Motion Capture Shooter

An upper body movement controlled first person shooter that implements motion capture hardware.
Summary

This application is the result of a year of user-centred and field research and aims to produce a physically interactive user experience that involves the use of upper body movements as a means of controlling a virtual character. The game uses a motion capture suit that is provided through the University’s joint venture with Animazoo, a world-leading motion capture hardware developer. It is built on the new generation Unreal 3 technology, an advanced engine used by hugely popular games. This engine and its software development tools are available in a package entitled “Unreal Development Kit”, released by Epic Games in November 2009. The development of the application involved working with the University's eMove research department and is partly based on previous work and research they did on motion capture.

This application can be played traditionally or through gesture-based motion recognition. The implemented code translates the player’s movements into virtual events. The motion capture had to be implemented while taking into account the complexity of the first person shooter's standard control scheme. The resulted application allows the player to control the virtual character (walk, rotate, strafe, jump, use complicated weapons) through upper body gestures. The gesture recognition concept and control scheme is unique and was designed for this game.

The most part in the development of this game was the creation of numerous 3d assets used to detail the virtual environment. It implements a set of custom weapons designed and built uniquely for the purpose of this project. This project also involved the implementation of artificial intelligence, particle systems, physics systems and sounds. Photoshop, 3d Studio Max (with UDK plugins), VisualStudio (with UnrealScript plugins), Audacity, Unreal Development Kit tools and other applications were needed to complete the various components of the game.

This application is programmed in UnrealScript, a programming language based on C, developed by Epic Games for this engine. A large part of this project involved researching how the engine development kit works, understanding the fundamentals of the engine and learning a new programming language.

UDK is based on Unreal Tournament 3, a futuristic, fantastically-themed first person shooter also developed by Epic Games. The game's overall theme is very different from that of the original. Many modifications and additions were made to the engine.

This project is in accordance with the supervisor’s and university’s interests and may be used to showcase Animazoo’s hardware. The game is provided on a CD and can be installed by following the instructions found in the appendix of this report. Additionally, all textures, models and source code have been included in this submission.
Acknowledgements

I would like to thank the following people for their help and assistance in completing this project:

**Supervisor:**
Dr. Martin White for his weekly advice that constantly pointed me in the right direction.

**eMove Team:**
Jake Slack and Cash Garman for their help with integrating the motion capture suit in the game.

**Epic Games Forum:**
“Geodav” and all the other forums members that make the Epic Forums a helpful resource for developing an Unreal 3 based game.
Statement of Originality

This report is submitted as a requirement for the undergraduate degree of BSc Computer Science at the University of Sussex. I hereby declare that all information presented in this piece of writing is the result of my own labour. I also declare that all intellectual content, technical specifications and design ideas are the product of my own work unless otherwise stated.

Signed: ........................................................................................................

Dated: ........................................................................................................
# Table of Contents

- Title Page
- Summary
- Acknowledgements
- Statement of Originality
- 1. Introduction
  - 1.1 Problem Statement
  - 1.2 Background Research
    - 1.2.1 Gaming Development Stages
    - 1.2.2 Gaming Engines
    - 1.2.3 Motion Capture
    - 1.2.4 Modding
    - 1.2.5 Level Creation Techniques
  - 1.3 Professional Considerations
    - 1.3.1 Code of Conduct
    - 1.3.2 Code of Good Practice
- 2. Requirements
  - 2.1 Requirement Gathering Techniques
  - 2.2 Specification:
    - 2.2.1 Primary Requirements
    - 2.2.2 Extensions
- 3. Design
  - 3.1 Game Design Overview
  - 3.2 Time Allocation
  - 3.3 Use Case Diagram
    - 3.3.1 Primary Course of Events
    - 3.3.2 Alternative Course of Events
  - 3.4 Motion Capture Suit
  - 3.5 Theme and Style
  - 3.6 The Unreal 3 Engine Pipeline
  - 3.7 Activity Diagram
  - 3.8 Control Scheme
- 4. Implementation
  - 4.1 Weapons
    - 4.1.1 Overview
    - 4.1.2 Texturing
    - 4.1.3 Rigging and Importing
    - 4.1.4 First Person Animations
    - 4.1.5 Pickup Classes
  - 4.2 First Person Hands
    - 4.2.1 Modelling
    - 4.2.2 Texturing
    - 4.2.3 Rigging
    - 4.2.4 Animation

---

Dissertation - Tudor Pascu - 2010 - Candidate Number 10356
8.10.2 Motion Capture Control Class Using Hand Distances and Buttons
8.10.3 Button Trigger Class
8.10.4 Weapon Classes
8.10.5 Weapon Relocation Classes
8.10.6 Third Person Weapon Classes
8.10.7 Weapon Ammo Classes
8.10.8 User Interface Configuration File
8.10.9 Weapon Recognition Configuration File
8.11 Screenshot Walkthrough
1. Introduction
1.1 Problem Statement

This project is centred on the design and implementation of a computer game that may work with motion capture technology provided through the university’s partnership with Animazoo. Animazoo is a world-leading motion capture hardware developer. The game should implement a control scheme where the keyboard and mouse inputs are replaced by upper body gestures. The first person shooter genre was chosen as it is the most immersive and popular type of computer action game.

A first person shooter, as the name would imply, allows the user to adopt a first-person point of view where her or she will see the game story unveil through the eyes of the virtual character. The first game of this genre that had international success dates back to 1992 when id Software released Wolfenstein 3D. This simple yet revolutionary game has set the standards for all first person shooter to follow. The concept behind Wolfenstein 3D was to experience the events from the perspective of a gun-slinging protagonist that is absorbed into an unknown and mysterious action-packed environment. As a result, a shooter themed game is now defined by the fight-for-survival premise where a player makes will use of a variety of weapons to progress through a level. This genre is the most sought after in action gaming and there have been many cases of success such as Activision's Modern Warfare 2 which earned 310 million dollars in sales during its first 24 hours of availability. The first person shooter genre is a reasonable choice.

Motion capture has been used in games for many years as a way of realistically animating characters. Only recently some have attempted to produce motion capture controlled games, an undertaking that regarded by many as the future of gaming. Nitendo showed that this was possible in 2007 with the release of the Wii, an instantly popular and affordable console system. The Wii’s motion capture capabilities are very limiting and as a result only simplistic were ever made for it.

The problem in motion gaming is that of hardware limitations and no games of mass-appeal genres have been produced. No affordable full-body motion capture system is accurate enough to control a proper game character. This project attempts to combine a massively popular computer gaming genre with motion capture technology to conclude an interesting, intuitive and insightful product. The motion capture capability of the game may create an immersive and unique gaming experience for any first person shooter enthusiast.
1.2 Background Research

1.2.1 Gaming Development Stages

The core aim of this project is to produce a mass-appeal genre game that works with motion capture. Because of this, the game should be similar to and closely follow the same development stages that other games do. Thousands of first person shooters have been made to date and understanding their development process is vital in concluding the scope of this project given the time budget.

![Diagram](image)

*Fig 1. The gaming development lifecycle from New Riders Games’ “Level Design for Games”.*

The various tasks described above are divided between different departments in the company. This model works in a professional and well-structured game development corporation and is not entirely relevant in the creation of this project. This project will be built by only one individual and its success lies in time management.
1.2.2 Gaming Engines

Few game developers build engines and it is quite common to license technology in order to build a game. A game engine can take up to 5 years to develop, five years that most game developers cannot afford. Building solid engine architectures is a large undertaking and few companies have the knowledgebase or the time to do so. As a result, only large game developers such as id Software (creators of Doom and Quake) and Epic Games (creators of the Unreal games, Gears of War etc.) invest money and time in creating development kits for others to use. The engines are then licensed and used by small companies that have the freedom to modify them to suit their projects.

A vital part in the development of a game is choosing the engine framework. It will represent the foundation and will influence the development of any game content. In making this decision, one must carefully analyze a project’s requirements and consider all development stages. For example, choosing the Unreal Engine to produce a simple game would be a mistake as it requires powerful hardware and a high level of detail in all 3d content. This is due to the fact that the Unreal Engine is a powerful hardware power-consuming tool. Similarly, it is a mistake to choose a simple engine such as Panda3D to create a hugely detailed game as the engine will not cope with highly detailed models.

For the purpose of this project, three engines were considered: Panda 3D, Unity and Unreal 3. Unity and Unreal 3 were compared, tested and contrasted as follows:

The Unreal Development Kit is a hugely popular engine, used in the creation of many modern generation games. It is a professional development framework that can be used in projects requiring advanced visualization. It is arguably the most powerful engine in the industry.

General Beneficiaries

- Fully integrated game editing environment.
- Unreal Gemini multithreaded rendering system will support highly detailed models.
- A material editor tool that can be used to create complicated materials and shaders.
- A powerful dynamic lighting system featuring global illumination that can be used to create realistically lit environments.
- Solid physics system that can be used to assign weights and properties to objects.
- Kismet, a powerful programming tool that can be used to create interactive environments and gameplay.
- Animation editor that facilitates the importing of animations from several 3D Modelling and Animation tools.
- Unreal Matinee editor, used for animating level elements.
- UnrealScript, a C-based programming language.
• Internet/LAN support.
• Basic AI bot system that can be used to create a single player mode.
• Visual Sound Cue Editor that facilitates the importing of sounds and supports various compression schemes.
• Particle effects editor that facilitates the implementation of muzzleflashes, impact effects and explosions.
• Online community support.

Personal Beneficiaries

• UnrealScript is an object-oriented programming language, similar to C and Java.
• Satisfies the determination to create highly detailed models.
• Previous experience with 3D Studio Max, a tool supported by UDK.
• Previous experience with Unreal engines and development tools.

Drawbacks

• Requires a high level of detail. The asset creation development stage is time consuming and given the time frame for this project may cause problems.
• Hardware-demanding
• Complicated interface (for beginners).
• Lack of beginner/introductory tutorials.

Unity is a multiplatform engine that comes with a user friendly set of development tools. It is an affordable engine architecture used by many phone applications. It can be used as the foundation for a game of any genre.

General Beneficiaries

• Simple editor interface.
• Direct3D 9 and OpenGL support, making the Unity’s graphics system cross-platform compatible.
• Static lighting system.
• Easy 3D asset and texturing importing.
• Simplistic and easy-to-use particle system.
• Optimized texture compression.
• Pre-built shaders library including useful elements such as water.
• Terrain editor.
• Real-time texture painting.
• Asset library including vegetation and other useful game objects.
• Realtime networking and browser support.
• PhysX system support allowing for the allocation of physics properties to game objects.
• Prebuilt ragdoll and vehicle systems.
• Various audio compression types supported.
• JavaScript, C# and Boo (a dialect of Python) programming languages supported.
• Good library of tutorials and user documentation.

Personal Beneficiaries

• Previous experience in JavaScript and C#.
• Previous experience with 3D Studio Max, a tool supported by Unity.

Drawbacks

• Limited graphics.
• Indy version of Unity does not have dynamic lighting.
• No pre-built first person shooter mode.
• Various bugs and problems with the engine’s physics.

Given that both engines are available for free use, deciding which one to use involved understanding the scope of this project and concluding the system requirements. Part of this project’s goal is to create a visually pleasing product that displays highly detailed 3D artwork. A large stage in the development of this game is the 3D modelling and texturing. As a result, the Unreal Development Kit was chosen. Unity is very limiting as the independent free version of the engine does not support dynamic lighting. To make a believable environment, dynamic lighting is crucial and the Unreal Development Kit also supports Global Illumination. This decision was also based on previous work I have produced for past versions of the Unreal Engine. Furthermore, given that the genre of this game will be a first person shooter, UDK comes with a pre-built FPS mode that will be used in the final product.
1.2.3 Motion Capture

Motion capture gaming has never been possible due to hardware limitations. A large amount of data must be processed by a computer to implement a full-body motion capture system. However, in recent years, with large increases in hardware power, there is worldwide interest in motion capture technology. An example would be the release of Nintendo’s Wii and more recently the Wii Motion+, extremely successful gaming console. The Wii’s success culminated in Microsoft entering the arena with Project Natal due for release in fall 2010 for the XBox. Several handheld devices, such as the IPhone, have started using basic motion capture gyroscope/accelerometer hardware. This technology is turning out to be very profitable and various economic opportunities associated with this so-called ‘motion gaming’ have arisen. There is a gap in the market for low cost/consumer level motion capture peripherals such as that being developed by Animazoo. Animazoo, a world-leading motion capture hardware provider, currently supplying the eMove department with motion capture peripherals, are in the process of developing such consumer level motion capture hardware.

There are several motion capture approaches that vary in price and complexity:

- **Optical Systems:** Range in costs from the cheap, simple and inaccurate systems like Microsoft’s XBox-based Project Natal to the more costly, complicated and accurate system like Vicon’s multiple-camera motion detection. Optical systems use a camera and process the images being recorded by that camera. This can be done in three ways: using passive, using active markers or marker-less systems. The last of the three is much more hardware demanding as it uses Computer Vision principles.

- **Exoskeleton Mechanical Systems:** Animazoo is developing exoskeleton mechanical systems that measure the various angles of joints corresponding to the user’s body. Animazoo’s system is mixed with inertial components such as gyroscopes/accelerometers.

- **Magnetic motion capture systems** that calculate one’s position and orientation through relative magnetic fields between transmitters and receivers.

- **Full inertial motion capture systems** that are based on miniature and relatively costly inertial measurement units. Such a system works with a three dimensional gyroscopes/accelerometer that can detect yaw, pitch and roll. If calibrated appropriately, an inertial motion capture system can be very accurate.

Following discussions with the supervisor, research has shown the potential of motion capture technology being implemented in other computer-unrelated areas. These are areas of interest in the Interactive Systems: Computer Graphics Group’s research topics:

- theme park arcades
- advertising
- sports therapy
- digital heritage
- psychobiology
- stroke therapy
1.2.4 Modding

To fully understand what it takes to develop a game fan-made games, also known as “mods” or “modifications”, were analyzed. Essentially, mods are based on existing games and make use of existing game engines but implement a set of entirely new content and modified gameplay mechanics. The following three games are first person shooters and are three very different success cases.

Resistance and Liberation is a World War 2 themed first person shooter. It focuses on realism and historical accuracy to interest the history enthusiasts. This project was started in 2002 and it was designed to work on Valve’s Half Life 2 Source engine. As of now, the game has not been released and has been stuck in its final development stages for some time. The game is very well constructed and designed. Much concept artwork has been created by talented artists. The time allocation in the development of this game was poor and as a result, the developers have lost motivation to work on it. 8 years later, it is too late to release this game, and it shows the importance of time management in such an undertaking.

Red Orchestra is another World War 2 themed first person shooter based on the Unreal 2003 engine. It is a team-based multiplayer experience that takes place in a historically accurate theatre of war. It is in the most realistic WWII first-person multi-player combat to date on the PC. Because of its massive online success and its strong fan base, in 2006 game publisher Valve bought the rights to Red Orchestra. The fan-made game became a profitable, venture led by one of the biggest game developing and publishing companies in the world.
Point of Existence is a modern warfare themed vehicle-based first person shooter that uses the Battlefield 2 engine. This mod had many releases, each of which had many bugs and issues. The development team rushed some of its development stages and as a result, the game was never successful.

The first game of the three is competently built and very well premeditated, though it was never released. The second, similar to the first, was released and became successful money-making project. The third game is somewhere in the middle. It was released though it generated no profit and can be considered a partial failure. Understanding where others have failed or succeeded has helped determine what is and what isn’t important in developing a PC game. Again, time management is vital.

1.2.5 Level Creation Techniques

The level design of a game is one of the most time consuming stages in its development. “The level designer plays a key role in the overall process of game development.” (New Rider’s Level Design For Games p.1) Level building is unique to every game engine and involves the understanding of a set of tools unique to that game. However, there are level creation techniques and principles that apply to all games.

The Unreal Development Kit uses two level creation techniques, additive and subtractive modelling. A level has a foundation, a simple mesh that must be created in the level editor. This mesh needs to be sculpted using level design tools.

Additive

Additive level modelling is very similar to 3d modelling. The level design program provides a tool called a “brush”. The brush is a box that can be positioned and resized in the empty 3D space. It is then used to generate level mesh components such as walls, floors, stairs etc. It is often considered a primitive and restrictive method of 3D modelling.
Advantages:

- Similar to 3D Modelling.
- Easy to create outside environments that use sky domes.
- Can be converted to subtractive.
- Better for games that use skylights and other world properties requiring an outside environment.
- More efficient in compiling lighting.
- Better for games that use a lot of static meshes.

Disadvantages:

- Less efficient for creating indoor levels.

Subtractive

Subtractive level modelling is more commonly used in older games. Older games tend to use indoor levels as they are much less hardware demanding and simpler. Subtractive level modelling is done by carving out rooms from a solid 3D environment. It is a much quicker way of creating the mesh for building structures. For example, to create a room, only one brush is needed to empty out the room section. With additive level modelling, each wall has to be created independently.

Advantages

- Rapid and efficient method of creating indoor levels.
- A quick way to create rooms.

Disadvantages:

- Cannot build outside environments with a sky domes.
- Less efficient at compiling lighting.
- Fewer similarities with 3D Modelling.
1.3 Professional Considerations

1.3.1 Code of Conduct

In the development of this project all ethical issues that may apply to the development of this game have been taken into consideration. In Britain, the ethical standards involved in any computing project are defined by the BCS Code of Conduct and the Code of Practice. Therefore, those relevant have been discussed:

The Public Interest

BCS Code of Conduct 1 - “In your professional role you shall have regard for the public health, safety and environment.”

It is important to take into account the health and safety of the user considering that this project will involve body movements and gestures as a means of interacting with the game. The game will feature an instructive page and/or manual so that the user is aware of the physical actions required to control the game character prior to playing the game.

BCS Code of Conduct 2 - “You shall have regard to the legitimate rights of third parties.”

This project gives consideration to all third parties from which content and/or research has been gathered by correctly and clearly referencing those in question.

BCS Code of Conduct 3 - "You shall ensure that within your professional field/s you have knowledge and understanding of relevant legislation, regulations and standards, and that you comply with such requirements."

The project will use the independent free version of Epic Games’ Unreal Development Kit game engine and will comply with all legislation.

Duty to Relevant Authority

BCS Code of Conduct 8 - “You shall not disclose or authorise to be disclosed, or use for personal gain or to benefit a third party, confidential information except with the permission of your relevant authority.”

I am aware of the fact that my project is the property of the University of Sussex. The project will make use of certain technology that is exclusive to this institution and I will not, under any circumstances, release anything to the public without the consent of my supervisor and/or university.
Duty to the Profession

BSC Code of Conduct 11 - “You shall act with integrity in your relationships with all members of the BCS and with members of other professions with whom you work in a professional capacity.”

The implementation of the motion capture suit will involve briefly working together with the eMove team in a professional environment and I will interact with all in questions efficiently, proficiently and with integrity.

Professional Competence and Integrity

BSC Code of Conduct 14 - “You shall seek to upgrade your professional knowledge and skill, and shall maintain awareness of technological developments, procedures and standards which are relevant to your field.”

As the development period of this project is six months in which one or several third party and/or drivers may be updated, the project must take into consideration and comply with all those updates.

BSC Code of Conduct 15 - “Professional Competence and Integrity. You shall only offer to do work or provide a service that is within your professional competence.”

I have chosen this dissertation topic knowing what the required skill set is while being prepared to acquire any additional skills that may be required in completing the project. I consider this project to be within my professional competence.

1.3.2 Code of Good Practice

Code of Good Practice 2 - “Report any overruns to budget or timescales as they become apparent; do not assume that you will be able to recover them later.”

Because this project involves the creation of a large number of detailed 3d assets and consists of several other uniquely challenging stages, the question of budget and timescale arises. This project attempts to follow a set of self-imposed milestones that, if followed carefully, the project will meet the final deadline.
2. Requirements
2.1 Requirement Gathering Techniques

The requirements gathering stage of a project’s development is a key stage in its development. Before starting to develop an application, one must know and be certain of what that application is required to do. Frequent discussion between the developers and stakeholders must commence long before the first line of code is written. For this particular project, the stakeholder is the supervisor as well as the members of the eMove research team. Whether or not this project is relevant to their research is important regardless of whether the final release will directly be of use to them. The requirements gathering process began at a very early stage, prior to any project proposal. The requirements were established in weekly meetings with the supervisor and occasional meetings with the research team.

The meetings log can be seen in section 8.8 of the appendix.

There are many requirement gathering techniques, each of which having a distinct purpose. Here are some that were useful in concluding the requirements for this project:

1. **Document Analysis** – understand where others succeeded or failed. Document analysis is done during the research stages of a project.
2. **Focus Group/Brainstorming** – an effective way devising new ideas and concepts by discussing them with others.
3. **Interview** – important in concluding the stakeholder’s interests.
4. **Prototyping** – testing or playing-out various ideas.
5. **Observation/Player Testing** – evaluating your prototype and detecting problem areas.
6. **Survey** – a method for collecting quantitative information that may be useful in making important design decisions.
7. **Reverse Engineering** – understanding how the Unreal Development Kit works and what existing code can be used requires some reverse engineering, essentially taking the game apart. To define the requirements one must understand what the technology can do.

A focus group was organized and took place on the 29th of November where several Informatics students and friends gathered to discuss the fundamentals behind this project. The purpose of this meeting was to observe a group’s opinion on various topics. The focus group was based on a small presentation made in front of the attendees to show and explain the project. Many ideas were debated and new ideas arisen.

The focus group topics, questions and answers can be seen in section 8.9 of the appendix.
2.2 Requirements Specification

2.2.1 Primary Requirements

R1: Motion Capture Suit

R1.1 - Input in the game data from the motion capture suit peripheral into the game engine in the form of angles and/or 3d coordinates.

R1.2 - Provide a set of ingame console commands that can be used to calibrate/modify the suit’s settings.

R1.3 - Provide simple suit calibration on start-up.

R1.4 - Allow for modification of gesture detection sensitivities.

R2: Motion Capture Player Interactivity

R2.1 - Control the main player model.

R2.2 - Walk through the 3D environment using upper torso gestures.

R2.3 - Rotate and look throughout the 3D environment by pointing with the right arm (replacing the mouse scrolling with arm positioning).

R2.4 - Jump by rapidly raising the left arm.

R2.5 - Select from a set of three weapons by reaching for body parts that correspond to weapon holsters. Each weapon must have a suitable in-game animation showing the weapon being lowered, replaced and raised.

R2.6 - Fire a weapon by squeezing the motion capture suit mounted trigger. Each weapon must have a suitable in-game animation of gun recoil, empty cartridge disposal. Each muzzle flash will display that in the game.

R2.7 - Each weapon must have an alternative melee attack triggered either through rapid swings of the gun-holding arm (imitating a punch) or by a suit mounted button. Each weapon must have a suitable in-game animation to display this action.
R3: Game Mode Requirements

R3.1 - Provide an underlying theme that will be discovered during the player’s progression through the level.

R3.2 - Include a set of immediate threats and/or challenges in the form of AI-controlled enemies.

R3.3 - Include a checkpoint/objective or scoring system that will keep track of the player’s progression through the game.

R4: Environment Requirements

R4.1 - Provide a short, detailed and themed level.

R4.2 - Contain a set of static models that are representative of the overall game theme and add detail to the level.

R4.3 - Contain a set of realistic textures that are representative of the overall game theme.

R5: User Interface

R5.1 - Implement UDK’s menu system.

   R5.1.1 - Menu system interface must be able to access both standard UDK maps and the extended game content.

R5.2 - Include a comprehensible HUD (heads up display) containing relevant information in completing the game.

   R5.2.1 - HUD must display a health percentage and ammo count.

R6: Ease of Learning

R6.1 - Because the game will use a unique set of controls based on upper body movements, those controls must be intuitive and easy to discover.

R6.2 - Package must include an installation manual.

   R6.2.1 - All instructions must be made available to read before the game has been started so that the user is aware of the physical requirements.

R7: Performance

R7.1 - Game must be developed while taking into consideration the hardware on which it will run.

R7.2 - The assets’ level of detail (polygon count) must be optimized so that the game runs at a high and constant frame rate.
R7.3 - The game must be tested thoroughly and all possible problem areas fixed before the final release.

**R8: Installation**

R8.1 - The game must be presented as an add-on/expansion to the Unreal Development Kit.

R8.2 - Provide clear installation instructions.

## 2.2.2 Extensions

**R1: Motion Capture Player Interactivity**

R1.1 - Provide animated first person hands that are visible only in first person and interact with the weapon animations.

R1.2 - Provide several enemy character models to diversify the gameplay.

R1.3 - Create a fourth melee weapon that will make further use of the motion capture capabilities of the suit.

**R2: Environment**

R3.1 - Make use of the Unreal Development Kit fractal map system to create an interactive and destructible environment that the player can affect.

R3.2 - Expand the library of level assets and further detail areas of the level.

R3.3 - Add physics objects that react to the player’s movement and to the guns being fired.

**R3: User Interface**

R4.1 - Create a section in the heads up display that displays a picture useful in understanding how to switch between weapons by reaching for different body parts.

**R4: Sound**

R5.1 - Provide a basic audio to the game including weapons and character sounds. The game may use of carefully chosen standard Unreal Development Kit sounds as well as new material.
3. Design
3.1 Game Design Overview

The design of an application is very important in concluding an implementation approach that satisfied the needs of the client. Game development differs from general software development as the design team must fully understand the technology used and the features it supports. Taking into account that gaming development kits, such as UDK, come with a long list supported features, the design team must find creative ways of using the full potential of the technology in question. Game engines are hardware-demanding and built in an extremely code-efficient manner. Engine developers invest a lot of money and time in optimizing and testing their product. It is very time consuming and risky to change any of the engine’s core elements and game designers are often limited by the engine’s capabilities.

In large game development companies, the design of a game is divided between several teams, each having a unique task. The system design team focuses on the technical high level design and defines a structured way of developing the game. The game design team chooses the artistic direction that the game will follow and focuses on creating: concept art, level layouts, gameplay modes etc.

A successful design will aid the programmers and content creators throughout all stages of implementation. To reach this project’s goal and to meet the deadline, a clear and straightforward design was concluded early. It comprises of the motion capture control scheme, overall game theme and a gameplay specifics, all of which are discussed in the next few sections.

3.2 Time Allocation

The design of a piece of software should always take into account the time frame in which the product is to be developed released. Designs tend to be overly ambitious and unrealistic. As a result, more than half of game developers fail to meet their deadlines. After missing deadlines, many games remain in their final development stages and due to lack of funding and are never released. The most famous example is Duke Nukem Forever, a first person shooter that has been in development since 1997 and was never released. This was due to poor time management, overly ambitious design and subsequently led to conflicts within the development teams.

Time allocation is discussed in the project plan in section of 8.4 the appendix.
3.3 Use Case Diagram

3.3.1 Primary Course of Events

The primary course of events where the user loads the game and plays it with the default control calibrations:

![Use Case Diagram](image)

3.3.2 Alternative Course of Events

The alternative course of events where the user loads the game calibrates the controls before playing:

![Use Case Diagram](image)
3.4 Motion Capture Suit

This project will make use of the GYPSY5 motion capture suit provided by Animazoo. This prototype uses a set of experimental sensors that allow for accurate detection of upper body movements. The majority of the sensors correspond to upper body joint rotations. Additional gyroscope/accelerometers ensure that the player’s world position is known. The drivers for the suit and some UnrealScript code to pass data from those drivers into UDK was provided by eMove and Animazoo. The purpose of this project is to link the received stream of data to the player controls efficiently.

3.5 Theme and Style

The overall theme of the game was chosen while taking into account the quantity and type of content that needs to be developed. The game’s theme has to be different from that of the original and its unique content has to be used effectively. Modelling level assets is very lengthy and taking account the motion capture stage of the development process, the content creation stage had to be made as simple as possible without sacrificing quality.

The final decision was that the game should be set in the present. Bringing the futuristic and fantastical Unreal Tournament 3 into the present time was to be the main task in creating the content for this game. An office building was chosen for the level design as office buildings have many instances of the same object such as chairs, computers, desks and tables. The inspiration for that came from a map in Counter Strike entitled cs_office:

![Image of an office setting](image_url)
3.6 The Unreal 3 Engine Pipeline

Unreal Engine 3 is a development framework that implements a C based programming language entitled UnrealScript. Aside from the many development tools that come with this development kit, UDK provides this expandable programming language. The engine itself is very advanced and the task of creating a game using it is should only involve low level programming of the content. To expand the game framework, a lot of time must be invested in understanding the core code of the engine.

UnrealScript is object-oriented and relies heavily on class inheritance. Content is programmed using sets of classes that extend each other in long chains.

![Class Hierarchy Diagram](image)

By looking at the above diagram it is easy to see that the Unreal 3 engine relies on a set of fundamental classes, a set of building blocks such as the Object class. These building blocks get chiselled and shaped into specific game element through the classes that extend them. It is vital to understand how the Unreal 3 engine is structured in order to add new content such as weapons or a motion capture controlled player.
3.7 Activity Diagram

Fig 8 Activity diagram showing the game’s functionality.
3.8 Control Scheme

In a first person shooter, input is sent to the game either by moving the mouse or pressing keyboard buttons. A motion capture control scheme has to replace the keyboard and mouse through upper body gestures. There are two ways of inputting motion capture data into a game. The first is through bone mapping, essentially mapping the player’s body movements to those of the virtual character. The second is to have a person’s movements trigger game events. Game events range from simple walking movements to more complicated weapon usage. Because of the complexity of the control scheme of a first person shooter, the former motion capture control scheme is not feasible. The player needs to perform many game events that cannot be done through simple bone mapping. As a result, a more complex gesture-based control scheme was designed to trigger game events that are simulated through ingame animations. The following table shows how these events are triggered by upper body movements (in the first out of two motion capture implementations attempts):

<table>
<thead>
<tr>
<th>Game Event</th>
<th>Motion Capture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move Forward</td>
<td>Lean forward.</td>
</tr>
<tr>
<td>Move Backward</td>
<td>Lean backward.</td>
</tr>
<tr>
<td>Strafe Left</td>
<td>Lean left.</td>
</tr>
<tr>
<td>Strafe Right</td>
<td>Lean right.</td>
</tr>
<tr>
<td>Turn Left</td>
<td>Move right hand left.</td>
</tr>
<tr>
<td>Turn Right</td>
<td>Move right hand right.</td>
</tr>
<tr>
<td>Look Up</td>
<td>Move right hand above the up.</td>
</tr>
<tr>
<td>Look Down</td>
<td>Move right hand down.</td>
</tr>
<tr>
<td>Jump</td>
<td>Move left hand above neck height.</td>
</tr>
<tr>
<td>Crouch</td>
<td>Move left hand below knee height.</td>
</tr>
<tr>
<td>Fire</td>
<td>Right joystick trigger.</td>
</tr>
<tr>
<td>Alternative Fire</td>
<td>Right joystick button.</td>
</tr>
<tr>
<td>Select Weapon 1</td>
<td>Press right joystick button and reach for weapon position 1.</td>
</tr>
<tr>
<td>Select Weapon 2</td>
<td>Press right joystick button and reach for weapon position 2.</td>
</tr>
<tr>
<td>Select Weapon 3</td>
<td>Press right joystick button and reach for weapon position 3.</td>
</tr>
<tr>
<td>Select Weapon 4</td>
<td>Press right joystick button and reach for weapon position 4.</td>
</tr>
</tbody>
</table>

Dissertation - Tudor Pascu - 2010 - Candidate Number 10356
4. Implementation
4.1 Weapons

4.1.1 Overview

The core element of any first person shooter is its weapon system. For the purpose of this project four weapon classes were implemented and are based on the Unreal Tournament 3 ShockRifle. Each weapon belonging to a class has two firing modes, a primary mode and a secondary melee attack. This means that each weapon can fire ammunition or be used to hit an enemy.

The first weapon class is slightly different to the others as it is a melee-only class and consists of a wrench that is swung to inflict damage. The wrench uses no ammo and should be used only when all the ammo for the other weapons is depleted. The monkey wrench does have primary and secondary fire modes. The primary fire is a quick hit that inflicts little damage while the secondary fire is a longer hit that inflicts more damage and should be used with precision. The model is based on pictures of a standard heavy duty monkey wrench.

The second weapon class is a light firearm based on the Desert Eagle handgun. The Desert Eagle (or “Deagle”) is a powerful pistol that, if used with precision, can inflict large amounts of damage. This weapon has a slow fire rate, a small clip and is inferior to all the other firearms present in the game. It has a secondary melee mode and can be used to hit an enemy. The game model was built based on reference pictures and blueprints and is accurate.

The third weapon class is an automatic firearm that has a high rate of fire. The weapon is based on the Mac 10, a small machinegun. It has a small calibre 30 bullet clip and is not very accurate. However, because it can shoot huge amounts of bullets, it is much more powerful than the handgun class. The Mac 10 has a simple yet very unique boxy shape and resembles the famous UZI machinegun. The game model is accurate though it has no extendable handle.

Lastly, the fourth weapon class is the most powerful. It is a pump action shotgun, designed to fit with the game’s overall theme. This weapon is based on the Franchi SPAS 12, an Italian combat shotgun. This design was chosen as the SPAS 12 does not have an extendable handle. If the shotgun did have a handle, the player model would need an individual animation for it so that it is held against the player’s shoulder. The overall design of the shotgun is simplistic and gives the impression of worn-down weapon.
More details on the weapon implementations can be seen in section 8.11.2-8.11.5 of the appendix.

4.1.2 Texturing

Texturing is done using UV unwrapping. It is a technique that involves taking the polygons of the 3D model and laying them out on a 2D surface so that they can be painted. The 2D surface becomes the texture file. Games nowadays use huge textures (2048 by 2048) and this game is no exception. To make the task of texturing easier, the textures are 1024 by 1024, similar to the standard unreal weapons. To make the texture as efficient as possible, a “sandwiching” technique was used where the same part of the texture can be projected on several polygons. Essentially, it is much more efficient to texture half of a gun’s barrel and have that part mirrored onto the other half. It shortens the process of painting the texture and allows for better and higher resolution textures. Unreal generally does not sandwich polygon projections. This is due to lightmaps, transparent shadow textures that are projected on top of the texture. For the purpose of this project, I decided not to use complicated lightmaps and keep the texturing process as simple as possible.

All the standard UDK models use normal maps. Normal maps are 3D textures used in faking dents and scratches and further detailing a low polygon model. They are similar to bumpmaps, textures that use black and white colours to create a “bumpy” surface where a darker shade of gray would define a bump’s climax and a light shade of gray would define a dent or hole. Normal maps use the RGB system to define more complicated “bumps”. The RGB values correspond to X, Y and Z coordinates and define 3D details without the need of a higher polygon count. Normal maps are generally made from high LOD versions of a game model. Given the time frame for this project, creating high LOD versions of each weapon was out of the question. Instead, to “fake” the normal maps, the Nvidia Photoshop plug-in was used and improved the overall look of the guns considerably.
4.1.3 Rigging and Importing

All Unreal Tournament weapons are rigged in a similar way. Each model requires two bones, one for the weapon body and one for the muzzleflash socket. Epic Games provides a sample file containing a partially rigged character model holding a gun. Based on that, I was able to scale the weapons so that they are the correct size in game. A bone was placed at the tip of the barrel of each firearm while a larger bone defines the body of the gun. The two bones have to be named “weapon_nub” and “weapon_body”. These names are required for UDK to understand and make use of the rigs. All the weapons, including the monkey wrench, are rigged as follows:

UDK uses “skeletal meshes” or game rigs for all game elements that require animation. These files use the .PSK file type and can be exported from 3D Studio Max only by using the ActorX plugin.
4.1.4 First Person Animation

A standard weapon in Unreal Tournament uses the following 5 animations. These animations are done in one continuous sequence in the 3d modelling program. They must be exported using a special plugin (ActorX) and the whole animation sequence needs to be divided into animation segments. A standard weapon in Unreal Tournament, such as the Shock Rifle, uses the following 5 animations:

<table>
<thead>
<tr>
<th>#</th>
<th>UT Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WeaponIdle</td>
<td>• The weapon in its idle state when the player is simply holding it. The WeaponIdle animation generally comprises of small organic movements.</td>
</tr>
<tr>
<td>2</td>
<td>WeaponEquip</td>
<td>• The weapon when it is selected from the inventory. The WeaponEquip animation consists of the weapon being lifted into view.</td>
</tr>
<tr>
<td>3</td>
<td>WeaponPutDown</td>
<td>• The weapon when it is deselected from the inventory. The WeaponPutDown animation consists of the weapon being lowered from the first person view.</td>
</tr>
<tr>
<td>4</td>
<td>WeaponFire</td>
<td>• Possibly the most complicated animation, the WeaponFire animation involves gun recoil, empty cartridge disposal, trigger pressing and muzzleflash.</td>
</tr>
<tr>
<td>5</td>
<td>WeaponAltFire</td>
<td>• The alternative fire animation depends on the type of weapon in question. All the weapons were designed to have alternative melee modes.</td>
</tr>
</tbody>
</table>

4.1.5 Pickup Classes

The pickup classes in games are objects that the player can add to their inventory such as health powerups and ammunition. This game uses four objects that can be obtained to sustain gameplay:

- Health Pack: restores 40% of the player’s health;
- Desert Eagle Ammunition: 30 bullets.
- Mac 10 Ammunition: 90 bullets.
- Shotgun Ammunition: 18 shells
4.2 First Person Hands

4.2.1 Modelling

The Unreal Development Kit is based on the Unreal Tournament 3 game, a futuristic first person shooter. UT3 is an online tournament-based game and is set in a distant future. This project is set in the present and, because of the difference between its theme and that of the original game, many modifications had to be made. The original game provides the player with laser beam or energy-powered weapons that are large and bulky. Because of the size of these weapons, the part where the hand holds the weapon is not visible in first person mode. Therefore, no hands are needed to hold that weapon. The weapons in this project game are much more realistic and are based on real firearms that are much smaller in size. When the futuristic weapons were replaced with smaller more realistic ones, a set of first person hands was needed for the first person view. The best way of doing this was to model a set of first person hands that would be imported and animated along with each weapon.

The following model is optimized and instantiated 4 times, once for each weapon:

![Fig 14 A female hand, modelled only for the first person view in the game.](image-url)
4.2.2 Texturing

Texturing was done in accordance with the main character model. The sleeve had to be textured in the same way that the character’s top was. The size and detail of the texture was dictated by how big the arms will appear on the screen in first person mode. Because they take a considerable amount of space on the screen, the texture size had to be large (1024 by 1024). The texture was painted from scratch and I used reference pictures of my own hand to paint the creases in the skin and fully detail the skin part of the texture. Because painting realistic skin is quite difficult and time consuming, especially for a complicated body part such as the hand, I decided to dress the hand in a leather glove. The darker and lighter parts of the arms, including the shine of the glove, were painted or “baked” into the texture.

![First person hand textured.](image)

*Fig 15 The first person hand textured.*

*More details on the hand model can be seen in section 8.10.1 of the appendix.*
4.2.3 Rigging

An arm, like any other organic model has to be rigged using a skeletal system. Essentially, bones are created and positioned inside of the mesh and envelopes are used to attach the bones to the mesh. Envelopes, as the name would imply, are containers of vertices that are assigned to each bone. For example, the tip of a finger would have a bone inside of it that uses an envelope to deform and effectively reposition all vertices that for the tip of that finger. The skeletal system uses a carefully chosen hierarchy of bones. For example, the wrist is the parent of all the child bones. When the wrist moves, all hand bones will move along with it.

Rigging an arm is difficult because of the amount of bones it has. It must have a bone for the arm itself, one or several for the wrist/palm (depending on the degree of deformation wanted) and three for each finger.

**Fig 2.3.1** The envelope of the middle section of a finger. The area linked to the bone is highlighted in red.

**Fig 16** The hierarchy of the hand model bones. Arm Model is the actual arm mesh and is not included in the hierarchy as it linked to the bone’s envelopes.

**Fig 17** Fully rigged arm.
4.2.4 Animation

The Unreal Development Kit uses files called AnimSets to store the various animations associated with a skeletal mesh. Each skeletal mesh component uses an animation channel that is unique to the corresponding object. For example, a gun’s clip will have an animation channel. The problem with adding arms as part of the weapon is the limitation of animation channels. UDK accepts as many animated objects per skeletal mesh as needed but fails at correctly make use of the animation hierarchy. For example, the left arm in the picture below is attached to the pump-action handle of the shotgun. It is therefore a child of that handle and its palm is the child of the arm and so forth. If each individual component is animated, a huge hierarchy of animations is created. This is not interpreted correctly by UDK. Because of this problem I was forced to animate the arms to hold the weapons and remove all bone constraints. As a result, the individual parts of the arms are not animated though they give the correct impression in the game as they move with the gun components. This approach of simplifying the animation sets, made the gun mechanic much easier to work with and, as a result, optimized the game.

Fig 18 Arms is cloned, mirrored and attached to the weapon. The two are animated and imported together as the first person shotgun mesh.
4.3 Level Implementation

4.3.1 Design, Layout and Blocking

The key design component in creating a level is commonly referred to as the “world/level diagram”. Also known as a mission flow chart, the world diagram represents the high level design and represents the foundation on which the level will be blocked. It is a simple document containing the various level rooms and areas that define the player navigation paths.

A world diagram will generally include all the levels contained in the game and should not be overly detailed. For the purpose of this project, the following world/level diagram is detailed as the game only contains one level. This diagram was created in the preproduction stage of the level, prior to using the level editor.

![World/level diagram](image-url)
Based on figure 2.4., a more detailed level diagram was concluded that contains information in regard to room sizes and their relative positioning. Also, the previous diagram was used in assessing how many static meshes (level objects) will be needed to fully detail the level.

A screenshot walkthrough of the final level can be seen in section 8.12 of the appendix.

4.3.2 Asset Building

UDK uses large amounts of static meshes to detail its virtual environments. Static meshes are models created outside of the level editor. Their sole purpose is to improve the visuals and aid gameplay (running behind a chair model for cover). Importing 3d Studio Max models into the UDK level editor is a straightforward process as the two applications share .ASE file type. Additionally, they also share the same scaling units and making the models the right size in Max saves a lot of time when they are added to the level.

Static meshes must have collision meshes so that the player does not walk through them. There are two ways of creating a collision mesh, in UDK or in 3ds Max. UDK can automatically generate collision boxes around simple objects such as crates. For more complex objects such as a chair or doorways more detailed collision meshes are needed. For this game, the collision meshes were modelled in 3d Studio Max. UDK detects a collision mesh if the imported file has two objects named in a specific way.

- Mesh Name: “chair”
- Collision Mesh Name: “UCX_chair” (the “UCX_” part is very important)
The following picture shows the chair mesh and its corresponding collision mesh modelled and imported.

![Chair mesh and collision mesh in UDK](image)

Fig 21 Chair mesh and collision mesh in UDK

Level assets also need texturing and UDK has a very powerful material creation tool. The following screenshot is of the elevator frame material. The game uses relatively simple materials in comparison to professionally made Unreal Tournament ones that push the engine to its limits.

![Elevator frame material using diffuse, specular, emissive and normal maps.](image)

Fig 22 Elevator frame material using diffuse, specular, emissive and normal maps.

A list of all level assets can be found in section 8.10.6 of the appendix.
4.3.3 Fractal Meshes

The level provides a reasonable amount of interaction. This is done through the creation of destructible objects. Static meshes can get destroyed through the use of fractal meshes. Fractal meshes are automatically generated boxes that slice a model into chunks. Each chunk can be assigned one of three properties:

1. **Destroyable**
2. **Support Chunk** (the whole object will fall if this chunk is hit)
3. **Spawn No Physics** (will not have any physics settings)

The doors are destructible and use support chunks for hinges. You can shoot parts of the door off but if you aim for the hinges, the door will fall over. The following diagram shows how the door fractal map works.

4.3.4 Lighting

Because the level is set in an office building, the same lighting setup can be reused all through the level. The map’s light model relies on simple spotlights and emissive materials applied to neon lamp meshes. The emissive materials give the impression that the light is coming from the bulb while the spotlights light up the environment.

The Unreal 3 engine lighting system was designed to work with dark fantastically-themed levels where the shadows displayed on the level mesh are low resolution. An office building has large white walls and the engine’s struggles to create smooth gradient shadows. To correct this, large lightmaps were used to increase the shadow details.
4.3.5 Artificial Intelligence

Artificial intelligence was implemented by defining player paths. The computer controlled enemies run on set paths around the level and each path is assigned a width/importance value. If a threat is detected, the enemy will move away from the path and attack.

AI paths are defined by nodes. Nodes act as linking points between used in generating path segments. There are three types of nodes.

- Path Definition Node
- PickupFactory Element (ammunition, weapons, health packs)
- Player Spawn Point

4.4 Motion Capture Suit Implementation

4.4.1 Linking Hardware

The eMove department have provided the motion capture suit drivers and the UDK code that communicates with the drivers to receive the stream of motion capture data. The data is stored as angles and those angles are converted into three-dimensional coordinates. A three-dimensional position is stored by UDK in a vector data structure.

4.4.2 Motion Capture Controls

The motion capture controls are implemented by extending some of the existing game code. The Unreal Development Kit source code is often enigmatic due to the fact that it is heavily optimized. The player control functions are scattered throughout tens of UnrealScript classes. The motion capture controls implementation required full understanding of the PlayerController and UTPawn class hierarchies. The control functions and variables that make the player move through keyboard and mouse inputs, once identified, had to be made accessible by the classes holding the motion capture data.

```plaintext
// Camera Rotation Functions:
function RotateRight(){
    PlayerController(Controller).PlayerInput.aTurn += scrollAmount;
}
// Walk Functions:
function WalkForward(){
    super.Dodge(DCLICK_Forward);
}
// Jump:
function YesJump(){
    super.DoJump(true);
    jumping = true;
}
// Select Weapons:
function SelectFirstWeapon(){
    SwitchWeapon(1);
}
...
```

*Fig 24 The way the motion capture suit accesses the game controls.*
The required player controls code was scattered throughout the following classes.

![Diagram of class hierarchy](image)

The eMovePawn class handles the gestured-triggered controls and works with the motion capture suit’s angle data. The eMovePlayerController class handles the suit-mounted buttons.

### 4.4.3 Console Commands

For testing purposes, a set of console commands had to be implemented to input motion capture positional values.

```plaintext
// Calibration Functions:
exec function CalLeanAmount(float x)
{
    leanAmount = x;
}

exec function CalScrollCenter(float x, float y, float z)
{
    scrollHandCenter.X = x;
    scrollHandCenter.Y = y;
    scrollHandCenter.Z = z;
}

exec function CalScrollAmount(float x)
{
    scrollAmount = x;
}
...
```

![Diagram of UDK class hierarchy](image)
4.4.4 InGame Calibration

The ingame calibration functions use a combination of buttons that when pressed, the game records relevant motion capture values and sets the game’s variables to those values.

```javascript
function SetCenter()
{
    if(rLockButton & rFireButton){
        scrollHandCenter = ReferenceMesh.GetBoneLocation('right_hand', 1);
        PlayerController(Controller).ClientMessage("Center Set!");
    }
}

function SetJumpLimit()
{
    if(lLockButton & lFireButton &
       (ReferenceMesh.GetBoneLocation('right_hand', 1).Y >
       ReferenceMesh.GetBoneLocation('neck', 1).Y)){
        jumpLimitY = ReferenceMesh.GetBoneLocation('right_hand', 1).Y;
        PlayerController(Controller).ClientMessage("Jump Limit Set!");
    }
}
...
```

Fig 27 Calibration functions.
5. Testing
5.1 No Motion Capture Suit

A significant stage in this project’s development has been the integration of the motion capture suit and its calibration. Because no motion capture suit was supplied with the unique purpose of testing this particular game a simulation program was needed. To simulate a motion capture suit, the control scheme of the game was modified so that vector coordinates can be assigned as simulated motion capture suit values. The game uses a complex control scheme and has to detect a set of carefully defined gestures requiring precise numeric values. The control and functionality was tested by manually inputting these values into the game. This was done through console commands created to access the motion capture game code, referred to in Unreal as “exec. functions”.

![Console commands that influence the motion capture code.](image)

5.2 User-Centred Testing

The eMove research department is working on several important projects and bringing people to test the game with the suit was not possible. The game was tested with the suit during its final development stage and perfected for the purpose of this submission. The time I was given to work with the eMove team was spent implementing additional features, mostly to do with the ingame calibration, optimizing the communication between the suit and the game engine and creating a list of motion capture console commands.

The suit drivers are the property of Animazoo and the eMove research team. The idea of releasing a beta client for others to test was considered but, because the drivers must not be made public, a beta testing of the game never took place. Instead, the game was stripped of any motion capture code and a set of people were given between 5-10 minutes to play the game on a test computer. The game behaved as a traditional first person shooter would and during testing, several bugs were detected. Additional feedback on the overall impression that the game gave to a first time user was also helpful.
5.3 Game Fixes

In testing the final build of the motion capture-less, the following list of bugs was concluded:

<table>
<thead>
<tr>
<th>#</th>
<th>Problem Description</th>
<th>Fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>- The AI cannot use the wrench efficiently.</td>
<td>- The wrench melee range was adjusted.</td>
</tr>
<tr>
<td></td>
<td>- Lighting issue with the couch that blocks the building’s main entrance.</td>
<td>- Couch was repositioned to receive light correctly.</td>
</tr>
<tr>
<td>2</td>
<td>- The game throws an error in loading one of the standard UDK weapons that was damaged in compiling the game.</td>
<td>- The rocket launcher files were reloaded from a fresh install of UDK.</td>
</tr>
<tr>
<td></td>
<td>- The shotgun’s sound is not synchronized with the first person animations.</td>
<td>- The shotgun’s sound cue was adjusted to match the animations.</td>
</tr>
<tr>
<td>3</td>
<td>- All weapons use the wrong impact particle system.</td>
<td>- Simple fix, the renaming of one sound cue.</td>
</tr>
<tr>
<td>4</td>
<td>- The melee attack range on the firearms is incorrect.</td>
<td>- The bug was never fixed as it requires reprogramming of some standard UDK classes. Essentially, a UDK weapon has one damage range and that range is used by both primary and secondary fire. Given the time frame for this project, this bug was never fixed.</td>
</tr>
</tbody>
</table>

5.4 Requirements Testing and Evaluation

Requirements based testing was the final stage of the testing process. It is the confirmation that the resulted application is what the client wants and concludes the various features implemented. It is an evaluation of the final build of the application and may identify any additions that require implementation in future builds.

*Test plan can be seen in section 8.7 of the appendix.*
6. Conclusions
6.1 Project Evaluation

The application was weighted against the list of requirements and satisfied the majority. There have been many unforeseen problems in the development stages of this game, particularly during the motion capture suit implementation.

It has been a steep learning curve understanding how the Unreal 3 engine pipeline works. It is a very complicated and graphic-demanding engine that requires all of the 3D content to be heavily optimised. Because the engine allows for such detailed models, the asset creation stage of this project took most of the time. The reasonable polygon count of the models produced a lag-free game. Texture creation was as time-consuming as the modelling because many of the textures were hand-painted using a tablet. The resulted game has a meticulously-built environment. The process of creating content for a modern generation engine was underestimated.

The process of integrating the motion capture suit in the application was done with the help of the eMove team that provided helpful code and advice. Making use of that motion capture data required a lot of research and understanding of the engine’s fundamentals. Gesture-based controlled gameplay was more complicated to program than anticipated and the code initially developed was not usable due to problems calculating the local positions of the character limbs. Directly mapping the suit to the player model would have been even more complicated. However, the underlining objective of the project was to develop a game based on the exiting first person shooter mass-appeal genre and find a way of implementing motion capture. As submitted, the suit’s implementation is not finished, nor bug-free. The control scheme had to be simplified due to time limitations.

The game takes, on average, 10-15 minutes to complete, four times longer than required. This shows that too much time was invested in the actual game development. More time should have been dedicated to implementing the suit and perfecting the player controls. The problem with the controls is that the game will not work unless every single control is mapped to a gesture. There is no point in trying to play a game using motion capture if you can shoot, fire, select weapons but not walk forward. Ultimately, this project’s game development (independent of the motion capture) goal was achieved, and it conforms to the requirements specified. The final package delivers a large amount of custom content and solid gameplay. The motion capture suit implementation goal was partly achieved despite programming the control scheme twice.

The evaluation can be seen in section 8.7 of the appendix.
6.2 Improvements

The following list contains various improvements that could be made to the game as well as some areas of this project that could be expanded to create a superior product. These elements were not achieved due to poor time management and several university-unrelated problems:

**Motion Capture**

- Fix and the current motion capture controls scheme to make full use of the suit.
- Find some interesting way to map the motion capture suit to the player character.

**Gameplay**

- Create a single player story-based mode.
- Add several player/enemy models.

**Other**

- Further polish the game.
- Add an in-game tutorial teaching the user how to play the game.
- Convert static meshes to physics objects and make the environment even more interactive.

6.3 Further Work and Study

If required, this project may be perfected, expanded and modified to suit the University’s needs and those of the eMove team. The process of researching motion capture and developing this application has been challenging but enjoyable and I am considering continuing research degree in this field.
7. Bibliography
7.1 Literature


7.2 References

[1] “Principles of Traditional Animation Applied to 3D Computer Animation”
   http://www.siggraph.org/education/materials/HyperGraph/animation/character_animation/principles/prin_trad_anim.htm


[22] Point of Existence: http://www.pointofexistence.com/

[23] Sound Files Source: http://www.fpsbanana.com/
8. Appendix
8.1 Project Proposal

Please note that the topics discussed in the following project proposal are no longer relevant as the project migrated from Unity to the Unreal 3 engine due to the free release of UDK in November 2009.

Design Approach

I will be developing a shooter that will work with the motion capture technology provided. The majority of the motion capture based games I’ve researched are simple and repetitive. I want to take a popular game genre and build an interesting and interactive user experience using this motion capture technology. I plan to focus on creating an attractive and immersive 3D environment.

The theme of the game will be entirely fictional and exaggerated and will hopefully intrigue the user straight from the start. The game will have a background story that I am currently trying to design. I will avoid making the game too violent by creating a themed and exaggerated storyline. I want the game to feel like a snippet of what could be a much greater story.

Technical Approach

I’ve researched Panda3D and Unity (as they work with the motion capture suit) to see which engine would be best suited for my game. My dissertation will be partly based on creating high quality 3D models and environments and therefore require a powerful engine to display that work. I have made a list of how I plan to create such artwork and import it into Unity:

- **3D mesh** importing is a straight forward process with Unity from 3ds Max. I experienced problems importing low polygon models with other games but Unity does support Max’s smoothing groups and will convert and import the mesh correctly.

- **Animation and rigging** will be done in 3ds Max. Unity supports 3ds Max animations and the bone rigs (for the characters) and they can be imported without much difficulty.

- **Texturing** will be based around UV unwrapping all my models and using those texture mapping coordinates in Unity. Importing UV maps is supported by the game engine and should be a simple process. I will not be using bumpmaps unless the time allows me to. I will be using Photoshop for the textures and a Wacom tablet for free-hand digital painting.

- **Lighting** is essential as it gives a game its general atmosphere and adds realism. Unity offers basic lighting while Panda3D doesn’t.

- **Motion capture control and calibration** is not something I was able to research but I hope to get advice soon in the research lab.

- **Programming** in Unity is done in JavaScript and C#, two programming languages that I know. Panda3D uses Python which I am not familiar with.
Primary Objectives

- **Find an interesting and enjoyable way of using motion capture** in a first person shooter. I will try to control as much of the game with intuitive body movements. For example: The angle of the torso would be controlling the forward and back movement while one of the arms will help the character look in all directions and rotate in space (effectively replacing the mouse). Buttons on the joysticks will provide additional character movement such as strafing. The joystick triggers/buttons will fire the weapons and control the in-game menu and HUD.

- **Design a main character** that the player will identify with. The main character of my game should tell a story just by the way he looks.

- **Create a set of believable weapons** that will be used to complete the game. They will have strengths and weaknesses (a limited amount of ammunition and will require reloading).

- **Design enemies** that will serve as targets and cause harm to the player making the game challenging. They will require a basic form of AI.

- **Build a detailed level** that the player will have to traverse in order to complete the game. The level design will be influenced on how good I can make the enemy AI.

Secondary Objectives

- **Develop additional weapon functionalities** such as melee, aiming through sights and anything that would be interesting to use with motion capture.

- **Create a multiplayer mode** where two or more people will be able to challenge each other on an empty version of the same level.

- **Improve lighting and add more game detail.**

Relevance

This project would be relevant to my degree as I chose the 3D/Animation strand of computer science. I plan to pursue a career in 3D and Animation and this project would be a valuable asset to my portfolio.

I have worked on games previously for the Unreal and Refractor 2 engines as an inorganic modeller and texture artist. I have researched, modelled, textured and imported weapons, vehicles and player models. I plan to use what I’ve practiced as a starting point for my game.

Research/Background Work

As of now I have researched the game engine and looked at many examples of games built on it. I have concluded what LOD the game will support and what the technical requirements are. If there was a license for the engine, I could begin importing test content to understand how the engine SDK works. I looked into motion capture games and designed a way that I think could work well with this genre.
8.2 System Requirements

Minimum System Requirements - *the game will run*

- Processor: 2.0+ GHZ Single Core Processor
- RAM: 512 Mbytes DDR2
- Graphics Card: NVIDIA 6200, ATI Radeon 9600 or better
- Hard Drive: 4 GB
- Sound Card: DirectX 9.0c compliant
- Operating System: Windows XP SP2, Vista or 7
- Unreal Development Kit:

Recommended System Requirements - *+30 frames per second, resolution set to 1680x1050, all video settings set to medium.*

- Processor: 2.4+ GHZ Dual Core
- RAM: 1GB DDR2
- Graphics Card: NVIDIA 7800GTX, ATI x1300 or better
- Hard Drive: 4 GB
- Sound Card: DirectX 9.0c compliant
- Operating System: Windows XP SP2, Vista or 7
- Unreal Development Kit:

Optimal System Requirements - *+45 frames per second, resolution set to 1920x1200, all video settings set to highest.*

- Processor: 2.4+ GHZ Dual Core
- RAM: 4GB DDR3
- Graphics Card: 512mb+ modern generation NVIDIA or ATI Radeon
- Hard Drive: 4 GB
- Sound Card: DirectX 9.0c compliant
- Operating System: Windows XP SP2, Vista or 7
- Unreal Development Kit:
8.3 Installation/CD Contents

CD Contents:

- /InstallFiles/ - all the game files that need to be manually installed.
- /Models/ - 3ds Max files of the static meshes in the game.
- /Textures/ - Targa texture files.
- /ImportFiles/ - All rigged skeletal meshes.
- /InstallationInstructions.pdf

Installation/Load Game Instructions:

1. Make a clean install of UDK version May 2010.
2. Copy the contents of /InstallFiles/ inside the freshly installed /UDK-2010-03/
3. Run Editor - Unreal Development Kit It will ask you if you want to compile, click yes.
4. Do not open the editor. Just run the game normally and follow the menu:
   Instant Action > Deathmatch > MocapGameMap > Click Next (5 opponents recommended)

- Note that running the game with the motion capture is slightly more complicated and should only be done by eMove.
8.4 Project Plan

8.4.1 Gantt Chart

The following Gantt chart encloses the various development stages included in this project. Each stage represents an amount of work that can be categorized under one topic. It displays the stages in chronological order and shows the four main milestones.

1. Background reading and research used in concluding the design.
2. Modelling and texturing game assets.
3. Rigging and animating all relevant game assets.
4. Importing game content and achieving functionality.
5. Connecting motion capture hardware to the relevant software components.
6. Receiving feedback in order to perfect the product.
7. Turning the game into a complete and presentable package.
Milestones are sets of goals or stages of completion that the project has to comply with by certain dates. The workload was divided into four progressive steps, each corresponding to a set of closely related objectives. Please note that the following information is only concerning the software development process and not any other dissertation requirements.

<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1st of January 2010</td>
<td>• Complete the modelling and animation of the basic assets needed to begin building the game. All such content must be imported into the game engine and some basic coding and functionality must be achieved.</td>
</tr>
<tr>
<td>2</td>
<td>1st of February 2010</td>
<td>• Achieve basic game functionality. At this stage, a preliminary player testing must be complete.</td>
</tr>
<tr>
<td>3</td>
<td>1st of March 2010</td>
<td>• Achieve full game functionality and finish the process of creating and animating 3D assets. At this point, all motion capture suit controls must be finalized and calibrated.</td>
</tr>
<tr>
<td>4</td>
<td>1st of April 2010</td>
<td>• All game development and player testing must be completed. The game must be finalized and presentable.</td>
</tr>
</tbody>
</table>
## 8.5 Risk Assessment

The following table encloses a list of the main risks involved in the main development stages. It comprises of the importance of each stage, the probability of failing to complete each stage and a possible solution if a failure occurs.

<table>
<thead>
<tr>
<th>#</th>
<th>Event</th>
<th>Impact</th>
<th>Probability</th>
<th>Contingency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Failing to model all the level assets.</td>
<td>Average</td>
<td>Medium</td>
<td>Create fewer and/or less detailed level assets.</td>
</tr>
<tr>
<td>2</td>
<td>Failing to texture all the level assets.</td>
<td>Average</td>
<td>Medium</td>
<td>Use fewer level assets, essentially giving up some untextured content, and compromise texture quality.</td>
</tr>
<tr>
<td>3</td>
<td>Failing to model all the player assets.</td>
<td>Large</td>
<td>Minimal</td>
<td>Create fewer and/or less detailed player assets.</td>
</tr>
<tr>
<td>4</td>
<td>Failing to texture all the player assets.</td>
<td>Large</td>
<td>Minimal</td>
<td>Use fewer player assets, essentially giving up some untextured content, and compromise texture quality.</td>
</tr>
<tr>
<td>5</td>
<td>Failing to achieve data transfer between the motion capture suit and the game.</td>
<td>Critical</td>
<td>Medium</td>
<td>Sacrifice time in other development stages and focus on achieving basic data transfer with the suit.</td>
</tr>
<tr>
<td>6</td>
<td>Failing to meet important motion capture requirements.</td>
<td>Critical</td>
<td>Medium</td>
<td>Sacrifice time in other development stages and focus on achieving basic control of the player using the suit.</td>
</tr>
<tr>
<td>7</td>
<td>Failing to meet all the motion capture requirements.</td>
<td>Large</td>
<td>High</td>
<td>Assign buttons to player actions instead of motion captured gestures.</td>
</tr>
<tr>
<td>8</td>
<td>Failing to create a simple level.</td>
<td>Large</td>
<td>Minimal</td>
<td>Use an existing level that is provided with the UDK</td>
</tr>
<tr>
<td>9</td>
<td>Failing to fully detailing the level.</td>
<td>Average</td>
<td>Medium</td>
<td>Compromise detail while keeping a consistent LOD.</td>
</tr>
<tr>
<td>10</td>
<td>Failing to create a custom menu.</td>
<td>Small</td>
<td>Minimal</td>
<td>Use the existing UDK menu</td>
</tr>
<tr>
<td>11</td>
<td>Failing to create a custom heads-up display.</td>
<td>Small</td>
<td>Minimal</td>
<td>Use the existing UDK heads-up display.</td>
</tr>
<tr>
<td>12</td>
<td>Failing to create an install wizard and instructions manual.</td>
<td>Small</td>
<td>Minimal</td>
<td>Provide a README file with instructions on how to manually copy the custom game files.</td>
</tr>
</tbody>
</table>
### 8.6 Test Plan

#### 8.6.1 Primary Requirements

This section encloses a list of all the requirements and their corresponding test results. The results conclude whether or not each of the requirements is fully met, partially met or fails:

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Comments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Motion Capture Suit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.1</td>
<td>Input in the game data from the motion capture suit peripheral into the game engine in the form of angles and/or 3d coordinates.</td>
<td>The suit sends angular data to the game.</td>
<td>PASS</td>
</tr>
<tr>
<td>R1.2</td>
<td>Provide a set of ingame console commands that can be used to calibrate/modify the suit’s settings</td>
<td>A custom set of console commands can be used to calibrate control values.</td>
<td>PASS</td>
</tr>
<tr>
<td>R1.3</td>
<td>Provide simple suit calibration on start-up.</td>
<td>The suit can be calibrated at the beginning of the game by pressing both triggers and main joystick buttons.</td>
<td>PASS</td>
</tr>
<tr>
<td>R1.4</td>
<td>Allow for modification of gesture detection sensitivities.</td>
<td>A custom set of console commands can be used to adjust sensitivity values.</td>
<td>PASS</td>
</tr>
</tbody>
</table>

**R2: Motion Capture Player Interactivity**

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Comments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2.1</td>
<td>Walk through the 3D environment using upper torso gestures.</td>
<td>The player can walk through body gestures if the user leans forward or backward.</td>
<td>PARTIAL PASS</td>
</tr>
<tr>
<td>R2.2</td>
<td>Rotate and look throughout the 3D environment by pointing with the right arm (replacing the mouse scrolling with arm positioning).</td>
<td>The player can strafe through left/right body gestures.</td>
<td>PARTIAL PASS</td>
</tr>
<tr>
<td>R2.3</td>
<td>Jump by rapidly raising the left arm.</td>
<td>The player can rotate by moving the right arm.</td>
<td>PASS</td>
</tr>
<tr>
<td>R2.4</td>
<td>Select from a set of three weapons by reaching for body parts that correspond to weapon holsters. Each weapon must have a suitable in-game animation showing the weapon being lowered, replaced and raised.</td>
<td>The player can jump and crouch by rapidly raising or lowering the left arm</td>
<td>PARTIAL PASS</td>
</tr>
<tr>
<td>R2.5</td>
<td>Fire a weapon by squeezing the motion capture suit mounted trigger. Each weapon must have a suitable in-game animation of gun recoil, empty cartridge disposal. Each muzzle flash will display</td>
<td>The game has three fully animated firearms that inflict damage when fired.</td>
<td>PASS</td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
<td>Status</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>R2.6</td>
<td>Each weapon must have an alternative melee attack triggered either through rapid swings of the gun-holding arm (imitating a punch) or by a suit mounted button. Each weapon must have a suitable in-game animation to display this action.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R3.1</td>
<td>Provide an underlying theme that will be discovered during the player’s progression through the level.</td>
<td>PARTIAL PASS</td>
<td></td>
</tr>
<tr>
<td>R3.2</td>
<td>Include a set of immediate threats and/or challenges in the form of AI-controlled enemies.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R3.3</td>
<td>Include a checkpoint/objective or scoring system that will keep track of the player’s progression through the game.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R4.1</td>
<td>Provide a short, detailed and themed level.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R4.2</td>
<td>Contain a set of static models that are representative of the overall game theme and add detail to the level.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R4.3</td>
<td>Contain a set of realistic textures that are representative of the overall game theme.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R5.1</td>
<td>Implement UDK’s menu system.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R5.1.1</td>
<td>Menu system interface must be able to access both standard UDK maps and the extended game content.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R5.2</td>
<td>Include a comprehensible HUD (heads up display) containing relevant information in completing the game.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R5.2.1</td>
<td>HUD must display a health percentage and ammo count.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R6.1</td>
<td>Because the game will use a unique set of controls based on upper body movements, those controls must be intuitive and easy to discover.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R6.2</td>
<td>Package must include an instruction manual.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R6.2.1</td>
<td>All instructions must be made available to read before the game has been started so that the user is aware of the physical requirements.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R7.1</td>
<td>Game must be developed while taking into consideration the hardware on which it will run.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R7.2</td>
<td>The assets’ level of detail (polygon count) must be optimized so that the game runs at a high and constant frame rate.</td>
<td>PASS</td>
<td></td>
</tr>
<tr>
<td>R7.3</td>
<td>The game must be tested thoroughly and all possible problem areas fixed</td>
<td>PARTIAL</td>
<td></td>
</tr>
</tbody>
</table>
before the final release.  

R8: Installation

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
<th>Comments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>R8.1</td>
<td>The game must be presented as an add-on/expansion to the Unreal Development Kit.</td>
<td></td>
<td>PASS</td>
</tr>
<tr>
<td>R8.2</td>
<td>Provide clear installation instructions.</td>
<td></td>
<td>PASS</td>
</tr>
</tbody>
</table>

### 8.6.2 Extensions

This section encloses a list of all the extended requirements and their corresponding test results. The results conclude whether or not each of the requirements is fully met, partially met or fails:

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Comments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.1</td>
<td>Provide animated first person hands that are visible only in first person and interact with the weapon animations.</td>
<td></td>
<td>PASS</td>
</tr>
<tr>
<td>R1.2</td>
<td>Provide several enemy character models to diversify the gameplay.</td>
<td></td>
<td>FAIL</td>
</tr>
<tr>
<td>R1.3</td>
<td>Create a fourth melee weapon that will make further use of the motion capture capabilities of the suit.</td>
<td></td>
<td>PASS</td>
</tr>
<tr>
<td>R2.1</td>
<td>Make use of the Unreal Development Kit fractal map system to create an interactive and destructible environment that the player can affect.</td>
<td></td>
<td>PASS</td>
</tr>
<tr>
<td>R2.2</td>
<td>Expand the library of level assets and further detail areas of the level.</td>
<td></td>
<td>PASS</td>
</tr>
<tr>
<td>R2.3</td>
<td>Add physics objects that react to the player’s movement and to the guns being fired.</td>
<td></td>
<td>PARTIAL PASS</td>
</tr>
<tr>
<td>R3.1</td>
<td>Create a section in the heads up display that displays a picture useful in understanding how to switch between weapons by reaching for different body parts.</td>
<td></td>
<td>PASS</td>
</tr>
<tr>
<td>R4.1</td>
<td>Provide a basic audio to the game including weapons and character sounds. The game may use of carefully chosen standard Unreal Development Kit sounds as well as new material.</td>
<td></td>
<td>PASS</td>
</tr>
<tr>
<td>R4.2</td>
<td>Provide clear audio instructions.</td>
<td></td>
<td>PASS</td>
</tr>
</tbody>
</table>
# 8.7 Meetings Log

The following lists enclose all meetings with the project supervisor, the eMove research team and list the topics discussed. Such meetings took place on a weekly basis.

## 8.7.1 Second Trimester

<table>
<thead>
<tr>
<th>Week</th>
<th>Person(s)</th>
<th>Discussion Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>#3</td>
<td>Supervisor: Martin White</td>
<td>• Proposed the project.</td>
</tr>
</tbody>
</table>
| #3   | eMove Team: Cash Garman Jake Slack | • Project proposal.  
• Previous eMove projects.  
• Animazoo’s motion capture suit.  
• Unity, Panda 3D and Unreal 3 game engines.  
• General game level of detail (LOD). |
| #4   | Supervisor: Martin White | • Project proposal.  
• Unity, Panda 3D and Unreal 3 game engines.  
• Motion capture suit controls.  
• General game level of detail (LOD).  
• Game theme and target audience. |
| #5   | Supervisor: Martin White | • Project proposal.  
• Overall game genre, theme and target audience.  
• Progress regarding the modelling of assets.  
• Motion capture suit controls. |
| #5   | eMove Team: Cash Garman | • Programming in Unity vs. programming in Panda 3D.  
• How to use the software provided.  
• Game control details.  
• Game level of detail. |
| #6   | Supervisor: Martin White | • Modelling progress.  
• General game level of detail (LOD).  
• Motion capture suit controls.  
• Game theme and target audience. |
| #7   | Supervisor: Martin White | • Modelling progress.  
• Unity/Unreal Development Kit engine licensing.  
• 3d Assets.  
• Audio. |
| #7   | eMove Team Cash Garman Jake Slack | • Modelling progress.  
• Game fundamentals.  
• Animation.  
• Testing sessions. |
### 8.7.2 Third Trimester

<table>
<thead>
<tr>
<th>Week</th>
<th>Person(s)</th>
<th>Discussion Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Supervisor: Martin White</td>
<td>• Interim report correction.</td>
</tr>
</tbody>
</table>
| #2   | eMove Team: Cash Garman Jake Slack | • Migrating to the Unreal Development Kit.  
• Project changes.  
• eMove UDK-based projects.  
• Animazoo’s motion capture suit. |
| #4   | Supervisor: Martin White | • Migrating to the Unreal Development Kit.  
• Project changes.  
• Weapons.  
• Level walkthrough. |
| #5   | Supervisor: Martin White | • Overall progress report.  
• Weapons in-game.  
• Adding first person view hands.  
• Level interaction.  
• Possibility of doing a PhD. |
| #6   | Supervisor: Martin White | • Overall progress report.  
• Motion capture suit controls.  
• Possibility of doing a PhD. |
| #7   | Supervisor: Martin White | • Overall progress report.  
• Motion capture suit implementation.  
• Possibility of doing a PhD. |
| #8   | eMove Team Cash Garman | • Motion capture suit implementation.  
• Game progress. |
| #9   | eMove Team Jake Slack Cash Garman | • Motion capture suit implementation.  
• Game progress. |
| #10  | Supervisor Martin White | • Motion capture suit implementation.  
• Dissertation report details. |
8.8 Focus Group

I described in general terms the overall scope of the project and presented the basic idea. What is the overall reaction on this topic?

**What key gameplay features define a good, fun and successful first person shooter game?**

- Fun
- Storyline
- Multiplayer
- Cutscenes
- Action Themed
- Sense of Achievement
- Game Physics
- Discovery
- Single Player

**What type of first person shooter do you prefer? Should I take the realistic approach and build a fight simulator or an arcade-type unrealistic game?**

- Realistic but Fun
- Fast Paced
- Discussed Source Games

**What overall theme and style should the game have?**

- Zombies
- Zombies
- Modern Warfare/Terrorists
- Steampunk
- Aliens

In terms of the heads up display, what information do you find useful to have on screen at all times.

- Objectives
- Ammunition Bar
- Health Bar
- Compass
In terms of weapons, I have built designed several weapons classes. Weapons include: a monkey wrench, the desert eagle, the mac10 and a pump-action shotgun. What do you think of these?

- Well-balanced Set
- Bigger Guns
- More Melee Weapons

Is there something that you always wanted experience in a first person shooter? If you could add a feature to your favourite FPS, what would that feature be?

- Games these days include a lot of features but they lack the fun factor. Taking the FPS genre back to its roots would be a good approach.

I presented my game motion capture controls methods. Do you find this intuitive or have any suggestions?

- Not much feedback was received for this question as, despite my efforts of explaining, no one knew exactly how the motion capture suit works. The only suggestion was to add more melee weapon controlled through arm swinging gestures.

Considering that this game will use a complicated set of controls, what do you think is the best way to teach players the controls?

- Descriptive Videos
- Introductory Tutorial
- Documentation

This game will be a demonstration of what can be done with motion capture technology. We may be trying to showcase the game. What do you think would be the best way of achieving that?

- Community Sites
- Approach Gaming Networks
- Document the Development Stages
- Videos
- Website

In terms of the game interface, what would be the best visual design approach?

- Simple
- Clear
- Clean
- Sleek

Would you be interested in testing the game?
8.9 Modelling and Texturing

8.9.1 Character Hands

<table>
<thead>
<tr>
<th></th>
<th>Polygon Count</th>
<th>Modelling</th>
<th>Rigging</th>
<th>Animation</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• 1179</td>
<td></td>
<td>• 2 days</td>
<td></td>
</tr>
<tr>
<td>Modelling</td>
<td></td>
<td>• 2.5 days</td>
<td></td>
<td>• 2 days</td>
<td></td>
</tr>
<tr>
<td>Texturing</td>
<td></td>
<td>• 2 days</td>
<td></td>
<td></td>
<td>• 0.5 days/weapon</td>
</tr>
</tbody>
</table>

![Fig 31 Player first person hands mesh render in 3ds Max.]

![Fig 32 Hands texture.]

Dissertation - Tudor Pascu - 2010 - Candidate Number 10356
8.9.2 Monkey Wrench

<table>
<thead>
<tr>
<th>Polygon Count</th>
<th>1985 (1P) 1044 (3P)</th>
<th>Rigging</th>
<th>0.5 days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling</td>
<td>3 days</td>
<td>Animation</td>
<td>1 days</td>
</tr>
<tr>
<td>Texturing</td>
<td>2 days</td>
<td>Implementation</td>
<td>2 days</td>
</tr>
</tbody>
</table>

**Fig 33** Wrench first person/third person meshes in UDK.

**Fig 34** Wrench texture.
8.9.3 Desert Eagle

<table>
<thead>
<tr>
<th>Polygon Count</th>
<th>• 2761 (1P) 1816 (3P)</th>
<th>Rigging</th>
<th>• 0.5 days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling</td>
<td>• 3 days</td>
<td>Animation</td>
<td>• 1 days</td>
</tr>
<tr>
<td>Texturing</td>
<td>• 2 days</td>
<td>Implementation</td>
<td>• 2 days</td>
</tr>
</tbody>
</table>

**Fig 35 Desert Eagle first person/third person meshes in UDK.**

**Fig 36 Desert Eagle texture.**
8.9.4 Mac 10

| Polygon Count | • 3342 (1P) 2401 (3P) | Rigging | • 0.5 days |
| Modelling     | • 3 days              | Animation | • 1 days |
| Texturing     | • 2 days              | Implementation | • 3 days |

Fig 37 Mac10 first person/third person meshes in UDK.

Fig 38 Mac 10 texture.
### 8.9.5 Shtogun

<table>
<thead>
<tr>
<th>Category</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Polygon Count</strong></td>
<td>4126 (1P) 2244 (3P)</td>
</tr>
<tr>
<td><strong>Modelling</strong></td>
<td>3.5 days</td>
</tr>
<tr>
<td><strong>Texturing</strong></td>
<td>2.5 days</td>
</tr>
<tr>
<td><strong>Rigging</strong></td>
<td>0.5 days</td>
</tr>
<tr>
<td><strong>Animation</strong></td>
<td>1.5 days</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>2.5 days</td>
</tr>
</tbody>
</table>

*Fig 39 Shotgun first person/third person meshes in UDK.*

*Fig 40 Shotgun texture.*
8.9.6 Level Assets List

![Fig 41 Folder]
![Fig 42 File Cabinet]

![Fig 43 Exit Sign]
![Fig 44 Exit Doors]

![Fig 45 Elevator Frame]
![Fig 46 Elevator Door]
Fig 59 Box

Fig 60 Pencil

Fig 61 Neon Light

Fig 62 Magazine

Fig 63 Window
8.9.7 Character Model

This character model was never used as implementing it using a standard Unreal 3 character rig is not possible. The dimensions of a female body differ too much from the UDK rig. To implement this model, a new rig would be needed.
8.10 Code

Written in a text editor that does not provide colour-highlighting of code elements.

8.10.1 Motion Capture Control Class Using Local Positions

- **Despite developing and testing this code through coordinate input simulation, it was impossible to input the motion capture suit values. This was due to Unreal 3 not being able to correctly identify local bone positions. This code can be made to work if that problem is solved.**

```cpp
/**
 * Motion Capture Player Control by Tudor Pascu (unless otherwise stated) - tp60@sussex.ac.uk
 */

class eMovePawn extends UTPawn;

// The reference skeleton
var SkeletalMeshComponent ReferenceMesh;
var array<BoneMapping> BoneMappings;

// Limit Values:
var float dZAmount;
var float crouchLimitY;
var float jumpLimitY;
var Vector scrollHandCenter;
var float scrollAmount;
var float leanAmount;
var bool jumping;
var bool crouching;

// Buttons:
var bool lockButton;
var bool fireButton;
var bool altFireButton;

// Neck Position:
var Vector neck;

// Right Hand Position:
var Vector rightHand;

// Left Hand Position:
var Vector leftHand;

// Weapon Positions:
var Vector weapon1;
var Vector weapon2;
var Vector weapon3;
var Vector weapon4;

// Defines Default Values:
DefaultProperties
{
    // Limit Values:
    dZAmount = 0.01
    crouchLimitY = 2
}
jumpLimitY = 10
scrollHandCenter = vect(1,2,3);
leanAmount = 0.25;
jumping = false;
crouching = false;

// Spine Position:
neck = vect(5,5,0);

// Right Hand Position:
rightHand = vect(0,4,0);

// Left Hand Position:
leftHand = vect(0,5,0);

// Weapon Positions:
weapon1 = vect(-2,2,0);
weapon2 = vect(2,2,0);
weapon3 = vect(0,4,0);
weapon4 = vect(0,7,0);

// Buttons:
lockButton = false;
fireButton = false;
altFireButton = false;

// Attaching the reference mesh to the player
Begin Object Class=SkeletalMeshComponent Name=ReferenceSkeletalMeshComponent
  AnimTreeTemplate=AnimTree'EMoveCommon.Robot.Robot_Controller'
  SkeletalMesh=SkeletalMesh'EMoveCommon.Robot'
End Object
Components.Add(ReferenceSkeletalMeshComponent)
ReferenceMesh = ReferenceSkeletalMeshComponent

// Default bone mappings (by Cash Garman)
BoneMappings.Empty
BoneMappings.Add((ControllerName="hips", AIndex=2, BIndex=0, CIndex=1, AMultiplier=-1, BMultiplier=1, CMultiplier=1, euler_order=ZYX))
BoneMappings.Add((ControllerName="spine", AIndex=5, BIndex=3, CIndex=4, AMultiplier=-1, BMultiplier=1, CMultiplier=1, euler_order=ZYX))
BoneMappings.Add((ControllerName="neck", AIndex=8, BIndex=6, CIndex=7, AMultiplier=-1, BMultiplier=1, CMultiplier=1, euler_order=ZYX))
BoneMappings.Add((ControllerName="head", AIndex=11, BIndex=9, CIndex=10, AMultiplier=-1, BMultiplier=1, CMultiplier=1, euler_order=ZYX))
BoneMappings.Add((ControllerName="right_shoulder", AIndex=15, BIndex=17, CIndex=16, AMultiplier=1, BMultiplier=-1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="right_arm", AIndex=23, BIndex=21, CIndex=22, AMultiplier=1, BMultiplier=-1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="right_forearm", AIndex=29, BIndex=27, CIndex=28, AMultiplier=1, BMultiplier=-1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="right_hand", AIndex=35, BIndex=33, CIndex=34, AMultiplier=1, BMultiplier=1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="left_shoulder", AIndex=12, BIndex=14, CIndex=13, AMultiplier=-1, BMultiplier=1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="left_arm", AIndex=20, BIndex=18, CIndex=19, AMultiplier=1, BMultiplier=1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="left_forearm", AIndex=26, BIndex=24, CIndex=25, AMultiplier=1, BMultiplier=1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="right_hand", AIndex=32, BIndex=30, CIndex=31, AMultiplier=1, BMultiplier=1, CMultiplier=1, euler_order=ZXY))
}

// Called Over Time to Refresh Player Input:
simulated event Tick(float DeltaTime)
{
  super.Tick(DeltaTime);
}
Motion Capture Shooter

SelectFourthWeapon();
if(rightHand.X < (scrollHandCenter.X - dZAmount) && lockButton == false){
    RotateRight();
}
if(rightHand.X > (scrollHandCenter.X + dZAmount) && lockButton == false){
    RotateLeft();
}
if(rightHand.Y > (scrollHandCenter.Y + dZAmount) && lockButton == false){
    RotateUp();
}
if(rightHand.Y < (scrollHandCenter.Y - dZAmount) && lockButton == false){
    RotateDown();
}
if(neck.Z > leanAmount){
    WalkForward();
} else if(neck.Z < -leanAmount){
    WalkBackwards();
}
if(neck.X > leanAmount){
    StrafeLeft();
} else if(neck.X < -leanAmount){
    StrafeRight();
}
if(leftHand.Y > jumpLimitY){
    YesJump();
} else{
    NoJump();
}
if(leftHand.Y < crouchLimitY){
    YesCrouch();
} else{
    NoCrouch();
}
    SelectFirstWeapon();
    SelectSecondWeapon();
    SelectThirdWeapon();
    SelectFourthWeapon();
}
if(fireButton){
    YesFire();
} else if(altFireButton){
    YesAltFire();
} else{
    NoFire();
}

// TODO: Do anything you want to to the reference mesh here (ie. move it around, whatever)
// log("[eMove] Reference mesh: "$ReferenceMesh);
PlayerController(Controller).ClientMessage("[eMove] Right hand world space: 
"$ReferenceMesh.GetBoneLocation('right_hand', 0));
PlayerController(Controller).ClientMessage("[eMove] Right hand bone space: 
"$ReferenceMesh.GetBoneLocation('right_hand', 1));
}

function UpdateMocapValues()
{
    rightHand = ReferenceMesh.GetBoneLocation('right_hand', 1);
    neck = ReferenceMesh.GetBoneLocation('neck', 1);
    leftHand = ReferenceMesh.GetBoneLocation('right_hand', 1);
}

// Calibration Functions:
exec function CalLeanAmount(float x)
{
    leanAmount = x;
}

exec function CalScrollCenter(float x, float y, float z)
{
    scrollHandCenter.X = x;
    scrollHandCenter.Y = y;
    scrollHandCenter.Z = z;
}

exec function CalScrollAmount(float x)
{
    scrollAmount = x;
}

exec function CalDeadZone(float x)
{
    dZAmount = x;
}

exec function CalJumpLimit(float x)
{
    jumpLimitY = x;
}

exec function CalWeapon1(float x, float y, float z)
{
    weapon1.X = x;
    weapon1.Y = y;
    weapon1.Z = z;
}

exec function CalWeapon2(float x, float y, float z)
{
    weapon2.X = x;
    weapon2.Y = y;
    weapon2.Z = z;
}

exec function CalWeapon3(float x, float y, float z)
{
    weapon3.X = x;
    weapon3.Y = y;
    weapon3.Z = z;
}

exec function CalWeapon4(float x, float y, float z)
{
    weapon4.X = x;
    weapon4.Y = y;
    weapon4.Z = z;
}
// Camera Rotation Functions:
function RotateRight(){
    PlayerController(Controller).PlayerInput.aTurn += scrollAmount;
}

function RotateLeft(){
    PlayerController(Controller).PlayerInput.aTurn -= scrollAmount;
}

function RotateUp(){
    PlayerController(Controller).PlayerInput.aLookUp += scrollAmount;
}

function RotateDown(){
    PlayerController(Controller).PlayerInput.aLookUp -= scrollAmount;
}

// Walk Functions:
function WalkForward(){
    super.Dodge(DCLICK_Forward);
}

function WalkBackwards(){
    super.Dodge(DCLICK_Back);
}

function StrafeLeft(){
    super.Dodge(DCLICK_Left);
}

function StrafeRight(){
    super.Dodge(DCLICK_Right);
}

// Jump:
function YesJump(){
    super.DoJump(true);
    jumping = true;
}

function NoJump(){
    if(jumping){
        super.DoJump(false);
        jumping = false;
    }
}

// Crouch:
function YesCrouch(){
    super.StartCrouch(-20);
    crouching = true;
}

function NoCrouch(){
    if(crouching){
        super.EndCrouch(20);
        crouching = false;
    }
}

// Select Weapons:
function SelectFirstWeapon(){
    SwitchWeapon(1);
}

function SelectSecondWeapon(){
    SwitchWeapon(2);
}
function SelectThirdWeapon()
{
    SwitchWeapon(3);
}

function SelectFourthWeapon()
{
    SwitchWeapon(4);
}

// Fire Weapon:
function YesFire()
{
    super.StartFire(0);
}
function YesAltFire()
{
    super.StartFire(1);
}
function NoFire()
{
    super.StopFire(0);
    super.StopFire(1);
}

// Handle Mocap Angles (by Cash Garman)
function HandleMocapAngles(array<float> Angles)
{
    local int i;
    local Rotator StartRot, EndRot;
    local Quat Quaternion;

    // Loop through all the mocap bone mappings
    for(i = 0; i < BoneMappings.Length; i++)
    {
        if(BoneMappings[i].Controller != none)
        {
            // Convert the mocap rotation to a rotator
            StartRot.Roll = 65535.0 * (Angles[BoneMappings[i].AIndex]/360.0) * BoneMappings[i].AMultiplier;
            StartRot.Pitch = 65535.0 * (Angles[BoneMappings[i].BIndex]/360.0) * BoneMappings[i].BMultiplier;
            StartRot.Yaw = 65535.0 * (Angles[BoneMappings[i].CIndex]/360.0) * BoneMappings[i].CMultiplier;

            // Convert the rotator to a quaternion (to make it order independant)
            Quaternion = class'eMove'.static.RotToQuat(StartRot, BoneMappings[i].euler_order);

            // Convert the quaternion back into a rotatot (to convert it to the unreal specific rotation order)
            EndRot = QuatToRotator(Quaternion);

            // Apply the rotation to the bone
            BoneMappings[i].Controller.BoneRotation = EndRot;
        }
    }
}

// Assign Bones Values (by Cash Garman)
simulated function PostBeginPlay()
{
    local int i;
    Super.PostBeginPlay();
8.10.2 Motion Capture Control Class Using Hand Distances and Buttons

- **This code was used in the final submission though it does not use the full motion capture suit potential.**

```csharp
/**
 * Motion Capture Player Control by Tudor Pascu (unless otherwise stated) - tp60@sussex.ac.uk
 */

class eMovePawn extends UTPawn;

// The reference skeleton
var SkeletalMeshComponent ReferenceMesh;
var array<BoneMapping> BoneMappings;

// Limit Values:
var float jumpLimit; // used to detect jump
var float weaponTouchLimit; // how close to collect weapon
var float scrollAmount; // how fast the scrolling should be done
var float wpNum; // weapon number
var float walkLimit; // how far should the arm be extended to walk forward
var float heightCenter; // height center to detect vertical rotation
var float heightS; // height detection sensitivity

// Buttons:
var bool rr; // rotate right flag
var bool rl; // rotate left flag
var bool rd; // rotate down flag
var bool ru; // rotate up flag
var bool wf; // walk forward flag
var bool fire; // fire
var bool aFire; // alternative fire
var bool lLock; // right hand joystick lock
var bool closeEnough; // is the weapon close enough to the hand to be picked up
var bool jumping; // is the player jumping

// Defines Default Values:
DefaultProperties {
    // Attaching the reference mesh to the player
    Begin Object Class=SkeletalMeshComponent Name=ReferenceSkeletalMeshComponent
    AnimTreeTemplate=AnimTree'eMoveCommon.Robot.Robot_Controller'
    SkeletalMesh=SkeletalMesh'eMoveCommon.Robot.Robot'
    End Object
    Components.Add(ReferenceSkeletalMeshComponent)
    ReferenceMesh = ReferenceSkeletalMeshComponent

    // Default bone mappings (byC)
    BoneMappings.Empty
}
```
BoneMappings.Add((ControllerName="hips", AIndex=2, BIndex=0, CIndex=1, AMultiplier=-1, BMultiplier=-1, CMultiplier=1, euler_order=ZYZ))
BoneMappings.Add((ControllerName="spine", AIndex=5, BIndex=3, CIndex=4, AMultiplier=-1, BMultiplier=-1, CMultiplier=1, euler_order=ZYZ))
BoneMappings.Add((ControllerName="neck", AIndex=8, BIndex=6, CIndex=7, AMultiplier=-1, BMultiplier=-1, CMultiplier=1, euler_order=ZYZ))
BoneMappings.Add((ControllerName="head", AIndex=11, BIndex=9, CIndex=10, AMultiplier=-1, BMultiplier=-1, CMultiplier=1, euler_order=ZYZ))
BoneMappings.Add((ControllerName="right_shoulder", AIndex=15, BIndex=17, CIndex=16, AMultiplier=1, BMultiplier=-1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="right_arm", AIndex=23, BIndex=21, CIndex=22, AMultiplier=1, BMultiplier=-1, CMultiplier=-1, euler_order=ZYX))
BoneMappings.Add((ControllerName="right_forearm", AIndex=29, BIndex=27, CIndex=28, AMultiplier=-1, BMultiplier=1, CMultiplier=-1, euler_order=ZXY))
BoneMappings.Add((ControllerName="right_hand", AIndex=35, BIndex=33, CIndex=34, AMultiplier=1, BMultiplier=1, CMultiplier=1, euler_order=ZYX))
BoneMappings.Add((ControllerName="left_shoulder", AIndex=12, BIndex=14, CIndex=13, AMultiplier=1, BMultiplier=1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="left_arm", AIndex=20, BIndex=18, CIndex=19, AMultiplier=1, BMultiplier=1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="left_forearm", AIndex=26, BIndex=24, CIndex=25, AMultiplier=1, BMultiplier=1, CMultiplier=1, euler_order=ZXY))
BoneMappings.Add((ControllerName="left_hand", AIndex=32, BIndex=30, CIndex=31, AMultiplier=1, BMultiplier=1, CMultiplier=1, euler_order=ZXY))

// Called Over Time to Refresh Player Input:
simulated event Tick(float DeltaTime) {
    super.Tick(DeltaTime);
    if(rr){
        RotateRight();
    }
    if(rl){
        RotateLeft();
    }
    if(aFire && rl){
        heightCenter = QuatToRotator(ReferenceMesh.GetBoneQuaternion('left_arm', 0)).Pitch;
    }
    PlayerController(Controller).ClientMessage(QuatToRotator(ReferenceMesh.GetBoneQuaternion('left_arm', 0)).Pitch);
    if(QuatToRotator(ReferenceMesh.GetBoneQuaternion('left_arm', 0)).Pitch > heightCenter + heightS){
        RotateUp();
    }
    if(QuatToRotator(ReferenceMesh.GetBoneQuaternion('left_arm', 0)).Pitch < heightCenter - heightS){
        RotateDown();
    }
        WalkForward();
    }
    if(abs(VSize(ReferenceMesh.GetBoneLocation('right_hand', 0)) - VSize(ReferenceMesh.GetBoneLocation('hips', 0))) < weaponTouchLimit && lLock){
        closeEnough = true;
    }
    if(closeEnough){
        if(wpNum == 4){
            wpNum = 1;
        }else{
            wpNum = wpNum + 1;
        }
    }
}
SelectNextWeapon(wpNum);
closeEnough = false;
}
if(fire){
    YesFire();
} else if(aFire){
    YesAltFire();
} else{
    NoFire();
}
PlayerController(Controller).ClientMessage(abs(VSize(ReferenceMesh.GetBoneLocation('right_hand', 0)) - VSize(ReferenceMesh.GetBoneLocation('spine', 0))));
if(abs(VSize(ReferenceMesh.GetBoneLocation('right_hand', 0)) - VSize(ReferenceMesh.GetBoneLocation('spine', 0))) < walkLimit){
    YesJump();
} else{
    NoJump();
}

// Calibration Functions:
exec function CalScrollAmount(float x)
{
    scrollAmount = x;
}
exec function CalJumpWeaponTouchLimit(float x)
{
    weaponTouchLimit = x;
}
exec function CalWalkLimit(float x)
{
    walkLimit = x;
}
exec function CalHeightSensitivityLimit(float x)
{
    jumpLimit = x;
}

// Camera Rotation Functions:
function RotateRight(){
    PlayerController(Controller).PlayerInput.aTurn += scrollAmount;
}
function RotateLeft(){
    PlayerController(Controller).PlayerInput.aTurn -= scrollAmount;
}
function RotateUp(){
    PlayerController(Controller).PlayerInput.aLookUp += scrollAmount;
}
function RotateDown(){
    PlayerController(Controller).PlayerInput.aLookUp -= scrollAmount;
}

// Walk Functions:
function WalkForward(){
    super.Dodge(DCLICK_Forward);
}

// Jump:
function YesJump()
{
    super.DoJump(true);
    jumping = true;
}
function NoJump()
{
    if(jumping){
        super.DoJump(false);
        jumping = false;
    }
}

// Select Weapons:
function SelectNextWeapon(int x){
    SwitchWeapon(x);
}

// Fire Weapon:
function YesFire()
{
    super.StartFire(0);
}
function YesAltFire()
{
    super.StartFire(1);
}
function NoFire()
{
    super.StopFire(0);
    super.StopFire(1);
}

// Handle Mocap Angles (by Cash Garman)
function HandleMocapAngles(array<float> Angles)
{
    local int i;
    local Rotator StartRot, EndRot;
    local Quat Quaternion;

    // Loop through all the mocap bone mappings
    for(i = 0; i < BoneMappings.Length; i++)
    {
        if(BoneMappings[i].Controller != none)
        {
            // Convert the mocap rotation to a rotator
            StartRot.Roll = 65535.0 * (Angles[BoneMappings[i].AIndex]/360.0) * BoneMappings[i].AMultiplier;
            StartRot.Pitch = 65535.0 * (Angles[BoneMappings[i].BIndex]/360.0) * BoneMappings[i].BMultiplier;
            StartRot.Yaw = 65535.0 * (Angles[BoneMappings[i].CIndex]/360.0) * BoneMappings[i].CMultiplier;

            // Convert the rotator to a quaternion (to make it order independant)
            Quaternion = class'eMove'.static.RotToQuat(StartRot, BoneMappings[i].euler_order);

            // Convert the quaternion back into a rotatot (to convert it to the unreal specific rotation order)
            EndRot = QuatToRotator(Quaternion);

            // Apply the rotation to the bone
            BoneMappings[i].Controller.BoneRotation = EndRot;
        }
    }
}
// Initial Values
simulated function PostBeginPlay()
{
    local int i;

    // setting all booleans to false
    rr = false;
    rl = false;
    rd = false;
    ru = false;
    wf = false;
    fire = false;
    aFire = false;
    lLock = false;
    closeEnough = false;

    // Assigning starting values
    jumpLimit = 5.5;
    weaponTouchLimit = 5;
    scrollAmount = 0.60;
    walkLimit = 5;
    heightS = 500;

    Super.PostBeginPlay();
    Mesh.SetHidden(true);

    // Cache all the bone controllers
    for(i = 0; i < BoneMappings.Length; i++)
        BoneMappings[i].Controller = SkelControlSingleBone(ReferenceMesh.FindSkelControl(BoneMappings[i].ControllerName));
}

8.10.3 Button Trigger Class

// The eMove player controller
//
// Cash Garman 2010 (c.garman@sussex.ac.uk)
// Modified by Tudor Pascu (tp60@sussex.ac.uk)
class eMovePlayerController extends PlayerController
    DLLBind(eMove);

// eMove details
var(eMove) float UpdateFrequency;
var(eMove) string DefaultActorFile;

// Mocap data
var array<float> Angles; // Current joint angles
var array<int> Buttons; // Current button states
var array<int> PrevButtons; // Previous button states

// Calibration
var bool CalibrationPressed; // Are the calibration buttons pressed?
var bool NorthDirectionSet; // Has the suit's north direction been set?
var bool ScreenDirectionSet; // Has the suit's live direction been set?
var bool ControlsCalibrated; // Have the controls been calibrated yet?
var bool GetDefaultPosition;

// eMove DLL methods
dllimport final function bool eMoveInit();
dllimport final function eMoveShutdown();
dllimport final function int eMoveFindSuits();
dllimport final function bool eMoveConnect();
dllimport final function eMoveClose();
dllimport final function string eMoveGetData();
dllimport final function eMoveSetNorthDirection();
dllimport final function eMoveSetScreenDirection();
dllimport final function bool eMoveLoadActorFile(string filename);

defaultproperties
{
    Name="eMovePlayerController"
    // eMove details
    UpdateFrequency = 0.01
    DefaultActorFile = "C:\Users\Cash\Documents\eMove\defaultActor.ac"
}

// Display detailed information on screen on all the pawns
simulated exec function ShowPawnInfo()
{
    `log("[eMove] Toggling pawn info");
    ConsoleCommand("displayall eMovePawn Blend");
}

// Set the eMove suit's north direction
simulated exec function SetSuitNorthDirection()
{
    `log("[eMove] Suit north direction set");
    eMoveSetNorthDirection();
}

// Set the eMove suit's screen direction
simulated exec function SetSuitScreenDirection()
{
    `log("[eMove] Suit screen direction set");
    eMoveSetScreenDirection();
}

simulated function PostBeginPlay()
{
    local int i;
    Super.PostBeginPlay();

    // If we can successfully initialise the Gyro
    if(eMoveInit())
    {
        // Success
        `log("[eMove] Successfully initialised eMove system");
        ClientMessage("Successfully initialised eMove system");

        // Search for any eMove suits
        `log("[eMove] Number of suits found: "$eMoveFindSuits());

        // Try to connect to any available eMove suit
        if(eMoveConnect())
        {
            // Success
            `log("[eMove] Successfully connected to an eMove suit");
            ClientMessage("Successfully connected to an eMove suit");

            // Load the default actor file
            `log("[eMove] Loading the default eMove actor file: "$DefaultActorFile);
            ClientMessage("Loading the default eMove actor file: "$DefaultActorFile);
            eMoveLoadActorFile(DefaultActorFile);

            // Start the mocap request timer
            //...
        }
    }
}
Motion Capture Shooter

```
log("[eMove] Starting eMove request timer");
SetTimer(UpdateFrequency, true);
}
else
{
    // Failed to connect
    log("[eMove] Failed to connect to an eMove suit");
    ClientMessage("Failed to connect to an eMove suit");
}
else
{
    // Failed to initialise
    log("[eMove] Failed to initialise eMove system");
    ClientMessage("Failed to initialise eMove system");
}
// Set all the previous buttons states to released
for(i = 0; i < 6; i++)
    PrevButtons[i] = 1;
// Stop the loading screen
class'Engine'.static.StopMovie(true);
// Set the suit's north as unset
NorthDirectionSet = false;
// Set the calibration buttons as not pressed
CalibrationPressed = false;
// Set the calibration right arm position.
GetDefaultPosition = false;
}
// Handle any cleanup when the controller is destroyed
event Destroyed()
{
    // Close the connection to the eMove suit
eMoveClose();
    // Shutdown the eMove system
eMoveShutdown();
    Super.Destroyed();
}
// Handle mocap data
function HandleMocapData(string MocapData)
{
    local array<string> parts;
    local int i;
    // Split the response up into pieces
    parts = SplitString(MocapData, " ");
    // Update the current mocap angles
    for(i = 0; i < 36; i++)
        Angles[i] = float(parts[i]);
    // Update the current button states
    for (i = 36; i < 42; i++)
        Buttons[i - 36] = int(parts[i]);
    // Send the mocap angles to the player
    eMovePawn(Pawn).HandleMocapAngles(Angles);
    ///// BUTTON 1 - Left joystick's center button////
```
// If button was pressed
if(Buttons[0] == 0 && PrevButtons[0] == 1 && ControlsCalibrated)
{
    eMovePawn(Pawn).rl = true;
}
// If button was released
else if(Buttons[0] == 1 && PrevButtons[0] == 0 && ControlsCalibrated)
{
    eMovePawn(Pawn).rl = false;
}

///// BUTTON 2 - Left joystick's trigger /////
// If button was pressed
if(Buttons[1] == 0 && PrevButtons[1] == 1 && ControlsCalibrated)
{
    eMovePawn(Pawn).aFire = true;
}
// If button was released
else if(Buttons[1] == 1 && PrevButtons[1] == 0 && ControlsCalibrated)
{
    eMovePawn(Pawn).aFire = false;
}

///// BUTTON 3 - Right joystick's right button /////
// If button was pressed
// If button was released

///// BUTTON 4 - Right joystick's left button /////
// If button was pressed
// If button was released

///// BUTTON 5 - Right joystick's center button /////
// If button was pressed
{
    eMovePawn(Pawn).rr = true;
}
// If button was released
{
    eMovePawn(Pawn).rr = false;
}

///// BUTTON 6 - Right joystick's trigger /////
// If button was pressed
{
    eMovePawn(Pawn).fire = true;
}
// If button was released
{
    eMovePawn(Pawn).fire = false;
}

// If all 4 calibration buttons has just been pressed
&& !CalibrationPressed)
{
    // If the suit's north has not been set yet
    if(!NorthDirectionSet)
    {
        // Set the suit's north direction
        SetSuitNorthDirection();
    }
// Flag the suit's north as set
NorthDirectionSet = true;
// Otherwise, if the suit's live has not been set yet
else if(!ScreenDirectionSet)
{
    // Set the suit's screen direction
    SetSuitScreenDirection();
    // Flag the suit's screen as set
    ScreenDirectionSet = true;
}
// If any of the calibration buttons has been released and all 4 we
// previously pressed
else if(CalibrationPressed && (Buttons[0] == 1 || Buttons[1] == 1 ||
    // Set the calibration buttons as released
    CalibrationPressed = false;
    // Update the previous button states
    PrevButtons = Buttons;
}
// Mocap request timer
function Timer()
{
    local string MocapData;
    // Grab the latest motion capture data
    MocapData = eMoveGetData();
    // Handle the data
    HandleMocapData(MocapData);
}
8.10.4 Weapon Classes

- **Used to define a weapon's properties:** firing range, 1st person mesh, sounds etc.

```cpp
/**
 * Wrench Weapon Code by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTWeap_Wrench extends UTWeap_WrenchBase;

defaultproperties
{
    //Weapon skeletal mesh:
    Begin Object class=AnimNodeSequence Name=MeshSequenceA
    End Object
    Begin Object Name=FirstPersonMesh
        SkeletalMesh=WP_Wrench.Mesh.Mesh_Wrench_1P
        AnimSets(0)=AnimSet'WP_Wrench.Anim.Anim_Wrench'
        Animations=MeshSequenceA
        Rotation=(Yaw=-16384)
        FOV=60.0
    End Object

    //3rd Person Mesh:
    AttachmentClass=class'UTGameContent.UTAttachment_Wrench'
    Begin Object Name=PickupMesh
        SkeletalMesh=WP_Wrench.Mesh.Mesh_Wrench_3P
    End Object

    //Weapon firing:
    InstantHitMomentum(0)=+60000.0
    WeaponFireTypes(0)=EWFT_InstantHit
    WeaponFireTypes(1)=EWFT_InstantHit

    //Damage inflicted:
    FireInterval(0)=+0.6
    FireInterval(1)=+0.9
    InstantHitDamageTypes(0)=None
    InstantHitDamageTypes(1)=None
    InstantHitDamage(0)=25
    InstantHitDamage(1)=50

    //Sounds:
    WeaponFireSnd[0]=SoundCue'WP_All.Sound.swingsound_Cue'
    WeaponFireSnd[1]=SoundCue'WP_Shotgun.Sound.shotgunswingsound_Cue'
    WeaponEquipSnd=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR.RaiseCue'
    WeaponPutDownSnd=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR.LowerCue'
    PickupSound=SoundCue'A_Pickups.Weapons.Cue.A_Pickup_Weapons_Shock_Cue'

    //AI:
    MaxDesireability=0.65
    AIRating=0.65
    CurrentRating=0.65

    //How the weapon should fire:
    bInstantHit=false
    bSplashJump=false
    bRecommendSplashDamage=false
    bSniping=true
    ShouldFireOnRelease(0)=0
    ShouldFireOnRelease(1)=0

    //How many bullets spent through primary/secondary fire:
    ShotCost(0)=0
    ShotCost(1)=0
}
//Positioning the first person mesh:
FireOffset=(X=20,Y=5)
PlayerViewOffset=(X=30.0,Y=7.5,Z=-5.5)

//Ammunition:
AmmoCount=0
LockerAmmoCount=0
MaxAmmoCount=0

//Camera animation when the weapon is fired.
FireCameraAnim(1)=CameraAnim'Camera_FX.ShockRifle.C_WP_ShockRifle_Alt_Fire_Shake'

//Tells UDK what animation to use for alternative fire:
WeaponFireAnim(1)=WeaponAltFire

//Where to place the crosshair:
CrossHairCoordinates=(U=256,V=0,UL=64,VL=64)

//Stop the camera from rotating:
LockerRotation=(Pitch=32768,Roll=16384)

//Weapon colour despite its texture (not used/important):
WeaponColor=(R=160,G=0,B=255,A=255)

//Weapon range:
WeaponRange=50 //as this is a melee weapon, it has a limited firing range.

//Inventory details:
InventoryGroup=1
GroupWeight=0.5

//Weapon Icon:
IconCoordinates=(U=728,V=382,UL=162,VL=45)
IconX=400
IconY=129
IconWidth=22
IconHeight=48

}/** * Desert Eagle Weapon Code by Tudor Pascu - tp60@sussex.ac.uk */
class UTWeap_DesertEagle extends UTWeap_DesertEagleBase;

defaultproperties
{
    //Weapon skeletal mesh:
    Begin Object class=AnimNodeSequence Name=MeshSequenceA
    End Object
    Begin Object Name=FirstPersonMesh
        SkeletalMesh=SkeletalMesh'WP_DesertEagle.Mesh.Mesh_DesertEagle_1P'
        AnimSets(0)=AnimSet'WP_DesertEagle.Anim.Anim_DesertEagle'
        Animations=MeshSequenceA
        Rotation=(Yaw=-16384)
        FOV=60.0
    End Object
    //3rd Person Mesh:
    AttachmentClass=class'UTGameContent.UTAttachment_DesertEagle'
    Begin Object Name=PickupMesh
        SkeletalMesh=SkeletalMesh'WP_DesertEagle.Mesh.Mesh_DesertEagle_3P'
    End Object
    //Weapon firing:
    InstantHitMomentum(0)=60000.0
    WeaponFireTypes(0)=EWFT_InstantHit
WeaponFireTypes(1)=EWFT_InstantHit

//Damage inflicted:
FireInterval(0)=+0.4
FireInterval(1)=+0.6
InstantHitDamageTypes(0)=None
InstantHitDamageTypes(1)=None
InstantHitDamage(0)=30
InstantHitDamage(1)=30

//Sounds:
WeaponFireSnd[0]=SoundCue'WP_DesertEagle.Sound.deserteaglesound_Cue'
WeaponFireSnd[1]=SoundCue'WP_All.Sound.swingsound_Cue'
WeaponEquipSnd=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_RaiseCue'
WeaponPutDownSnd=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_LowerCue'
PickupSound=SoundCue'A_Pickups.Weapons.Cue.A_Pickup_Weapons_Shock_Cue'

//AI:
MaxDesireability=0.65
AIRating=0.65
CurrentRating=0.65

//How the weapon should fire:
bInstantHit=true
bSplashJump=false
bRecommendSplashDamage=false
bSniping=true
ShouldFireOnRelease(0)=0
ShouldFireOnRelease(1)=0

//How many bullets spent through primary/secondary fire:
ShotCost(0)=1
ShotCost(1)=0

//Positioning the first person mesh:
FireOffset=(X=0,Y=0)
PlayerViewOffset=(X=25.0.0,Y=5.5,Z=-6.5)

//Ammunition:
AmmoCount=30
LockerAmmoCount=30
MaxAmmoCount=90

//Camera animation when the weapon is fired.
FireCameraAnim(1)=CameraAnim'Camera_FX.ShockRifle.C_WP_ShockRifle_Alt_Fire_Shake'

//Tells UDK what animation to use for alternative fire:
WeaponFireAnim(1)=WeaponAltFire

//MuzzleFlash/Particle System:
MuzzleFlashSocket=MF
MuzzleFlashPSCTemplate=WP_All.Particles.MuzzleFlash
MuzzleFlashAltPSCTemplate=WP_All.Particles.Blank
MuzzleFlashColor=(R=200,G=120,B=255,A=255)
MuzzleFlashDuration=0.33

//Where to place the crosshair:
CrossHairCoordinates=(U=256,V=0,UL=64,VL=64)

//Stop the camera from rotating:
LockerRotation=(Pitch=32768,Roll=16384)

//Weapon colour despite its texture (not used/important):
weaponColor=(R=160,G=0,B=255,A=255)

//Inventory details:
//Weapon Icon:
IconCoordinates=(U=728,V=382,UL=162,VL=45)
IconX=400
IconY=129
IconWidth=22
IconHeight=48

/**
 * Mac 10 Weapon Code by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTWeap_Mac10 extends UTWeap_Mac10Base;

defaultproperties
{
    //weapon skeletal mesh:
    Begin Object class=AnimNodeSequence Name=MeshSequenceA
    End Object
    Begin Object Name=FirstPersonMesh
      SkeletalMesh=SkeletalMesh'WP_Mac10.Mesh.Mesh_Mac10_1P'
      AnimSets(0)=AnimSet'WP_Mac10.Anim.Anim_Mac10'
      Animations=MeshSequenceA
      Rotation=(Yaw=-16384)
      FOV=60.0
    End Object

    //3rd Person Mesh:
    AttachmentClass=class'UTGameContent.UTAttachment_Mac10'
    Begin Object Name=PickupMesh
      SkeletalMesh=SkeletalMesh'WP_Mac10.Mesh.Mesh_Mac10_3P'
    End Object

    //weapon firing:
    InstantHitMomentum(0)=60000.0
    WeaponFireTypes(0)=EWFT_InstantHit
    WeaponFireTypes(1)=EWFT_InstantHit

    //Damage inflicted:
    FireInterval(0)=0.12
    FireInterval(1)=0.6
    InstantHitDamageTypes(0)=None
    InstantHitDamageTypes(1)=None
    InstantHitDamage(0)=10
    InstantHitDamage(1)=30

    //Sounds:
    WeaponFireSnd[0]=SoundCue'WP_Mac10.Sound.mac10sound_Cue'
    WeaponFireSnd[1]=SoundCue'WP_All.Sound.swingsound_Cue'
    WeaponEquipSnd=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_RaiseCue'
    WeaponPutDownSnd=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_LowerCue'
    PickupSound=SoundCue'A_Pickups.Weapons.Cue.A_Pickup_Weapons_Shock_Cue'

    //AI:
    MaxDesireability=0.65
    AIRating=0.65
    CurrentRating=0.65

    //How the weapon should fire:
    bInstantHit=true
    bSplashJump=false
    bRecommendSplashDamage=false
    bSniping=true
    ShouldFireOnRelease(0)=0
    ShouldFireOnRelease(1)=0
// How many bullets spent through primary/secondary fire:
ShotCost(0) = 1
ShotCost(1) = 0

// Positioning the first person mesh:
FireOffset = (X=0, Y=0)
PlayerViewOffset = (X=25.0, Y=5.5, Z=-7.5)

// Ammunition:
AmmoCount = 90
LockerAmmoCount = 270
MaxAmmoCount = 270

// Camera animation when the weapon is fired.
FireCameraAnim(1) = CameraAnim 'Camera_FX.ShockRifle.C_WP_ShockRifle_Alt_Fire_Shake'

// Tells UDK what animation to use for alternative fire:
WeaponFireAnim(1) = WeaponAltFire

// MuzzleFlash/Particle System:
MuzzleFlashSocket = MF
MuzzleFlashPSCTemplate = WP_All.Particles.MuzzleFlash
MuzzleFlashAltPSCTemplate = WP_All.Particles.Blank
MuzzleFlashColor = (R=200, G=120, B=255, A=255)
MuzzleFlashDuration = 0.33

// Where to place the crosshair:
CrossHairCoordinates = (U=256, V=0, UL=64, VL=64)

// Stop the camera from rotating:
LockerRotation = (Pitch=32768, Roll=16384)

// Weapon colour despite its texture (not used/important):
WeaponColor = (R=160, G=0, B=255, A=255)

// Inventory details:
InventoryGroup = 3
GroupWeight = 0.5

// Weapon Icon:
IconCoordinates = (U=728, V=382, UL=162, VL=45)
IconX = 400
IconY = 129
IconWidth = 22
IconHeight = 48

/**
 * Code by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTWeap_Shotgun extends UTWeap_ShotgunBase;
defaultproperties
{
// Weapon skeletal mesh:
Begin Object class=AnimNodeSequence Name=MeshSequenceA
End Object
Begin Object Name=FirstPersonMesh
SkeletalMesh=SkeletalMesh 'WP_Shotgun.Mesh.Mesh_Shotgun_1P'
AnimSets(0)=AnimSet 'WP_Shotgun.Anim.Anim_Shotgun'
Animations=MeshSequenceA
Rotation=(Yaw=-16384)
FOV=60.0
End Object

// 3rd Person Mesh:
AttachmentClass=class'UTGameContent.UTAttachment_Shotgun'
Begin Object Name=PickupMesh
  SkeletalMesh=SkeletalMesh'WP_Shotgun.Mesh.Mesh_Shotgun_3P'
End Object

//Weapon firing:
InstantHitMomentum(0)=+60000.0
WeaponFireTypes(0)=EWFT_InstantHit
WeaponFireTypes(1)=EWFT_InstantHit

//Damage Inflicted:
FireInterval(0)=+0.8
FireInterval(1)=+1.0
InstantHitDamageTypes(0)=None
InstantHitDamageTypes(1)=None
InstantHitDamage(0)=50
InstantHitDamage(1)=75

//Sounds:
weaponFireSnd[0]=SoundCue'WP_Shotgun.Sound_shotgunsound_Cue'
weaponFireSnd[1]=SoundCue'WP_Shotgun.Sound_shotgunsoundswing_Cue'
weaponEquipSnd=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_RaiseCue'
weaponPutDownSnd=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_LowerCue'
PickupSound=SoundCue'A_Pickups.Weapons.Cue.A_Pickup_Weapons_Shock_Cue'

//AI:
MaxDesireability=0.65
AIRating=0.65
CurrentRating=0.65

//How the weapon should fire:
bInstantHit=true
bSplashJump=false
bRecommendSplashDamage=false
bSniping=true
ShouldFireOnRelease(0)=0
ShouldFireOnRelease(1)=1

//How many bullets spent through primary/secondary fire:
ShotCost(0)=1
ShotCost(1)=0

//Positioning the first person mesh:
FireOffset=(X=20,Y=5)
PlayerViewOffset=(X=25.0.0,Y=6.5,Z=-9.5)

//Ammunition:
AmmoCount=18
LockerAmmoCount=18
MaxAmmoCount=54

//Camera animation when the weapon is fired.
FireCameraAnim(1)=CameraAnim'Camera_FX.ShockRifle.C_WP_ShockRifle_Alt_Fire_Shake'

//Tells UDK what animation to use for alternative fire:
WeaponFireAnim(1)=WeaponAltFire

//MuzzleFlash/Particle System:
MuzzleFlashSocket=MF
MuzzleFlashPSCTemplate=WP_All.Particles.MuzzleFlash
MuzzleFlashAltPSCTemplate=WP_All.Particles.Blank
MuzzleFlashColor=(R=255,G=185,B=75,A=0)
MuzzleFlashDuration=0.33

//Where to place the crosshair:
CrossHairCoordinates=(U=256,V=0,UL=64,VL=64)
//Stop the camera from rotating:
LockerRotation=(Pitch=32768,Roll=16384)

//Weapon colour despite its texture:
WeaponColor=(R=255,G=156,B=0,A=0)

//Inventory details:
InventoryGroup=4
GroupWeight=0.5

//Weapon Icon:
IconCoordinates=(U=728,V=382,UL=162,VL=45)
IconX=400
IconY=129
IconWidth=22
IconHeight=48

//Begin Object Class=ForceFeedbackWaveform
Name=ForceFeedbackWaveformShooting1

//Samples(0)=(LeftAmplitude=90,RightAmplitude=40,LeftFunction=WF_Constant,Ri
ghtFunction=WF_LinearDecreasing,Duration=0.1200)
//End Object
//WeaponFireWaveForm=ForceFeedbackWaveformShooting1

8.10.5 Weapon Relocation Classes

- **Unreal needs these abstract classes to locate the actual weapon classes.**

```cpp
/**
 * Used to locate the Desert Eagle weapon class.
 */
class UTWeap_DesertEagleBase extends UTWeapon
abstract;

/**
 * Used to locate the Mac 10 weapon class.
 */
class UTWeap_Mac10Base extends UTWeapon
abstract;

/**
 * Used to locate the Shotgun weapon class.
 */
class UTWeap_ShotgunBase extends UTWeapon
abstract;

/**
 * Used to locate the Wrench weapon class.
 */
class UTWeap_WrenchBase extends UTWeapon
abstract;
```
8.10.6 Third Person Weapon Classes

- **Used to assign 3rd person meshes and sounds to the character.**

```cpp
/**
 * Wrench 3rd Person by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTAttachment_Wrench extends UTWeaponAttachment;

defaultproperties
{
   // Weapon SkeletalMesh:
   Begin Object Name=SkeletalMeshComponent0
   SkeletalMesh=SkeletalMesh'WP_Wrench.Mesh.Mesh_Wrench_3P'
   End Object

   // Particles:
   DefaultImpactEffect=(ParticleTemplate=ParticleSystem'WP_All.Particles.BulletImpact', Sound=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_AltFireImpactCue')

   // Sound:
   BulletWhip=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_WhipCue'

   // Corresponding Class:
   WeaponClass=class'UTWeap_Wrench'
}

/**
 * Desert Eagle 3rd Person by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTAttachment_DesertEagle extends UTWeaponAttachment;

defaultproperties
{
   // Weapon SkeletalMesh:
   Begin Object Name=SkeletalMeshComponent0
   SkeletalMesh=SkeletalMesh'WP_DesertEagle.Mesh.Mesh_DesertEagle_3P'
   End Object

   // Particles:
   DefaultImpactEffect=(ParticleTemplate=ParticleSystem'WP_All.Particles.BulletImpact', Sound=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_AltFireImpactCue')

   // Sound:
   BulletWhip=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_WhipCue'

   // Corresponding Class:
   WeaponClass=class'UTWeap_DesertEagle'
}
/**
 * Mac 10 3rd Person by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTAttachment_Mac10 extends UTWeaponAttachment;

defaultProperties
{
    // Weapon SkeletalMesh:
    Begin Object Name=SkeletalMeshComponent0
    SkeletalMesh=SkeletalMesh'WP_Mac10.Mesh.Mesh_Mac10_3P'
    End Object

    // Particles:
    DefaultImpactEffect=(ParticleTemplate=ParticleSystem'WP_All.Particles.BulletImpact', Sound=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_AltFireImpactCue')

    // Sound:
    BulletWhip=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_WhipCue'

    // Corresponding Class:
    WeaponClass=class'UTWeap_Mac10'
}

/**
 * Shotgun 3rd Person by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTAttachment_Shotgun extends UTWeaponAttachment;

defaultProperties
{
    // Weapon SkeletalMesh:
    Begin Object Name=SkeletalMeshComponent0
    SkeletalMesh=SkeletalMesh'WP_Shotgun.Mesh.Mesh_Shotgun_3P'
    End Object

    // Particles:
    DefaultImpactEffect=(ParticleTemplate=ParticleSystem'WP_All.Particles.BulletImpact', Sound=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_AltFireImpactCue')

    // Sound:
    BulletWhip=SoundCue'A_Weapon_ShockRifle.Cue.A_Weapon_SR_WhipCue'

    // Corresponding Class:
    WeaponClass=class'UTWeap_Shotgun'
}
8.11.7 Weapon Ammo Classes

- **Used to define the properties of weapons’ pickup classes such as ammunition count and ingame mesh.**

```cpp
/**
 * Wrench Ammo Class by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTAmmo_Wrench extends UTAmmoPickupFactory;

defaultproperties
{
    // Amount of Ammo:
    AmmoAmount=0

    // Target Class:
    TargetWeapon=class'UTWeap_WrenchBase'

    // Sound:
    PickupSound=SoundCue'A_Pickups.Ammo.Cue.A_Pickup_Ammo_Shock_Cue'
    MaxDesireability=0.28

    // Mesh (set to Shockrifle's default mesh):
    Begin Object Name=AmmoMeshComp
        Translation=(X=0.0,Y=0.0,Z=-15.0)
    End Object

    // Collision Type:
    Begin Object Name=CollisionCylinder
        CollisionHeight=14.4
    End Object
}

/**
 * Desert Eagle Ammo Class by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTAmmo_DesertEagle extends UTAmmoPickupFactory;

defaultproperties
{
    // Amount of Ammo:
    AmmoAmount=30

    // Target Class:
    TargetWeapon=class'UTWeap_DesertEagleBase'

    // Sound:
    PickupSound=SoundCue'A_Pickups.Ammo.Cue.A_Pickup_Ammo_Shock_Cue'
    MaxDesireability=0.28

    // Mesh:
    Begin Object Name=AmmoMeshComp
        StaticMesh=StaticMesh'WP_DesertEagle.Mesh.Ammo_DesertEagle'
        Translation=(X=0.0,Y=0.0,Z=-15.0)
    End Object

    // Collision Type:
    Begin Object Name=CollisionCylinder
        CollisionHeight=14.4
    End Object
}
/**
 * Mac 10 Ammo Class by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTAmmo_Mac10 extends UTAmppickupfactory;

defaultproperties
{
   // Amount of Ammo:
   AmmoAmount=90

   // Target Class:
   TargetWeapon=class'UTWeap_Mac10Base'

   // Sound:
   PickupSound=SoundCue'A_Pickups.Ammo.Cue.A_Pickup_Ammo_Shock_Cue'
   MaxDesireability=0.28

   // Mesh:
   Begin Object Name=AmmoMeshComp
      StaticMesh=StaticMesh'WP_Mac10.Mesh.Ammo_Mac10'
      Translation=(X=0.0,Y=0.0,Z=-15.0)
   End Object

   // Collision Type:
   Begin Object Name=CollisionCylinder
      CollisionHeight=14.4
   End Object
}

/**
 * Shotgun Ammo Class by Tudor Pascu - tp60@sussex.ac.uk
 */
class UTAmmo_Shotgun extends UTAmppickupfactory;

defaultproperties
{
   // Amount of Ammo:
   AmmoAmount=18

   // Target Class:
   TargetWeapon=class'UTWeap_ShotgunBase'

   // Sound:
   PickupSound=SoundCue'A_Pickups.Ammo.Cue.A_Pickup_Ammo_Shock_Cue'
   MaxDesireability=0.28

   // Mesh:
   Begin Object Name=AmmoMeshComp
      StaticMesh=StaticMesh'WP_Shotgun.Mesh.Ammo_Shotgun'
      Translation=(X=0.0,Y=0.0,Z=-15.0)
   End Object

   // Collision Type:
   Begin Object Name=CollisionCylinder
      CollisionHeight=14.4
   End Object
}
8.10.8 User Interface Configuration File

- Used to define the informative messages displayed on screen when the weapons are picked up and used.

////////////////////////////////////////////////////////////////////////
// Added code by Tudor Pascu - tp60@sussex.ac.uk
////////////////////////////////////////////////////////////////////////

//Class Naming:
[UTWeap_wrenchBase]
ItemName="wrench"
PickupMessage="Wrench"

[UTWeap_DesertEagleBase]
ItemName="Desert Eagle"
PickupMessage="Desert Eagle"

[UTWeap_Mac10Base]
ItemName="Mac 10"
PickupMessage="Mac 10"

[UTWeap_ShotgunBase]
ItemName="Shotgun"
PickupMessage="Shotgun"

[UTWeap_Shotgun]
ItemName="Unreal Shotgun"
PickupMessage="Unreal Shotgun"

//UI Pickup Provider:
[UTAmmo_Wrench]
PickupMessage="I got me some Wrench ammo!"

[UTAmmo_DesertEagle]
PickupMessage="I got me some Desert Eagle ammo!"

[UTAmmo_Mac10]
PickupMessage="I got me some Mac 10 ammo!"

[UTAmmo_Shotgun]
PickupMessage="I got me some Shotgun ammo!"

// UI Provider:
[UTWeap_wrenchBase UTUIDataProvider_Weapon]
FriendlyName=Wrench
Description=Got a pipe that needs bending?

[UTWeap_DesertEagleBase UTUIDataProvider_Weapon]
FriendlyName=Desert Eagle
Description=50 caliber pistol. Woah!

[UTWeap_Mac10Base UTUIDataProvider_Weapon]
FriendlyName=Mac 10
Description=Spray and pray!

[UTWeap_ShotgunBase UTUIDataProvider_Weapon]
FriendlyName=Shotgun
Description=Got a shell-sized hole that needs plugging?

////////////////////////////////////////////////////////////////////////
8.10.9 Weapon Recognition Configuration File

- Tells the game to look at the new weapon classes and compile the relevant code.

```
/////// Added code by Tudor Pascu - tp60@sussex.ac.uk

[UTGame.UTWeap_WrenchBase]
Priority=1.0
bTargetFrictionEnabled=TRUE
bTargetAdhesionEnabled=TRUE

[UTGame.UTWeap_DesertEagleBase]
Priority=2.0
bTargetFrictionEnabled=TRUE
bTargetAdhesionEnabled=TRUE

[UTGame.UTWeap_Mac10Base]
Priority=3.0
bTargetFrictionEnabled=TRUE
bTargetAdhesionEnabled=TRUE

[UTGame.UTWeap_ShotgunBase]
Priority=4.0
bTargetFrictionEnabled=TRUE
bTargetAdhesionEnabled=TRUE
```

```
8.11 Screenshot Walkthrough

Fig 65 Building garage, beginning of level.

Fig 66 Stairs to reception.

Fig 67 Reception room #1.
Fig 68 Reception room #2.

Fig 69 Building main office.

Fig 70 Corridor to building secondary office.
**Fig 71** Secondary office.

**Fig 72** Corridor to fire escape.

**Fig 73** Stairs to second floor.
Fig 74 Corridor to first floor secondary office.

Fig 75 First floor secondary office.

Fig 76 Corridor to first floor main office.
Fig 77 First floor main office.

Fig 78 First floor private office #1.

Fig 79 First floor private office #2.
Fig 80 Vent to conference room.

Fig 81 Conference room.

Fig 82 Reception room #2 bridge.
Fig 83 Corridor to reception room #1 bridge.

Fig 84 Reception room #1 bridge.

Fig 85 Stairs to ground floor.