a different order of the rounds. In the one round of the game were used the low level of anthropomorphism and the scene4 from the five basic scenes version of the game in which the player 3 gets excluded by player 1. In the other round of the game were used the high level of anthropomorphism and the reinforcement learning version of the Cyberball. Both rounds of the game were lasting 180 seconds and in each group the starter player of the first round was Player 1 as well as in the second round was the Player 2.

6.3 Experimental Setup

6.3.1 Five Basic Scenes

The experiments were conducted inside the fMRI scanner. In order to synchronize the interactive 3D application with the fMRI scanner, several hardware parts had to be set up including the fMRI scanner, the synchronization box, the analog signal box, the button boxes, the sound mixer, the visual stimuli PC, on which the application was executed and the spike PC, which recorded the spike signals sent from the synchronization box. The Figure 60 describes the experimental setup and shows that the Visual Stimuli PC, which executed the interactive application, was connected to the fMRI projector and displayed the VEs on the projector screen. The button boxes were connected to that PC. The PC's sound card was connected to a sound mixer, which separated the two audio channels and directed the left towards the analog signal box and the right towards the participants' headphones. The analog signal box also received signals from the synchronization box which was connected to the fMRI scanner, as well as from the heart rate recorder attached to the participant's toe. The spike PC was dedicated in recording the inputs received from the analog signal box.

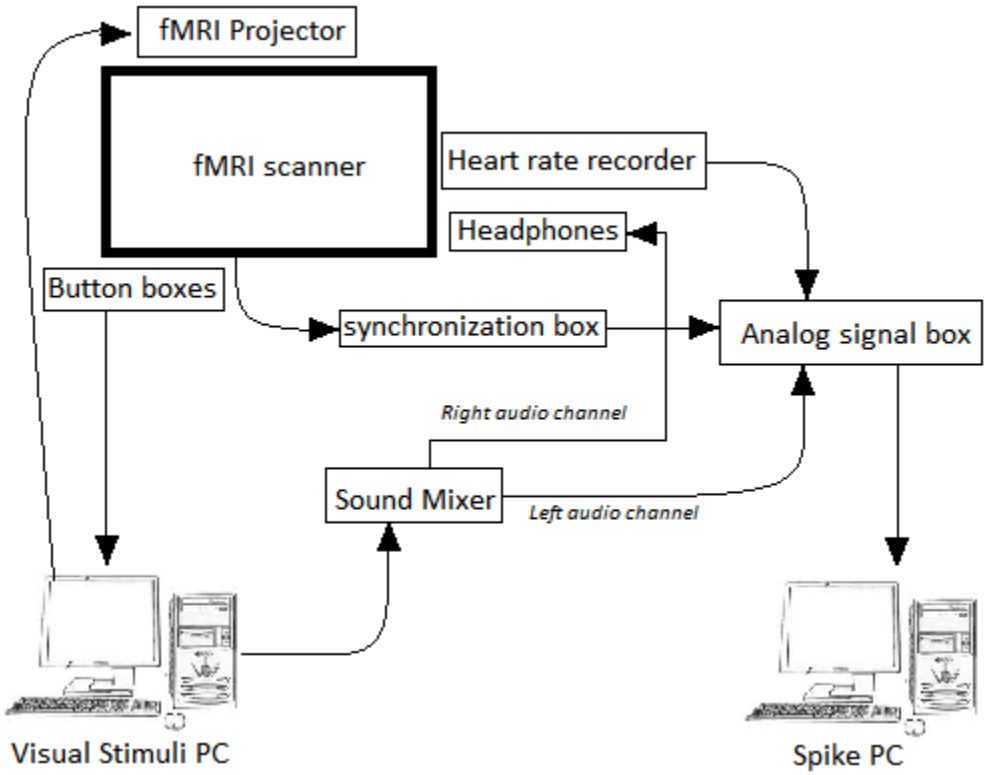


Figure 60: The experimental setup.

Whole brain functional Magnetic Resonance Imaging (fMRI) data was acquired on a 1.5 T Siemens Avanto scanner. To minimise signal artefacts originating from the sinuses, axial slices were tilted 30º from the intercommissural plane. Thirty-four slices (3mm thick, 0.75 mm interslice gap) were acquired with an in plane resolution of 3 x 3 mm (repetition time = 2.52 per volume, echo time = 43ms. The participants' heads were placed inside the coil that can be seen in the fMRI scanner. Through a mirror on top of the coil, the participants could see the VEs displayed on the projector screen behind them.

6.3.2 Five Basic Scenes and Reinforcement Learning

The experimental setup of the experiments of the second stage was very simple. The experiments were conducted at the Laboratory of Distributed Multimedia Information Systems and Applications of the Technical University of Crete. The participants played the game in a Standard PC of the laboratory and they used only the arrows buttons of the computer keyboard to throw the ball to the virtual players.

6.4 Data Analysis

6.4.1 Five Basic Scenes

Functional neuroimaging involved the measurement of brain activity. The scanner produced a map of the brain that was represented as voxels. A voxel represents a value on a grid in three-dimensional space – a combination of volume and pixel. As such each voxel represents the activity of a particular co-ordinate in the three dimensional space of the brain.

Acquired images were conventionally pre-processed before statistical analysis to remove noise and correct for sampling error.

Pre-processing was performed so data would approximate the following assumptions – all voxels in any given image of the series of images taken over time were acquired at the same time; each data point in the time series from a given voxel was collected from that voxel only; residual variance had a Gaussian distribution; when carrying out analyses across different subjects any given voxel corresponded to the same brain structure in all the subjects in the study. For example, to account for the motion of the head between scans, images were adjusted so each of the voxels corresponds to the same site in the brain; this is known as realignment. As imaging studies involved multiple participants who had slightly differently shaped brains, a process of normalization was employed so that each 3D image was transformed so that key brain structures line up. They were then set into standard space. Images were smoothed so voxels were averaged with their neighbours using a Gaussian filter to reduce noise. As such, standard spatial preprocessing [realignment, coregistration, segmentation, normalisation to Montreal Neurological Institute (MNI) space, and smoothing with an 8-mm FWHM Gaussian Kernel] was performed. Voxel size was interpolated during pre-processing to isotropic 3 x 3 x 3 mm.

Results were analysed using Statistical Parametric Mapping software (SPM8) on a Matlab platform. First level (individual) analysis modeled timing of block stimuli (i.e type of round of the game – level of anthropomorphism and level of inclusion) using the general linear model; a full factorial design was used at the second level (group) of analysis (random effects analysis).

6.4.2 Five Basic Scenes and Reinforcement Learning

The application was recording every action of the players, the score of each player as well as the total receiving tosses and scores of each player in a separate log file dedicated to each experimental round of the game in order to be able to understand and analyze the data gathered from each experiment. For each participant, a different log file was created for each round which follows the following naming convention: log(Participant Number)_r(Number of Round).csv and it recorded each ball tossing occurred by all players. The first data column recorded the time stamp when a player threw a ball, the second column which player threw the ball and the third column which player got the ball. In addition, for each participant, different log files were created for each round and each player which follow the following naming convention: log(Participant Number)_r(Number of Round)_P(Number of Player).csv and they recorded the score of each player of the game every time a ball tossing occurred . The first data column recorded the time stamp when a player threw a ball and the second column the score of that player. Finally, For each participant, a different log file was created for each round which follows the following naming convention: logScores(Participant Number)_r(Number of Round).csv and it recorded the total receiving tosses and scores of each player every time a ball tossing occurred. The time was measured in milliseconds starting just at the start of the game of the experiment assumed to be time point 0.

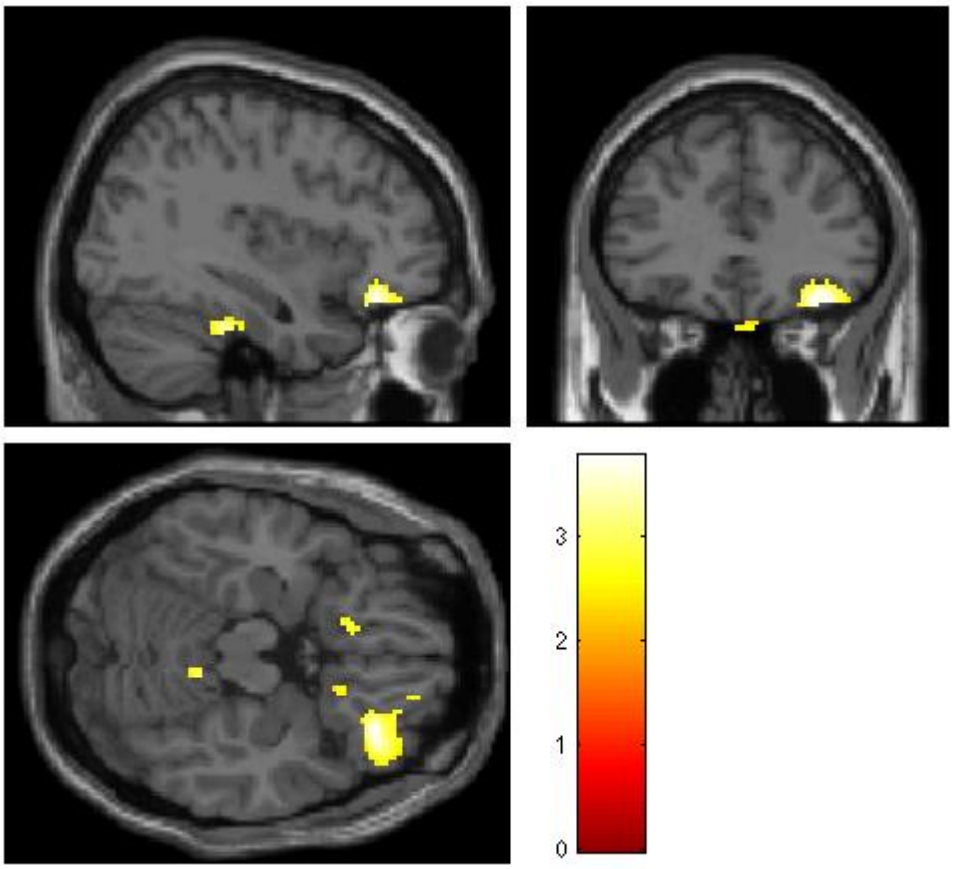
Each .csv file of each participant was analyzed using the Microsoft Office Excel program. Particularly, each .csv file was converted into .xls file in order to be able to design statistical charts. After the conversion of the .csv files into excel files we found the percentages of receiving/throwing tosses that each player got/threw from/to the other players in each round using mathematical functions. In addition, we had the information for any time of each round which player threw and got the ball, respectively, in order to have the ability to analyze the behavior of each participant during the game. Furthermore, we designed scatter charts describing the score of each participant based the time during the Reinforcement Learning round of the experiment.

6.5 Results

6.5.1 Five Basic Scenes

A whole brain analysis was performed with an uncorrected significance threshold of p <0.01. Threshold significance was set using the cluster extent to manage multiple comparisons across the whole brain **(Slotnick, 2008)**. 10,000 Monte Carlo simulations determined that clusters of 64 or more contiguous voxels activated at an uncorrected voxel-wise threshold of p<0.01 ensured the probability of Type-1 statistical errors was below 0.05.

CONTRAST HIGH>LOW ANTHROPOMORPHISM

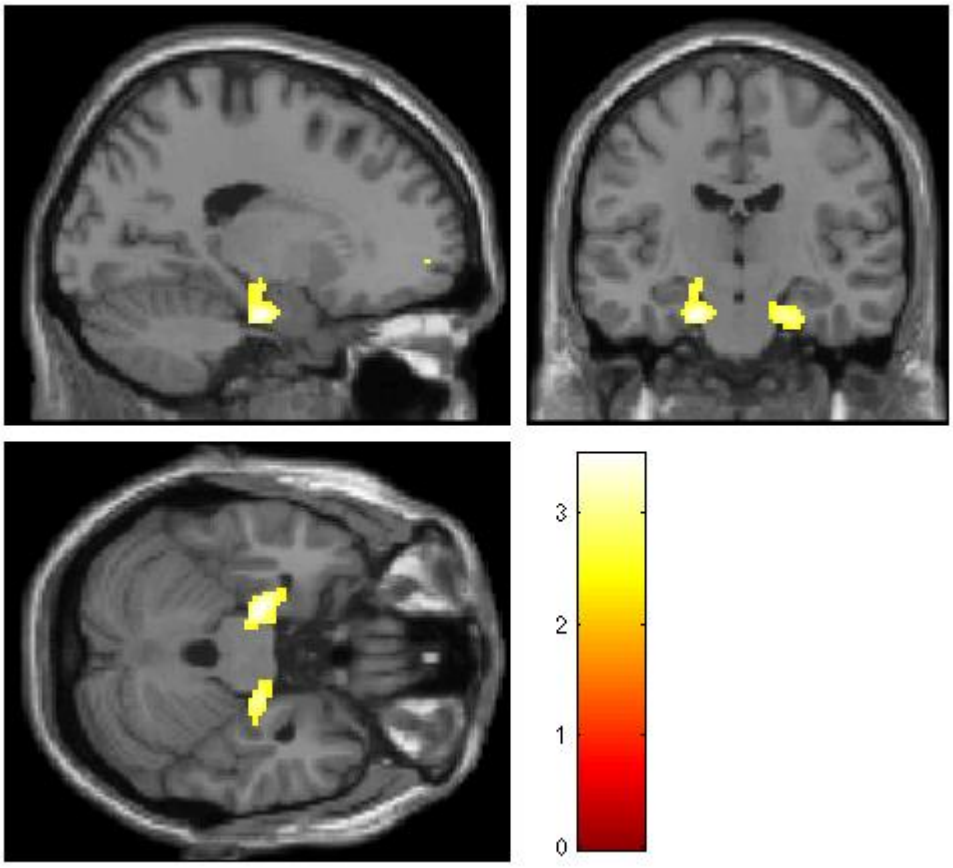


T-Contrast estimates showing main effect of high versus low anthropomorphism demonstrating activity in right orbitofrontal cortex.

TABLE II: Activations in the brain based on the contrast between the high and low anthropomorphism level.

Region	Cluster size (voxels)	X	Y	Z	Z score
Right lateral orbitofrontal cortex	252	38	34	-18	3.68
Right cerebellum	346	38	-32	30	3.42
Left superior temporal gyrus	170	-40	8	-26	3.33
Left lateral orbitofrontal cortex	182	20	20	-22	3.34

CONTRAST EXCLUSION OF OTHER > INCLUSION OF SELF

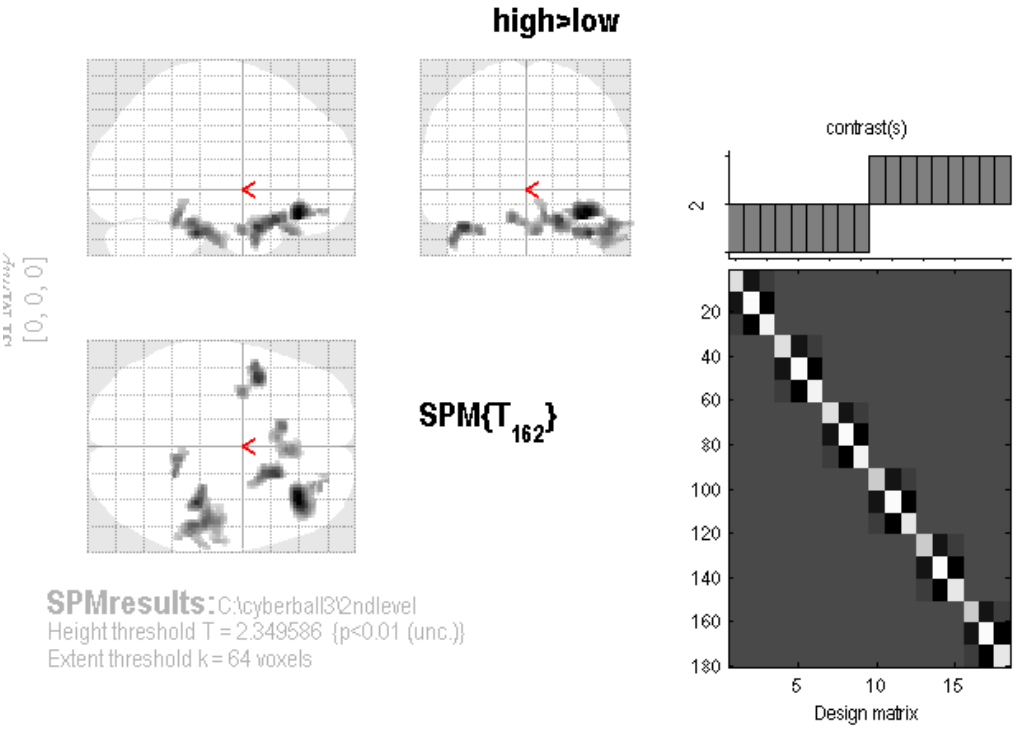


T-Contrast estimates showing main effect of exclusion of other versus inclusion, demonstrating activity in bilateral parahippocampal regions.

TABLE III: Activations in the brain based on the contrast between the exclusion of other and inclusion of self.

Region	Cluster size (voxels)	X	y	z	Z score
Left parahippocampus	246	-18	-16	-14	3.52
Left superior frontal gyrus and anterior cingulate	103	-15	56	-4	3.38
Left middle frontal gyrus	109	-28	42	36	3.19
Right parahippocampus	138	22	-16	-26	3.11

This work showed interesting differences when playing with high level anthropomorphic characters compared to low level avatars. Participating in a high anthropomorphism environment rather than a low anthropomorphism environment revealed significant activations in both frontal cortex and superior temporal gyrus. One can attribute this activation pattern to the imprecise mapping of avatar features to normative 'top-down' representational expectancies of the human body within extrastriate visual cortices (particularly areas like STS that are functionally tied to humans emotional signals) and to the processing of negatively-valenced stimuli, activating lateral orbitofrontal cortex. This suggests that compared to more human like avatars, playing the non-anthropomorphic avatars is less subjectively rewarding. Therefore, when studying complex emotional responses, a high level of anthropomorphism of synthetic characters is not only required but also able to engage common neuroscientific patterns of brain activation as in real-world circumstances. In addition, watching the exclusion of other players the bilateral parahippocampus regions that play an important role in the encoding and recognition of environmental scenes as well as the amygdala region of the brain which it relates to negative emotions, especially fear and sadness were activated. More complex accounts for these observations must draw more heavily on neural predictive coding models of perception.



Statistics: ***p-values adjusted for search volume***

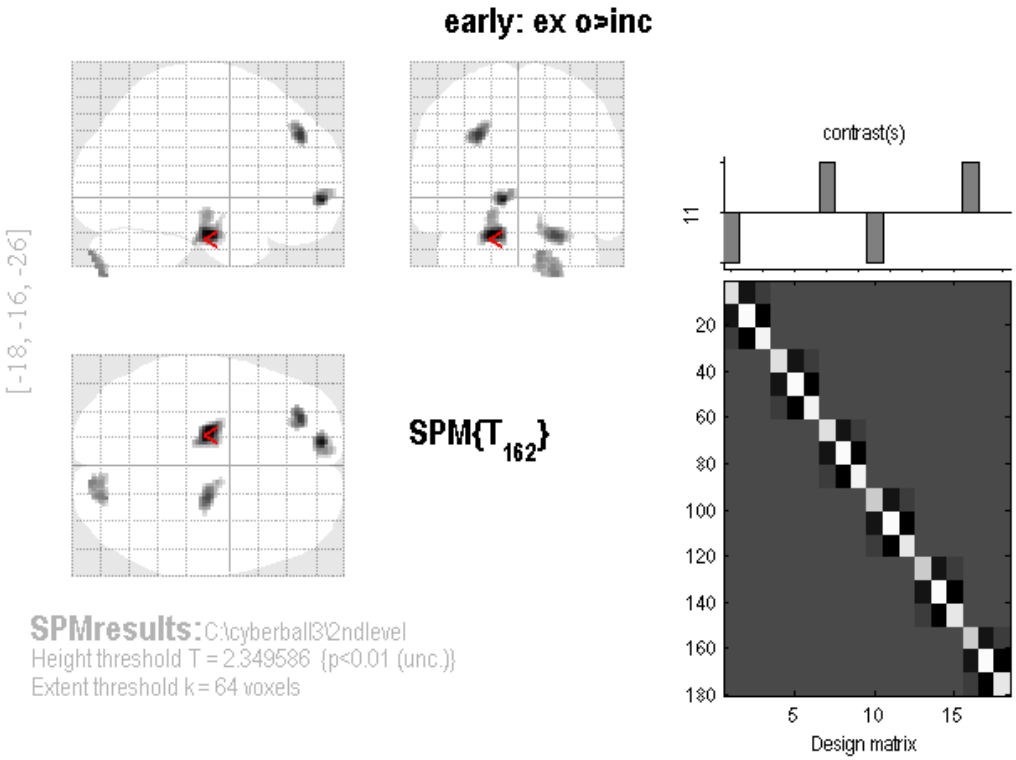
set-level		cluster-level				peak-level					mm mm mm		
p	c	$p_{FWE\text{-}corr}$	$q_{FDR\text{-}corr}$	k_E	p_{uncorr}	$p_{FWE\text{-}corr}$	$q_{FDR\text{-}corr}$	T	$(Z_{=})$	p_{uncorr}			
0.980	6	0.814	0.654	252	0.037	0.866	0.936	3.76	3.68	0.000	38	34	-18
						1.000	0.936	2.80	2.76	0.003	28	42	-18
						1.000	0.936	2.70	2.67	0.004	24	52	-16
		0.540	0.604	346	0.017	0.994	0.936	3.42	3.35	0.000	38	-32	-30
						1.000	0.936	3.16	3.11	0.001	52	-34	-24
						1.000	0.936	3.16	3.11	0.001	50	-26	-28
		0.972	0.699	170	0.080	0.996	0.936	3.39	3.33	0.000	-40	8	-26
						1.000	0.936	3.08	3.03	0.001	-50	6	-34
						1.000	0.936	2.96	2.91	0.002	-34	-2	-28
		0.959	0.699	182	0.071	0.998	0.936	3.34	3.28	0.001	20	20	-22
		0.995	0.836	131	0.119	1.000	0.936	3.06	3.01	0.001	-10	24	-20
						1.000	0.936	2.99	2.95	0.002	4	26	-28
						1.000	0.936	2.85	2.81	0.002	-4	20	-24
		1.000	0.921	85	0.203	1.000	0.936	3.04	3.00	0.001	18	-46	-22
						1.000	0.936	2.99	2.95	0.002	10	-42	-18

table shows 3 local maxima more than 8.0mm apart

Height threshold: T = 2.35, p = 0.010 (1.000)
Extent threshold: k = 64 voxels, p = 0.268 (1.000)
Expected voxels per cluster, <k> = 56.232
Expected number of clusters, <c> = 12.04
FWEp: 4.849, FDRp: Inf, FWEc: Inf, FDRc: Inf

Degrees of freedom = [1.0, 162.0]
FWHM = 11.9 11.7 11.4 mm mm mm; 6.0 5.9 5.7 {voxels}
Volume: 1607976 = 200997 voxels = 942.7 resels
Voxel size: 2.0 2.0 2.0 mm mm mm; (resel = 198.76 voxels)

Figure 61: The brain activity in a peak, cluster and set level based on contrast between high and low level of anthropomorphism.



Statistics: ***p-values adjusted for search volume***

set-level		cluster-level				peak-level					mm mm mm
p	c	$p_{FWE\text{-}corr}$	$q_{FDR\text{-}corr}$	k_E	p_{uncorr}	$p_{FWE\text{-}corr}$	$q_{FDR\text{-}corr}$	T	$(Z_{\equiv})$	p_{uncorr}	
0.993	5	0.830	0.921	246	0.039	0.982	1.000	3.52	3.45	0.000	-18 -16 -26
						1.000	1.000	2.73	2.69	0.004	-20 -16 -14
		0.999	0.921	103	0.164	0.997	1.000	3.38	3.32	0.000	-12 56 -4
						1.000	1.000	2.63	2.60	0.005	-8 62 0
		0.999	0.921	109	0.153	1.000	1.000	3.19	3.14	0.001	-28 42 36
		0.993	0.921	138	0.111	1.000	1.000	3.11	3.06	0.001	22 -16 -26
						1.000	1.000	2.64	2.61	0.005	12 -10 -20
		0.999	0.921	113	0.146	1.000	1.000	2.85	2.82	0.002	22 -80 -50
						1.000	1.000	2.83	2.79	0.003	12 -90 -36
						1.000	1.000	2.81	2.78	0.003	20 -86 -44

table shows 3 local maxima more than 8.0mm apart

Height threshold: T = 2.35, p = 0.010 (1.000)
Extent threshold: k = 64 voxels, p = 0.268 (1.000)
Expected voxels per cluster, <k> = 56.232
Expected number of clusters, <c> = 12.04
FWEp: 4.849, FDRp: Inf, FWEc: Inf, FDRc: Inf

Degrees of freedom = [1.0, 162.0]
FWHM = 11.9 11.7 11.4 mm mm mm; 6.0 5.9 5.7 {voxels}
Volume: 1607976 = 200997 voxels = 942.7 resels
Voxel size: 2.0 2.0 2.0 mm mm mm; (resel = 198.76 voxels)

Figure 62: The brain activity in a peak, cluster and set level based on contrast between exclusion of other and inclusion of self.

6.5.2 Five Basic Scenes and Reinforcement Learning

In this stage of the experiments, 12 healthy adult volunteers separated into two groups of six persons participated in two rounds of the Cyberball3D+ task. Each group of the participants played the same rounds of the game but in a different order of the rounds. In the one round of the game were used the low level of anthropomorphism and the scene4 from the 'Five Basic Scenes' version of the game in which the player 3 gets excluded by player 1. In the other round of the game were used the high level of anthropomorphism and the Reinforcement learning version of the Cyberball. The Group1 played first the round using the Scene4 from the 'Five Basic Scenes' version and secondly the round using the Reinforcement learning version of the Cyberball. The Group2 played the same rounds with Group1 but in reverse order.

In the round that we used the Scene4 from the 'Five Basic Scenes' version of the Cyberball the Player1 excluded the Player3 and the Player2 included all other players in the game. However, the Player3 and Participant were not able to throw the ball to each other straightly only through the Players 1 and 2. The Figure 63 shows the percentages of throwing tosses from player 1 and 2 to other players, respectively. In the round that we used the Reinforcement learning version of the Cyberball game the participants were not informed the way that the virtual players played as well as how they gain or lose points, they were able to figure it out on their own.

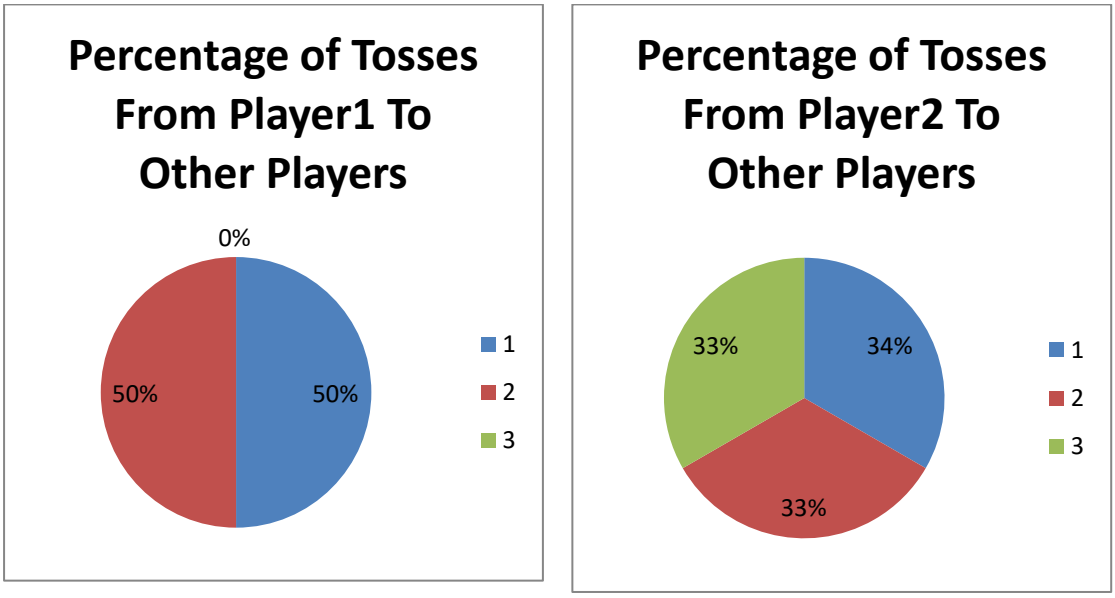


Figure 63: The screenshots show the percentages of throwing tosses from players 1 and 2 to the other players, respectively.

GROUP1

The Group1 consisted of three female and three male. In the first round of the game the most participants threw the ball to the Player2 more times in relation to the Player1; only one participant threw the ball to the Player1 more. On the contrary, in the second round the most participants threw the ball to the Player1 more times in relation to the Player2.

A) Round1 using Scene4 from the 'Five Basic Scenes' version

The participants of the Group1 played firstly the scene4 from the 'Five Basic Scenes' version. As we can see in the figure below, the most participants threw the ball to the Player2 more because they felt empathy for the Player3 which got excluded by Player1 as well as they were not able to throw the ball directly to the Player3. The Player1 excluded the Player3 while the Player2 threw the ball equally to all players. For this reason, the participants threw the ball more to the Player2 in order the Player2 was able to include the Player3 in the game.

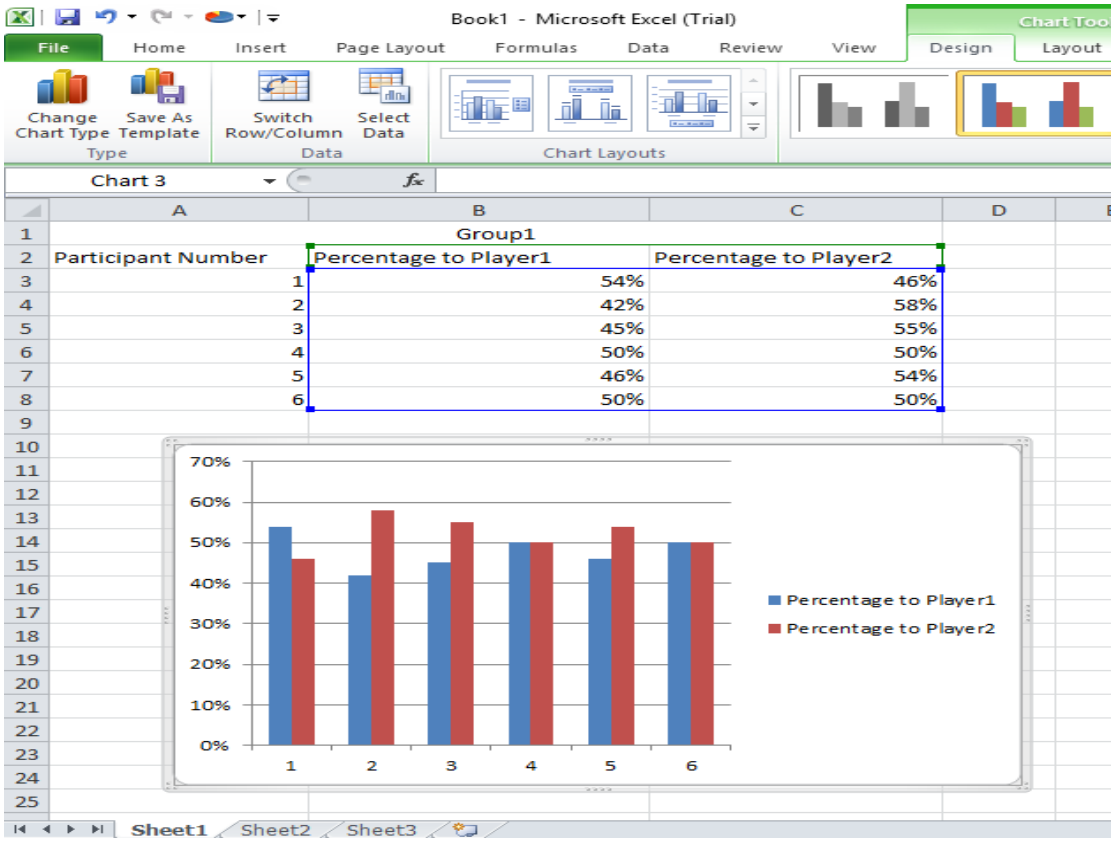


Figure 64: The percentages of throwing tosses of each participant of the Group1 to players 1 and 2.

B) Round2 using Reinforcement learning version

In the second round, the participants of the Group1 played the reinforcement learning version of the Cyberball. As we can see in the Figure 65, the most participants threw the ball to the Player1 more, except one participant that played the round with one virtual player only. In addition, the Player1 got the lowest number of receiving tosses so that means that the most participants showed empathy for this Player throwing the ball to him more. Therefore, the score of the participants that showed empathy for the Player with the lowest number of receiving tosses was on the increase. On the contrary, the score of the participant that played the round interacting only with one virtual player had a downward trend.

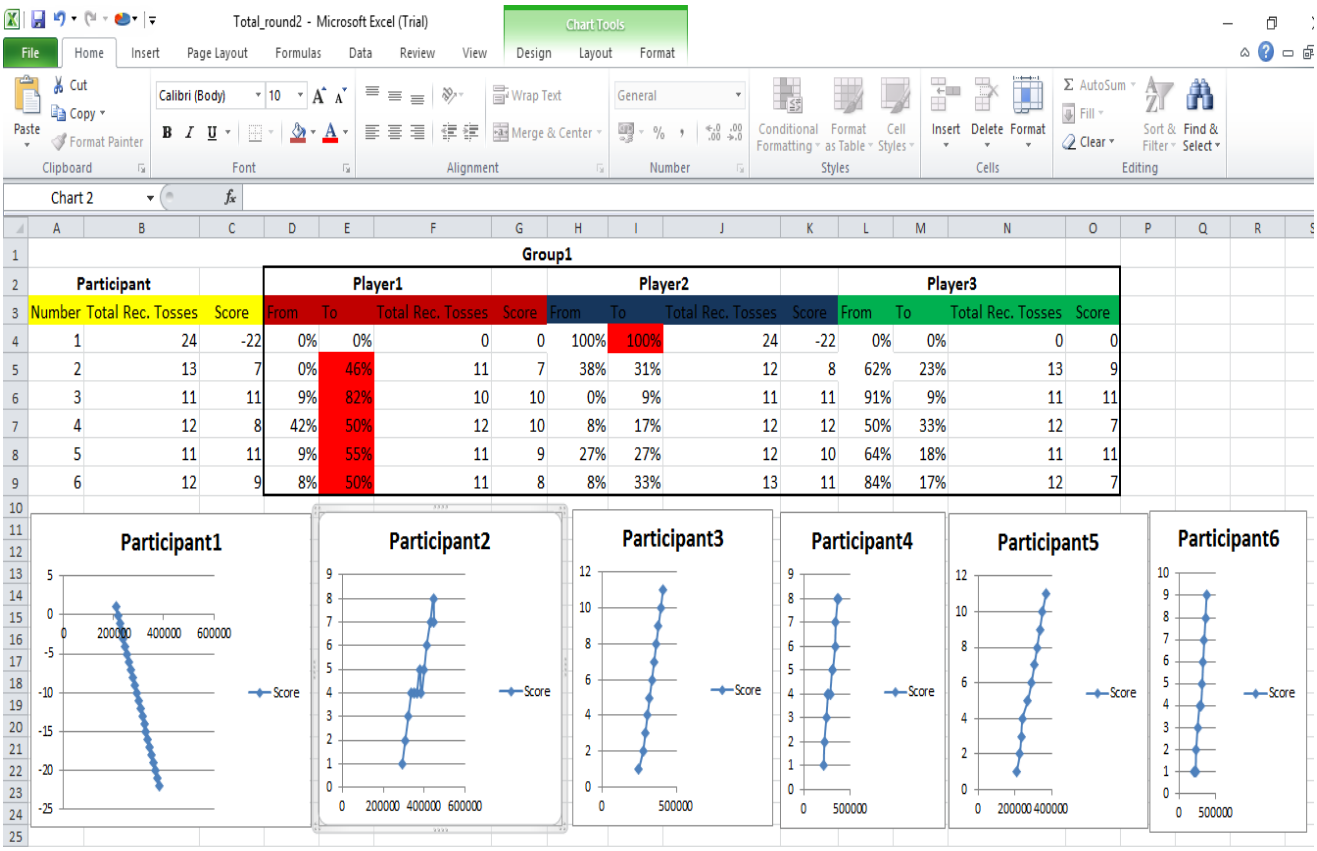


Figure 65: The percentages of receiving/throwing tosses From/To the virtual players To/From the Participants of the Group1 as well as the scores of each participant.

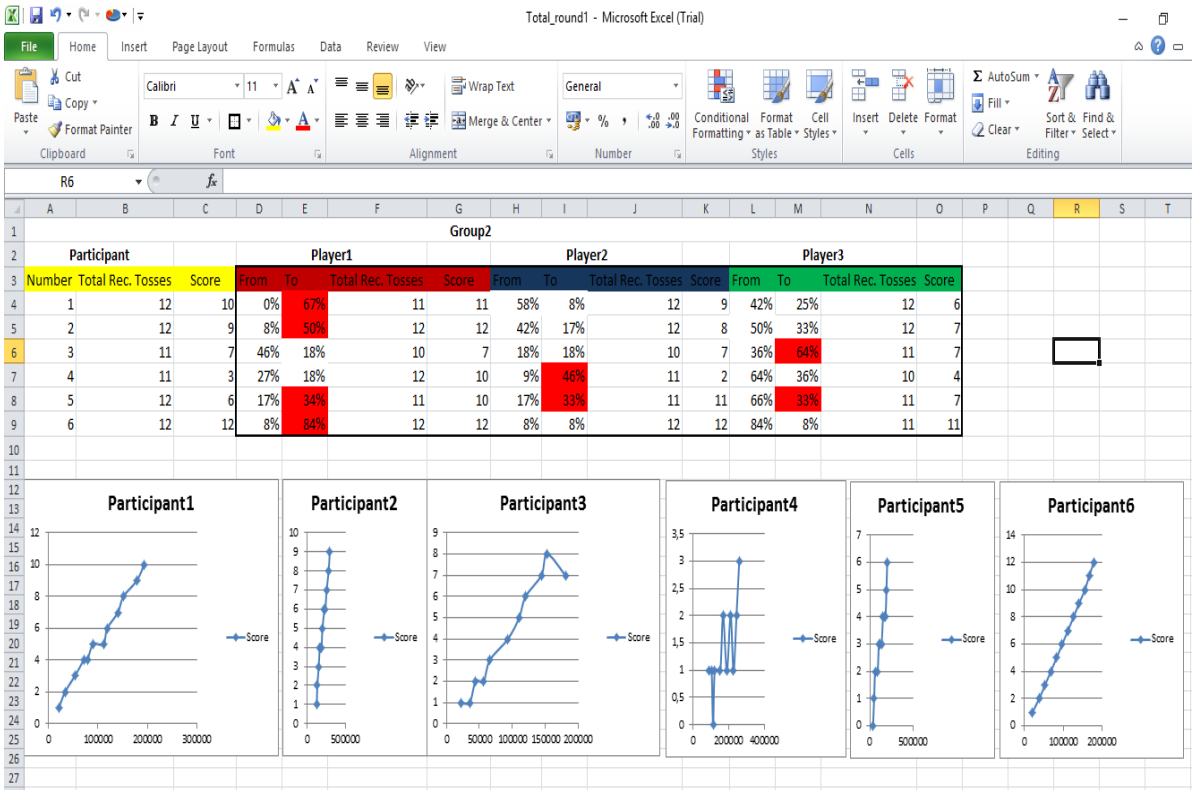
To sum up, observing the behavior of the participants of the Group1, we can see that the most of the participants felt empathy for the excluded Players and they tried to include them in both rounds of the game as well as only one participant did not feel empathy and he cared more about his inclusion.

GROUP2

The Group2 consisted of six male. Most participants of this group threw the ball to the Player1 more times in relation to the Player2 in both rounds.

A) Round1 using Reinforcement learning version

In the first round, the participants of the Group2 played the reinforcement learning version of the Cyberball. As we can see in the Figure 66, the most participants threw the ball to the Player1 more, except two participants who threw the ball to the Players 2 and 3 more, respectively. In addition, we observe that the scores of the participants had strong fluctuations. The reason of these fluctuations was that the participants played firstly the reinforcement learning scene that was a competitive round with scores and they had to find on how they could gain points, unlike the round1 of the Group1 that was the round with the Scene4 which was more relaxing and with no competition.



Participant			Player1				Player2				Player3			
Number	Total Rec. Tosses	Score	From	To	Total Rec. Tosses	Score	From	To	Total Rec. Tosses	Score	From	To	Total Rec. Tosses	Score
1	12	10	0%	67%	11	11	58%	8%	12	9	42%	25%	12	6
2	12	9	8%	50%	12	12	42%	17%	12	8	50%	33%	12	7
3	11	7	46%	18%	10	7	18%	18%	10	7	36%	64%	11	7
4	11	3	27%	18%	12	10	9%	46%	11	2	64%	36%	10	4
5	12	6	17%	34%	11	10	17%	33%	11	11	66%	33%	11	7
6	12	12	8%	84%	12	12	8%	8%	12	12	84%	8%	11	11

Figure 66: The percentages of receiving/throwing tosses From/To the virtual players To/From the Participants of the Group2 as well as the scores of each participant.

B) Round2 using Scene4 from the 'Five Basic Scenes' version

In the second round, the participants of the Group2 played the scene4 from the 'Five Basic Scenes' version. As we can observe in the figure below, the most participants threw the ball to the Player1 more because they were influenced from their performance in the first round. Particularly, because the first round was a competitive round they had to find on how they could gain points in order to be able to win this round of the game. Therefore, the most participants observed that throwing the ball to the Player1 more they gain points, so they followed the same strategy and in the second round of the game.

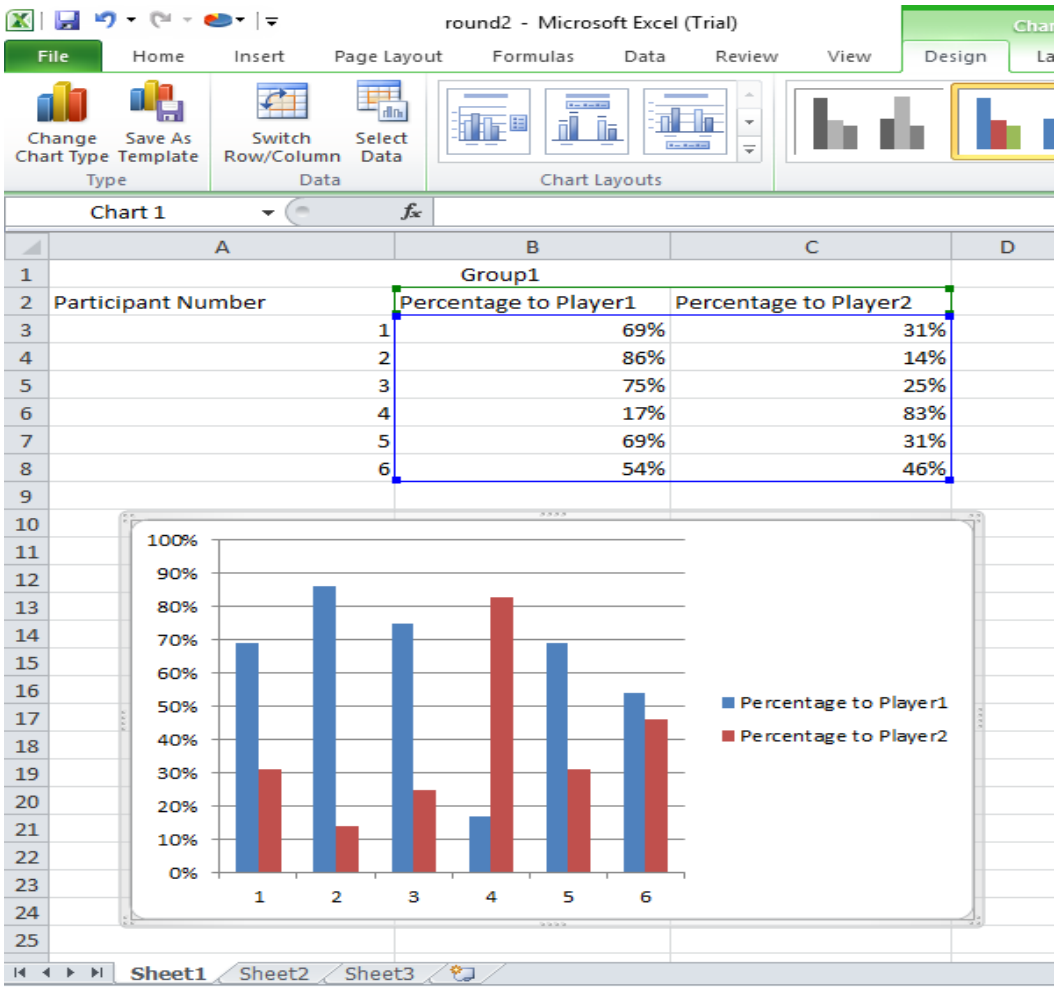


Figure 67: The percentages of throwing tosses of each participant of the Group2 to players 1 and 2.

To sum up, analyzing the behavior of the participants of the Group2, we can see that the most of the participants tried to find the way that they could gain points by observing the scores and the number of receiving tosses of each player in the first round of

the game as well as they adopted the same strategy in the second round of the game by throwing the ball to the same Player as did in the first round.

The Figures 68 and 69 show the average scores of the participants of the two Groups while playing the round of the game that we used the Reinforcement learning version. As we can observe, the main difference between these graphs is the trend in which the score increases. In the Group1 the score increases quickly and suddenly, while in the Group2 the score increases with steady and carefully steps by the participants. Therefore, as described above the participants of the Group1 due to they played firstly the scene4 and then the reinforcement learning scene they felt more the empathy feeling unlike the participants of the Group2 who played firstly the reinforcement learning scene that was a competitive round with scores.

Additionally, the figure 70 depicts two paradigms of the scores of two virtual players over intervals of time. As we can see from the upward trend of the scores, the playing of the virtual players was very competitive and intelligent and was very difficult for the participants to win the round in which we used the Reinforcement learning version of the Cyberball game.

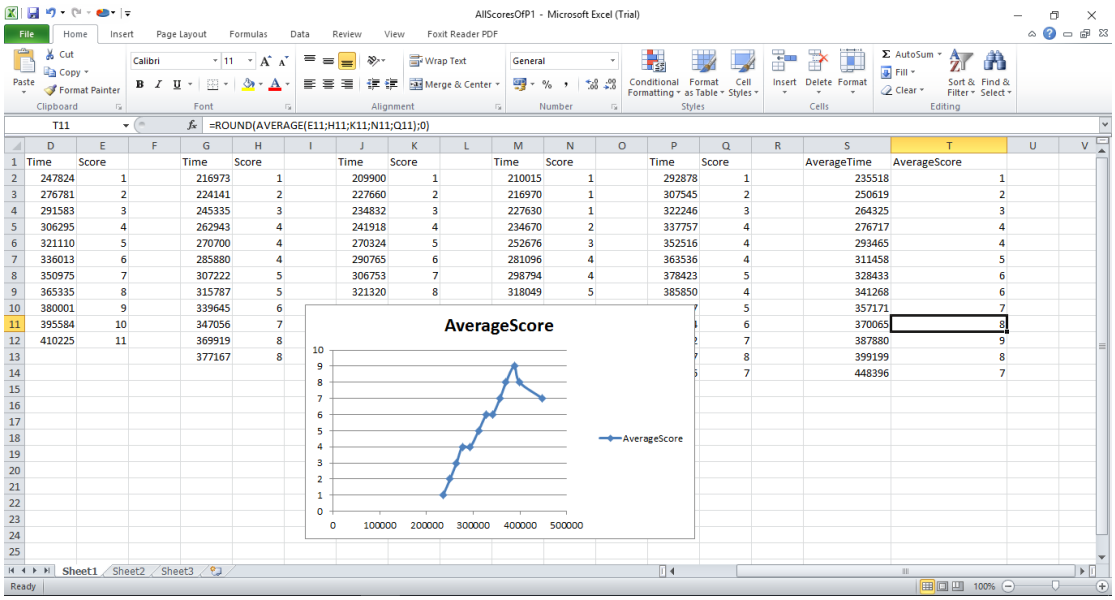


Figure68: The average scores of all participants of the group1 on intervals of time.

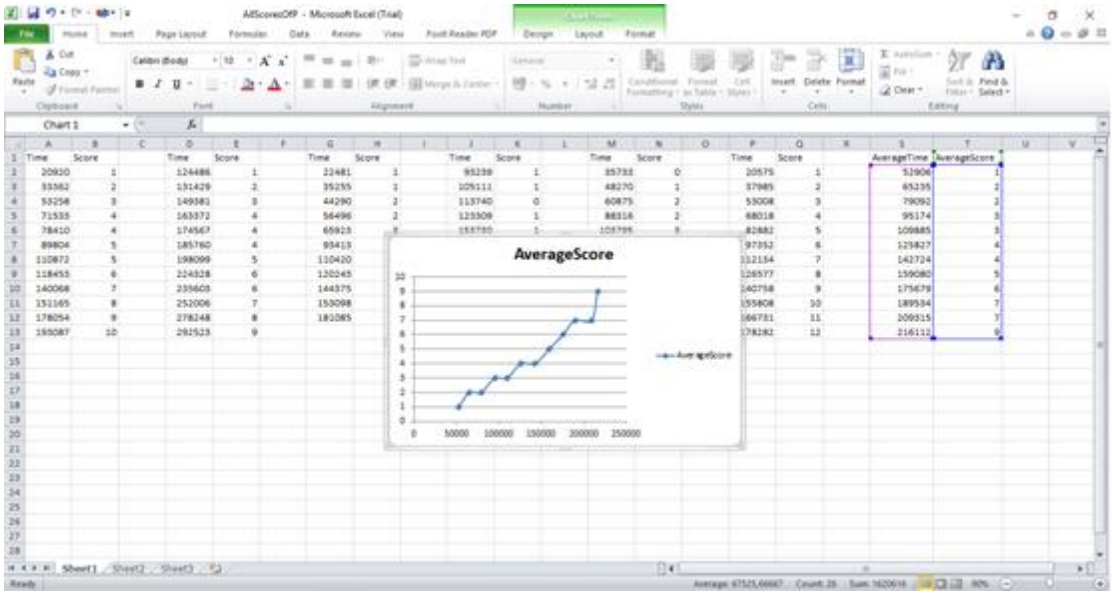


Figure 69: The average scores of all participants of the group2 in relation to time.

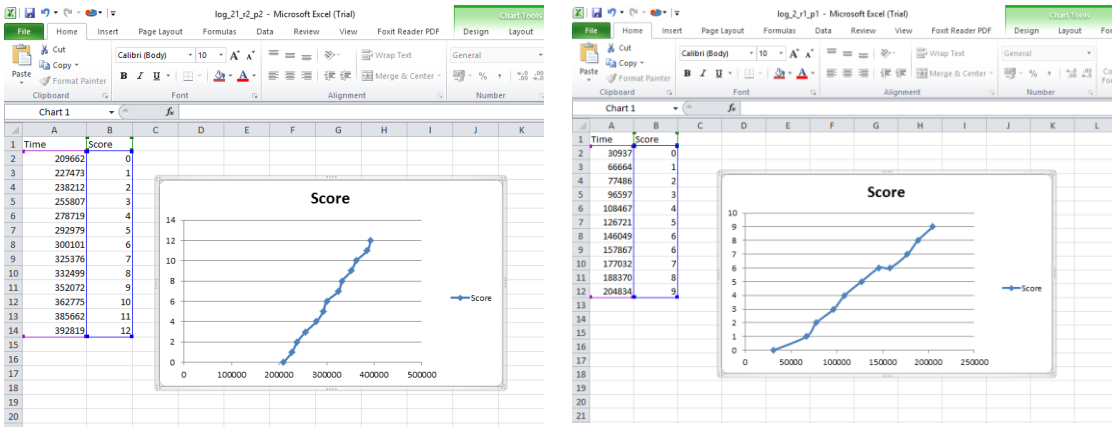


Figure 70: Screenshots depict the scores of two virtual players over intervals of time.

This chapter was concerned with the experimental methods employed when the actual experiments were conducted in the fMRI scanner at Brighton and Sussex Medical School and at the Technical University of Crete, respectively. Also, the experimental procedures as well as the results of the two groups of the experiments were presented.

7 Chapter 7 – Conclusions and Future Work

The presented framework put forward a sophisticated interactive real-time gaming system called Cyberball3D+ incorporating virtual characters to be played interactively in functional Magnetic Resonance Imaging (fMRI) for the study of empathy, social exclusion and ostracism. The 3D game proposed was designed to render an interactive VE on an fMRI display, enabling the conduct of formal neuroscientific experiments and investigating the effects of social exclusion, empathy and different level of anthropomorphism on human brain activity.

Although this work focused on the technical implementation of the system, the goal was to use this system to explore whether the pain felt by someone when socially excluded is the same when observing other people get socially excluded. Moreover, for the first time, we proposed a validated neuroscientific measure of character believability and emotional engagement.

It was challenge to develop an interactive system to be displayed in fMRI displays due to the infrastructural and technical demands. fMRI experiments of this type usually employ simple display material, for example using photographs, video clips or simple computerized stimuli which are non-stereoscopic. Using VEs in fMRI has the advantage that it is possible to involve participants in interactive animated environments which more realistically reflect social and emotional situations. This seamless naturalism and interactivity is impossible to achieve with video clips. In addition, some experiments could be only conducted using synthetic stimuli for ethical reasons.

The 3D gaming framework was developed in UDK, which enabled the development of interactively manipulated rendered synthetic scenes, as well as providing the necessary tools to overcome the technical difficulties of using the system in an fMRI display. The implementation of the 3D gaming framework was adjusted to address the challenges arising from the strict experimental protocol, including the creation of the synthetic scenes as well as the importing of the avatars and their animations. In order to make the VEs interactive, including the throwing of the ball to the virtual players, the 3D gaming framework responded to the fMRI-compliant button boxes, surpassing the challenge of handling user input.

The strict experimental protocol imposed time limits that the framework met, by specifying timers in order to change the experimental round of the game, meanwhile logging every action taking place in the VEs in a log file, in order to be able to understand and

analyze the data gathered from each experimental round of the game. The UIs were implemented through the use of Flash applications embedded in the framework and displayed on top of the displayed synthetic scene.

7.1 Main contributions

The innovative application designed to render an interactive Virtual Environment (VE) on an fMRI display, enabling the conduct of formal neuroscientific experiments and investigating the effects of social exclusion, empathy and different level of anthropomorphism on human brain activity. **For the first time,** the Cyberball game was implemented in three - dimensional form by modeling three levels of anthropomorphism as well as the neuroscientists were able to simulate any situation of the fairness of the game they wanted by filling in the percentages of the tosses that each player would throw to each one of the other players. Moreover, **for the first time**, we got a validated neuroscientific measure of character believability and emotional engagement at the same time the experience was taking place.

Results demonstrated that there are interesting differences when playing with high level anthropomorphic characters compared to low level avatars. Particularly, this work showed that participating in a high anthropomorphism environment rather than a low anthropomorphism environment activated both frontal cortex and superior temporal gyrus. This suggests that compared to more human like avatars, playing the non-anthropomorphic avatars is less subjectively rewarding and potentially anxiogenic. In addition, the results indicated that when studying complex emotional responses, a high level of anthropomorphism of synthetic characters engages neuroscientific patterns of brain activation as in real-world circumstances. Moreover, the results demonstrated that the exclusion of other activated emotional brain regions showing that the participants felt empathy for the excluded players.

An extension of the Cyberball3D+ game was to program players to be dynamically intelligent. For this reason, we included rules to the game and created intelligent avatars modeling their decision making behavior using the Q-learning algorithm. The rules consisted of the ability of each player to throw the ball to any of the rest of the players of the game and also the inclusion of a reward system; points were given or deducted from a player depending on his/her ball throws. The difference between this version and the previous one was that players did not have a pre-programmed behavior; in this version the players

dynamically adapted to the best strategy to win. For instance, if a player showed empathy and threw the ball more to the excluded player, then he got rewarded.

The results of the experiments of this version of the Cyberball indicated that when the participants played with the intelligent virtual players using the competitive round of the Reinforcement learning version without they trained first to a trial round in which were not used scores then they did not feel empathy for the excluded players because they focused to find the best strategy to win.

7.2 Implications for Future Work

The experiments described in detail in Chapter 6 were formally designed. However, certain improvements could be accomplished by the following actions:

- The experiments were designed to explore whether the pain felt by someone when socially excluded is the same when observing other people get socially excluded. It would be extremely interesting to explore whether there are differences in relation to empathy for friends and strangers. This could be employed by modelling the avatars of the game in order to look like a friend or a stranger, respectively.
- It would be innovative and useful to test the feelings of the social exclusion and empathy in even more extreme or more subtle fidelity variations. This could be employed by using the 'Probabilities' version of the game in which the neuroscientists are able to fill in the percentages of the tosses that each player makes to each one of the other players as well as modeling the avatars with highest fidelity of the anthropomorphism and utilizing motion capture data for the animations of the avatars.

8 References – Bibliography

Aitpayev and Gaber 2012	Aitpayev, K., & Gaber, J. (2012). Creation of 3d human avatar using kinect. *Asian Transactions on Fundamentals of Electronics, Communication & Multimedia*, *1*(5), 1-3.
Amato & Shani 2010	Amato, C., & Shani, G. (2010, May). High-level reinforcement learning in strategy games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1* (pp. 75-82). International Foundation for Autonomous Agents and Multiagent Systems.
Andari et al. 2010	Andari, E., Duhamel, J. R., Zalla, T., Herbrecht, E., Leboyer, M., & Sirigu, A. (2010). Promoting social behavior with oxytocin in high-functioning autism spectrum disorders. *Proceedings of the National Academy of Sciences*, *107*(9), 4389-4394.
Bailenson et al. 2008	Bailenson, J. N., Yee, N., Blascovich, J., & Guadagno, R. E. (2008). Transformed social interaction in mediated interpersonal communication. *Mediated interpersonal communication*, 77-99.
Bartlett et al. 2012	Bartlett, M. Y., Condon, P., Cruz, J., Baumann, J., & Desteno, D. (2012). Gratitude: Prompting behaviours that build relationships. *Cognition & emotion*, *26*(1), 2-13.
Bartneck et al. 2009	Bartneck, C., Kulić, D., Croft, E., & Zoghbi, S. (2009). Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots. *International journal of social robotics*, *1*(1), 71-81.
Bhatt & Camerer 2005	Bhatt, M., & Camerer, C. F. (2005). Self-referential thinking and equilibrium as states of mind in games: fMRI evidence. *Games and Economic Behavior*, *52*(2), 424-459.
Blanz and Vetter 1999	Blanz, V., & Vetter, T. (1999, July). A morphable model for the synthesis of 3D faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (pp. 187-194). ACM Press/Addison-Wesley Publishing Co.
Blascovich et al 2002	Blascovich, J., Loomis, J., Beall, A. C., Swinth, K. R., Hoyt, C. L., & Bailenson, J. N. (2002). Immersive virtual environment technology as a methodological tool for social psychology. *Psychological Inquiry*, *13(2)*, 103-124.
Breidt et al. 2003	Breidt, M., Wallraven, C., Cunningham, D., & Bulthoff, H. (2003). Combining 3d scans and motion capture for realistic facial animation. *Proceedings der Eurograph,(Eds.) Julian and F. and P. Cano and The Eurographics Association*, 63-66.
Burchardt et al, 2002	Burchardt, T., Le Grand, J. and Piachaud, D. (2002) 'Degrees of exclusion: developing a dynamic multidimensional measure', in J. Hills, J. Le Grand and D. Piachaud (eds) Understanding social exclusion, Oxford: Oxford University Press, pp 30-43.
Christodoulou et al. 2012	Christodoulou, G., Radulescu, E., Medford, N., Critchley, H., Watten, P. L., & Mania, K. (2012). A 3D Lighting System for fMRI Rendering Fidelity Experiments. *International Journal of Interactive Worlds*, *2012*, 1.

Cole, Yoo and Knutson 2012	Cole, S. W., Yoo, D. J., & Knutson, B. (2012). Interactivity and reward-related neural activation during a serious videogame.
Cosker, Krumhuber & Hilton 2010	Cosker, D., Krumhuber, E., & Hilton, A. (2010, July). Perception of linear and nonlinear motion properties using a FACS validated 3D facial model. In*Proceedings of the 7th Symposium on Applied Perception in Graphics and Visualization* (pp. 101-108). ACM.
Curio et al 2006	Curio, C., Breidt, M., Kleiner, M., Vuong, Q. C., Giese, M. A., & Bülthoff, H. H. (2006, July). Semantic 3d motion retargeting for facial animation. In *Proceedings of the 3rd symposium on Applied perception in graphics and visualization* (pp. 77-84). ACM.
Curio et al. 2008	Curio, C., Giese, M. A., Breidt, M., Kleiner, M., & Bülthoff, H. H. (2008, August). Probing dynamic human facial action recognition from the other side of the mean. In *Proceedings of the 5th symposium on Applied perception in graphics and visualization* (pp. 59-66). ACM.
Davis 1983	Davis, M. H. (1983). Measuring individual differences in empathy: evidence for a multidimensional approach. *Journal of personality and social psychology*,*44*(1), 113.
Decety and Lamm 2006	Decety, J., & Lamm, C. (2006). Human empathy through the lens of social neuroscience. *The Scientific World Journal*, *6*, 1146-1163.
Decety et al. 2013	Decety, J., Chen, C., Harenski, C., & Kiehl, K. A. (2013). An fMRI study of affective perspective taking in individuals with psychopathy: imagining another in pain does not evoke empathy. *Frontiers in Human Neuroscience*, *7*.
DiSalvo & Gemperle 2003	DiSalvo, C., & Gemperle, F. (2003, June). From seduction to fulfillment: the use of anthropomorphic form in design. In *Proceedings of the 2003 international conference on Designing pleasurable products and interfaces* (pp. 67-72). ACM.
D'Mell 2012	D'Mell, A. (2012). *The Effects of Social Reward on Reinforcement Learning*(Doctoral dissertation).
Duncan 2009	Duncan, J. (2009). The unusual birth of benjamin button. *Cinefex*, *116*, 70-99.
Eisenberg and Miller 1987	Eisenberg, N., & Miller, P. A. (1987). The relation of empathy to prosocial and related behaviors. *Psychological bulletin*, *101*(1), 91.
Eisenberger 2012	Eisenberger, N. I. (2012). The pain of social disconnection: examining the shared neural underpinnings of physical and social pain. *Nature Reviews Neuroscience*, *13*(6), 421-434.
Ekanayake et al. 2013	Ekanayake, H. B., Fors, U., Ramberg, R., Ziemke, T., Backlund, P., & Hewagamage, K. P. (2013). Affective Realism of Animated Films in the Development of Simulation-Based Tutoring Systems. *International Journal of Distance Education Technologies (IJDET)*, *11*(2), 96-109.
Ekman et al. 2002	Ekman, P., Friesen, W. V., & Hager, J. C. Facial action coding system. 2002.*Salt Lake City: Research Nexus eBook*.
Freeman et al. 2008	Freeman, D., Pugh, K., Antley, A., Slater, M., Bebbington, P., Gittins, M., ... & Garety, P. (2008). Virtual reality study of paranoid thinking in the general population. *The British Journal of Psychiatry*, *192*(4), 258-263.

Glueck et al. 2012	Glueck, M., Khan, A., Fiume, E., & Jackson, K. (2012). *Modeling and simulation of skeletal muscle for computer graphics: A survey*. Now.
Gong 2007	Gong, L. (2008). How social is social responses to computers? The function of the degree of anthropomorphism in computer representations. *Computers in Human Behavior*, *24*(4), 1494-1509.
Griesser et al. 2007	Griesser, R. T., Cunningham, D. W., Wallraven, C., & Bülthoff, H. H. (2007, July). Psychophysical investigation of facial expressions using computer animated faces. In *Proceedings of the 4th symposium on Applied perception in graphics and visualization* (pp. 11-18). ACM.
Hammond 2007	Hammond, D. C. (2007). What is neurofeedback?. Journal of Neurotherapy, 10(4), 25-36.
Hoffman 2001	Hoffman, M. L. (2001). *Empathy and moral development: Implications for caring and justice*. Cambridge University Press.
Ickes 1997	Ickes, W. J. (Ed.). (1997). *Empathic accuracy*. Guilford Press.
Igarashi et al. 1999	Igarashi, T., Matsuoka, S., Tanaka, H. Teddy. (1999): a sketching interface for 3D freeform design. SIGGRAPH, p.409-416.
Ishiguro H. 2005	Ishiguro, H. (2005). Towards a new cross-interdisciplinary framework. In*CogSci Workshop Towards social Mechanisms of android science, Stresa*.
Joshi et al. 2005	Joshi, P., Tien, W. C., Desbrun, M., & Pighin, F. (2005, July). Learning controls for blend shape based realistic facial animation. In *ACM SIGGRAPH 2005 Courses* (p. 8). ACM.
Krill, Platek & Wathne 2008	Krill, A. L., Platek, S. M., & Wathne, K. (2008). Feelings of control during social exclusion are partly accounted for by empathizing personality. *Personality and individual differences*, *45*(7), 684-688.
Krill and Platek 2009	Krill, A., & Platek, S. M. (2009). In-group and out-group membership mediates anterior cingulate activation to social exclusion. *Frontiers in Evolutionary Neuroscience*, *1*.
Krumhuber et al. 2007	Krumhuber, E., Manstead, A. S., Cosker, D., Marshall, D., Rosin, P. L., & Kappas, A. (2007). Facial dynamics as indicators of trustworthiness and cooperative behavior. *Emotion*, *7*(4), 730.
S. Lee, G.J. Kim and J. Lee 2004	Lee, S., Kim, G. J., & Lee, J. (2004, November). Observing effects of attention on presence with fMRI. In *Proceedings of the ACM symposium on Virtual reality software and technology* (pp. 73-80). ACM.
Lelieveld et al. 2012	Lelieveld, G. J., Moor, B. G., Crone, E. A., Karremans, J. C., & van Beest, I. (2012). A penny for your pain? The financial compensation of social pain after exclusion. *Social Psychological and Personality Science*, *4*(2), 206-214.
Levitas et al 2007	Levitas, R., Pantazis, C., Fahmy, E., Gordon, D., Lloyd, E., & Patsios, D. (2007). The multi-dimensional analysis of social exclusion.
Loomis, Blascovich and Beall 1999	Loomis, J. M., Blascovich, J. J., & Beall, A. C. (1999). Immersive virtual environment technology as a basic research tool in psychology. *Behavior Research Methods, Instruments, & Computers*, *31*(4), 557-564.
MacDorman 2006	MacDorman, K. F. (2006, July). Subjective ratings of robot video clips

	for human likeness, familiarity, and eeriness: An exploration of the uncanny valley. In *ICCS/CogSci-2006 long symposium: Toward social mechanisms of android science* (pp. 26-29).
Magnenat-Thalmann, Zhang and Feng 2009	Magnenat-Thalmann, N., Zhang, J. J., & Feng, D. D. (Eds.). (2009). *Recent Advances in the 3D Physiological Human*. Springer.
Mania et al. 2010	Mania, K., Badariah, S., Coxon, M., & Watten, P. (2010). Cognitive transfer of spatial awareness states from immersive virtual environments to reality. *ACM Transactions on Applied Perception (TAP)*, *7*(2), 9.
Martin et al. 2006	Martin, J. C., Niewiadomski, R., Devillers, L., Buisine, S., & Pelachaud, C. (2006). Multimodal complex emotions: Gesture expressivity and blended facial expressions. *International Journal of Humanoid Robotics*, *3*(03), 269-291.
Masten et al 2011	Masten, C. L., Colich, N. L., Rudie, J. D., Bookheimer, S. Y., Eisenberger, N. I., & Dapretto, M. (2011). An fMRI investigation of responses to peer rejection in adolescents with autism spectrum disorders. *Developmental cognitive neuroscience*, *1*(3), 260-270.
Masten et al. 2009	Masten, C. L., Eisenberger, N. I., Borofsky, L. A., Pfeifer, J. H., McNealy, K., Mazziotta, J. C., & Dapretto, M. (2009). Neural correlates of social exclusion during adolescence: understanding the distress of peer rejection. *Social cognitive and affective neuroscience*, *4*(2), 143-157.
Masten, Morelli & Eisenberger 2011	Masten, C. L., Morelli, S. A., & Eisenberger, N. I. (2011). An fMRI investigation of empathy for 'social pain'and subsequent prosocial behavior. *Neuroimage*,*55*(1), 381-388.
Mathiak & Weber 2006	Mathiak, K., & Weber, R. (2006). Toward brain correlates of natural behavior: fMRI during violent video games. *Human brain mapping*, *27*(12), 948-956.
Maurage et al. 2012	Maurage, P., Joassin, F., Philippot, P., Heeren, A., Vermeulen, N., Mahau, P., ... & de Timary, P. (2012). Disrupted regulation of social exclusion in alcohol-dependence: an fMRI study. *Neuropsychopharmacology*, *37*(9), 2067-2075.
Mayr, Horleinsberger & Petta 2014	Mayr, S., Horleinsberger, W., & Petta, P. (2014, September). The Trauma Treatment Game: Design Constraints for Serious Games in Psychotherapy. In*Games and Virtual Worlds for Serious Applications (VS-GAMES), 2014 6th International Conference on* (pp. 1-6). IEEE.
McDonnell and Breidt 2010	McDonnell, R., & Breidt, M. (2010, December). Face reality: Investigating the uncanny valley for virtual faces. In *ACM SIGGRAPH ASIA 2010 Sketches* (p. 41). ACM.
McDonnell, Breidt and Bülthoff 2012	McDonnell, R., Breidt, M., & Bülthoff, H. H. (2012). Render me real?: investigating the effect of render style on the perception of animated virtual humans. *ACM Transactions on Graphics (TOG)*, *31*(4), 91.
McDonnell et al. 2009	McDonnell, R., Jörg, S., Hodgins, J. K., Newell, F., & O'sullivan, C. (2009). Evaluating the effect of motion and body shape on the perceived sex of virtual characters. *ACM Transactions on Applied Perception (TAP)*, *5*(4), 20.
McDonnell et al. 2008	McDonnell, R., Jörg, S., McHugh, J., Newell, F., & O'Sullivan, C. (2008, August). Evaluating the emotional content of human motions on real

	and virtual characters. In *Proceedings of the 5th symposium on Applied perception in graphics and visualization* (pp. 67-74). ACM.
McDonnell, Larkin, Hernández, Rudomin & O'Sullivan 2009	McDonnell, R., Larkin, M., Hernández, B., Rudomin, I., & O'Sullivan, C. (2009, July). Eye-catching crowds: saliency based selective variation. In *ACM Transactions on Graphics (TOG)* (Vol. 28, No. 3, p. 55). ACM.
Meyer et al. 2012	Meyer, M. L., Masten, C. L., Ma, Y., Wang, C., Shi, Z., Eisenberger, N. I., & Han, S. (2012). Empathy for the social suffering of friends and strangers recruits distinct patterns of brain activation. *Social cognitive and affective neuroscience*, nss019.
Meyer et al. 2015	Meyer, M. L., Masten, C. L., Ma, Y., Wang, C., Shi, Z., Eisenberger, N. I., ... & Han, S. (2015). Differential neural activation to friends and strangers links interdependence to empathy. *Culture and Brain*, *3*(1), 21-38.
Milgram 1963	Milgram, S. (1963). Behavioral study of obedience. *The Journal of Abnormal and Social Psychology*, *67*(4), 371.
Minato et al. 2005	Minato, T., Shimada, M., Itakura, S., Lee, K., & Ishiguro, H. (2005, July). Does gaze reveal the human likeness of an android?. In *Development and Learning, 2005. Proceedings. The 4th International Conference on* (pp. 106-111). IEEE.
Moor et al. 2012	Moor, B. G., Güroğlu, B., de Macks, Z. A. O., Rombouts, S. A., Van der Molen, M. W., & Crone, E. A. (2012). Social exclusion and punishment of excluders: neural correlates and developmental trajectories. *Neuroimage*, *59*(1), 708-717.
Mori 1970	Mori, M. (1970). Bukimi no tani [The uncanny valley], Energy,[Online] 7 (4), 33-35.
Nishino, Utsumiya & Korida 1998	Nishino, H., Utsumiya, K., & Korida, K. (1998, November). 3d object modeling using spatial and pictographic gestures. In *Proceedings of the ACM symposium on Virtual reality software and technology* (pp. 51-58). ACM.
Novembre, Zanon, and Silani 2014	Novembre, G., Zanon, M., & Silani, G. (2014). Empathy for social exclusion involves the sensory-discriminative component of pain: a within-subject fMRI study. *Social cognitive and affective neuroscience*, nsu038.
Nowak 2003	Nowak, K. L. (2003). Sex categorization in computer mediated communication (CMC): Exploring the utopian promise. *Media Psychology*, *5*(1), 83-103.
Nowak and Biocca 2003	Nowak, K. L., & Biocca, F. (2003). The effect of the agency and anthropomorphism on users' sense of telepresence, copresence, and social presence in virtual environments. *Presence*, *12*(5), 481-494.
Nowak & Rauh 2005	Nowak, K. L., & Rauh, C. (2005). The influence of the avatar on online perceptions of anthropomorphism, androgyny, credibility, homophily, and attraction. *Journal of Computer-Mediated Communication*, *11*(1), 153-178.
Nunez & Srinivasan 2006	Nunez, P.L., Srinivasan, R. (2006): Electric Fields of the Brain. Oxford University Press, New York.
Park and Hodgins 2006	Park, S. I., & Hodgins, J. K. (2006, July). Capturing and animating skin deformation in human motion. In *ACM Transactions on Graphics*

	(TOG) (Vol. 25, No. 3, pp. 881-889). ACM.
Parke and Waters 1996	Parke, F. I., and Keith Waters. "Computer Facial Animation, 1996." *AK Peters. Wellesley, MA*.
Parsons 2011	Parsons, T. D. (2011). Neuropsychological assessment using virtual environments: enhanced assessment technology for improved ecological validity. In *Advanced Computational Intelligence Paradigms in Healthcare 6. Virtual Reality in Psychotherapy, Rehabilitation, and Assessment* (pp. 271-289). Springer Berlin Heidelberg.
Perani et al. 2001	Perani, D., Fazio, F., Borghese, N. A., Tettamanti, M., Ferrari, S., Decety, J., & Gilardi, M. C. (2001). Different brain correlates for watching real and virtual hand actions. *Neuroimage*, *14*(3), 749-758.
Rivera et al. 2010	Rivera, F., Watten, P., Holroyd, P., Beacher, F., Mania, K., & Critchley, H. (2010, July). Real-time compositing framework for interactive stereo fMRI displays. In *ACM SIGGRAPH 2010 Posters* (p. 16). ACM.
Sagar 2006	Sagar, M. (2006, July). Facial performance capture and expressive translation for king kong. In *ACM SIGGRAPH 2006 Sketches* (p. 26). ACM.
Schkolne, Pruett & Schröder 2001	Schkolne, S., Pruett, M., & Schröder, P. (2001, March). Surface drawing: creating organic 3D shapes with the hand and tangible tools. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 261-268). ACM.
Seyama and Nagayama 2007	Seyama, J. I., & Nagayama, R. S. (2007). The uncanny valley: Effect of realism on the impression of artificial human faces. *Presence: Teleoperators and Virtual Environments*, *16*(4), 337-351.
Shapiro 1997	Shapiro, K. (1997). A phenomenological approach to the study of nonhuman animals.
Sifakis et al. 2005	Sifakis, E., Neverov, I., & Fedkiw, R. (2005, July). Automatic determination of facial muscle activations from sparse motion capture marker data. In *ACM Transactions on Graphics (TOG)* (Vol. 24, No. 3, pp. 417-425). ACM.
Sjölie et al 2010	Sjölie, D., Bodin, K., Elgh, E., Eriksson, J., Janlert, L. E., & Nyberg, L. (2010, April). Effects of interactivity and 3D-motion on mental rotation brain activity in an immersive virtual environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 869-878). ACM.
Slater et al. 2006	Slater, M., Antley, A., Davison, A., Swapp, D., Guger, C., Barker, C., ... & Sanchez-Vives, M. V. (2006). A virtual reprise of the Stanley Milgram obedience experiments. *PloS one*, *1*(1), e39.
Sun, Fox, and Bailenson 2012	Sun J. A., Fox J., & Bailenson J. N. (2012) *Leadership in Science and Technology: A Reference Handbook*, Chapter: 79 AVATARS.
Szita 2012	Szita, I. (2012). Reinforcement learning in games. In *Reinforcement Learning* (pp. 539-577). Springer Berlin Heidelberg.
Taubert et al. 2012	Taubert, N., Christensen, A., Endres, D., & Giese, M. A. (2012, August). Online simulation of emotional interactive behaviors with hierarchical Gaussian process dynamical models. In *Proceedings of the ACM Symposium on Applied Perception* (pp. 25-32). ACM.

Theobald et al. 2009	Theobald, B. J., Matthews, I., Mangini, M., Spies, J. R., Brick, T. R., Cohn, J. F., & Boker, S. M. (2009). Mapping and manipulating facial expression.*Language and speech*, *52*(2-3), 369-386.
Tost et al 2014	Tost, D., Pazzi, S., von Barnekow, A., Felix, E., Puricelli, S., & Bottiroli, S. (2014, May). SmartAgeing: a 3D serious game for early detection of mild cognitive impairments. In *Proceedings of the 8th International Conference on Pervasive Computing Technologies for Healthcare* (pp. 294-297). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
Aggeliki Tsoli 2014	Tsoli, A. (2014). *Modeling the Human Body in 3D: Data Registration and Human Shape Representation* (Doctoral dissertation, Brown University, Department of Computer Science Providence, RI, USA).
Walker and Walker 1997	Walker, A., & Walker, C. (Eds.). (1997). *Britain divided: The growth of social exclusion in the 1980s and 1990s* (p. 8). London: Cpag.
Wang, Sourina and Nguyen 2011	Wang, Q., Sourina, O., & Nguyen, M. K. (2011). Fractal dimension based neurofeedback in serious games. *The Visual Computer*, *27*(4), 299-309.
Watkins & Dayan 1992	Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.
Williams 1997	Williams, K. D. (1997). Social ostracism. In *Aversive interpersonal behaviors*(pp. 133-170). Springer US.
Williams 2001	Williams, K. D. (2001). Ostracism: The Power of Silence (pp. 7–11). *New York: Guilford*.
Williams and Jarvis 2006	Williams, K. D., & Jarvis, B. (2006). Cyberball: A program for use in research on interpersonal ostracism and acceptance. *Behavior research methods*, *38*(1), 174-180.
Williams & Zadro 2005	Williams, K. D., & Zadro, L. (2005). Ostracism: The Indiscriminate Early Detection System.
Williams, Cheung and Choi 2000	Williams, K. D., Cheung, C. K., & Choi, W. (2000). Cyberostracism: effects of being ignored over the Internet. *Journal of personality and social psychology*,*79*(5), 748.
Williams, Forgas & von Hippel 2005	Williams, K. D., Forgas, J. P., & Von Hippel, W. (Eds.). (2005). *The social outcast: Ostracism, social exclusion, rejection, and bullying*. Psychology Press.
Witmer and Singer 1998	Witmer, B. G., & Singer, M. J. (1998). Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoperators and virtual environments*, *7*(3), 225-240.
Yoshida et al. 2010	Yoshida, W., Seymour, B., Friston, K. J., & Dolan, R. J. (2010). Neural mechanisms of belief inference during cooperative games. *The Journal of Neuroscience*, *30*(32), 10744-10751.
Zeleznik, Herndon & Hughes 1996	Zeleznik, R.C., Herndon, K., & Hughes, J. (1996). SKETCH: an interface for sketching 3D scenes. SIGGRAPH. p. 163-170.
Zhang et al. 2013	Zhang, Y., Han, T., Ren, Z., Umetani, N., Tong, X., Liu, Y., ... & Cao, X. (2013, October). BodyAvatar: Creating freeform 3D Avatars using first-person body gestures. In *Proceedings of the 26th annual ACM symposium on User interface software and technology* (pp. 387-396).

	ACM.
3D face scanning using the Kinect	http://kinecthacks.net/3d-facescanning-using-the-kinect.
3D Model Repositories	https://charactergenerator.autodesk.com/
Fastmocap technology	http://www.npasja.net/modern-3d-motion-capture-from-fastmocap-technology/
Hank Virtual Environments Lab	http://psychology.uiowa.edu/hank-virtual-environments-lab
Torque 3D	http://torque3d.org/
Unity 3D	http://unity3d.com/
Unreal Development Kit	http://www.unrealengine.com/udk/
Virtual - Storytelling Experiment	https://mrl.nyu.edu/~perlin/experiments/virtual-storytelling/