KINGSTON UNIVERSITY

Nikunj Givan

Decent into Madness:

Creation of a PC Video game.

Individual Project

CI3330

K0910896

03/05/13

Supervisor – Jarek Francik

2nd Marker – Andreas Hoppe

Table of Contents

CHAPTER 1: INTRODUCTION TO THE PROJECT

There are many games out there that provide a memorable experience, allow you to experience interesting scenarios and take part in awe inspiring action, these games are played by millions of people on a daily basis. The industry alone has exploded and is now a multibillion dollar industry rivalling those of music and film (Wolf, 2008). With this inspiration and from witnessing it first-hand as part of my placement at Jagex Games Studios (Cambridge, UK) the final goal of this Final Year Project is to take one step in producing a game that will invoke all of these emotions.

In this report chapter 1-4 will include setting out the aims and objectives, a critical discussion on previous successful games and an analysis of the agile methodology that this project will follow in addition to the analysis of technology and engines available to use.

Chapter 5 consists of various concept ideas and designs about the game and about modelling a main character which is a key feature of this project.

Chapter 6 will expand upon the design and will talk more in depth about the character design, how it was modelled, how it was rigged, how the animations where implemented and how the textures were applied to the mesh.

Chapter 7 will then focus on the implementation of the game code, were the world is built, various features of the game and how the model interacts with the game world.

To finish off chapter 8 will provide a critical review of the whole project with my own personal opinions on what was done correctly and what could have been changed, also included in this chapter will be the future plans, because as mentioned this is just the first step in creating a memorable game.

Throughout the project a main objective was to learn and experiment with the major tools in games development that are used in the current industry; these include not only programming but also using modelling/animation software, researching and implementing sound and concept designs etc… the reason for this is that I aspire to be a content developer therefore I would need a good grasp of conceptual and technical design when implementing a game.

The game produced by this project is called Decent into madness (DiM), it is meant to be played on the windows platform and has been written in C++ and compiled through visual studio 2010.

1.1 Aims and objectives

This section will help identify the major aims and objectives of the project, first outlined will be my personal aims and objectives but it is also important to bear the player in mind and recognise their aims and objectives when creating a game.

Personal aims and objectives

Aims:-

- My main aim is to gain new skills that would not normally be obtained in my course, this will add to my current skill set to create a complete game.
- To research all the tools and engines available to me as a game designer.
- Making the game impressive to show off to future employees/portfolio.
- Gain further insight into creating games, in particular the modelling/animations aspects
- To emulate the peers I worked alongside on my placement.

Objectives:-

- To model at least 1 game asset from scratch
- To animate at least 1 game asset
- To texture at least 1 game asset
- Create a suitable game in C++
- Design a navigation and interface
- Using object orientated language to make my code reusable.

Player aims and objectives

- To be engaged in the gameplay
- Enjoying the storylines
- To be able to navigate and play the game through a user friendly interface.
- To enjoy new features that they may not have experienced.

Chapter 2: Literature review

Video Games have come a long way from its humble 2D beginnings, with the advancements in technology and the transition into 3D the way games are developed and played has changed dramatically with many new and innovative features being implemented.

Through this chapter, key features of major games will be critically reviewed, all these features are relevant to the project and have provided inspiration to implement them into the project.

2.1 Choosing the correct game perspective

Game discussed: Assassins Creed (Ubisoft Montreal, 2007)

The perspective in which a game is played by the player influences a major part in the overall experience for the player and the fundamental design choices the developer has to make at the start, the two main perspectives are first person and third person, and whilst games often incorporate both these one is more prevalent.

Research by (Patrick Salamin, 2006) has found that players prefer one view for one action and another view for a different type of actions, when the game involves actions such as shooting or manipulating objects that would normally be done with hands; players prefer to be in the first person.

The third person is preferred when interacting with moving objects or the need to evaluate distance (i.e. jumping off a building), (Patrick Salamin, 2006) also believes the player has a greater connection with the main character when controlling them in third person

The story a game is also affected by the perspective it is played in, games lean more towards the RPG and single player genres prefer to play in third person, the designers will also prefer to tell the narrative in third person as an article from (Filipowich, 2013) states *"games must unite the player and the protagonist"*.

Taking into account the above statement, designers will need to make an important decision on how they frame their characters in relation to the scene when making a 3rd person game. A prime example of this would be the Assassin's Creed series developed by (Ubisoft Montreal, 2007), one major aspect of this game is that the player is able to climb on top and jump from

high buildings. To showcase the climbing actions the camera is placed close to the character perpendicular to it, similarly when the player reaches the top of a building the camera sweeps close to the camera, this gives the player a sense of scale and scope of the environment they are playing in.

One downfall in having this type of perspective in a game is that people affected by motion sickness can often get disorientated quickly, this is most noticeable in a first person game that often does not have a focal point to focus on, such as crosshairs in middle of screen or seeing a weapon in the corner, the term cybersickness has been coined for this phenomenon (Bruck S, 2009).

Another issue that can arise from choosing a wrong game perspective will be not displayed the scene correctly, particularly if the perspective is at a fixed angle, it may not be optimal for a situation, even a hindrance in some cases, for example if the game involves a player needing to travel across a narrow ledge around a mountain, having the camera fixed in third person would not allow them to properly view what is ahead, in this situation having the camera fixed side on to the character would be the correct solution.

2.2 Exploring an Open World

Game discussed: - The Elder Scrolls V: Skyrim (Bethesda Game Studios, 2011)

Exploration is another cornerstone feature in games, people in general love the sense of exploring to a new area, as people we would always be curious to know what is around the corner or over the mountain.

There are games that have taken this concept to huge lengths, entire virtual cities and even continents have been planned and constructed meticulously, these games tend to leave a lasting legacy on the player particularly on their first time playing through an area.

A great example game is ‘The Elder Scrolls V: Skyrim’ developed by (Bethesda Game Studios, 2011), where players traverse through a diverse range of areas that range from snowy mountains to lush jungles, 100’s of hours would be needed to complete every single aspect of the game, in addition to the main story line there are a multitude of activities that interact with the environment, such as hunting or picking herbs to enhance certain skills on your character.

Having a vast world to explore is also a clever time sink, often players would spend hours simply traversing around the terrain, to admire the scenery or in the hope they may come across something unexpected.

Replay ability also goes hand in hand with exploration, in a game such as Skyrim (Bethesda Game Studios, 2011), no two playthroughs would be exactly the same, often the second time round provokes a sense of nostalgia and anticipation that the player would be keen to repeat (Harris, 2007).

Another example is the Grand theft auto series, in GTA: San Andres developed by (Rockstar North, 2004), players can use a variety of vehicles early on in the game such as cars, trains, boats and even fly in a plane, when doing a mission the game will give a start and end point and it is often up to the player on what route they take, this sense of freedom is another reason why players would want to keep replying the game.

However open world games may not be for everybody, often there are multiple objectives to complete and it can lead to overwhelming a player, particular to someone who is new to the genre, quest/mission logs can spiral out of control if every single quest is picked up, and it could lead to confusion or a "broken" storyline experience.

To a explore every single aspect in an open world game, a large amount of time would need to be invested, this may turn of players who are looking for something more casual to just pick up and play for a little bit, this can lead to incomplete playthroughs, i.e. only the main core missions are done, or they may not even get to the end of the narrative.

2.3 Rewards of Questing/task completion

Games discussed: (various MMOs, Dead space (Visceral Games (formerly EA Redwood Shore), 2008)

Quest or task design is another major feature in developing a game, how the player undertakes them is an integral part of the overall user experience and effects the overall enjoyment of the game as it will be the main time sink.

Many exploration type games often have multiple tasks or quests that make up their activity list, things for players to do in the game that tie in with the game narrative; there are two main types, optional and mandatory.

Mandatory tasks or quest have to be completed in order to progress through the game narrative, whilst optional tasks and quest can be completely missed and does not always affect or is related to the main storyline of the game.

This is important to note as every single game has some tasks to do that fall into the two main category types, however some genres only implement one or another and still give the same experience.

Games such as Dead Space have a clear start and finish and a linear progression, due to this the designer can specifically implement features they want the user to experience such a main action sequence or a trigger that will always fire, i.e. walking down the corridor will that will always spawn a monster right at the end.

MMO's such as World of Warcraft (Blizzard Entertainment, 2004) have many small quest chains, with the majority of them being optional, in this game the main goal is for the player is to reach the maximum level and skills through doing quests (Billieux, et al., 2013), players can choose to skip all of the main quests and level up an alternative way, such as killing monsters for experience Here the developers have planned for multiple paths a user can follow; this has the benefit of the player trying a different path should they want to replay the game with another character adding replay ability. However often optional quests and tasks can be very repetitive and quickly become boring to the user.

2.4 Understanding Game theory – Normal form

The path or strategy a player undertakes through the game can be representative mathematically through 'normal form' which is a specification of game theory, (Myerson, 1997).

Using this we can specify the sequence of activities a player needs to do in order to follow a certain path and the decision they need to make at a branch, a table is them drawn up showing the payoffs (outcome) they would receive, this can be used to gauge every possible outcome expected and is useful to help develop algorithms as discussed in the journal by (Roughgarden, 2010).

This is mainly used in multiplayer games where the action of player one can directly influence player two's action when they observe player ones choice; however with modifications this can also be applied to single player game. (Myerson, 1997)

2.5 Getting good graphics in games – 3D modelling

The graphics of video games has come a long way from the days of 2D sprite drawing's, to current vector graphics and in the future full-motion capture.

A lot of emphasis is put on making the game as realistic as possible such as the terrain or characters, even in a fantasy or sci-fi setting; this often provides a lot of enjoyment for the players alone as they can immerse themselves and associate more with the game visually.

In the industry today there are three main ways designers are creating 3D assets from concept, these are the traditional Polygon modelling, virtually sculpting digital clay and using Motion Capture i.e. motion capture suits.

Each type of modelling has their own advantages and disadvantages but all ultimately want the same goal which is to get as close to the concept idea as possible, this will be analysed further in Chapter 3: design.

2.6 Object orientated approach

Object oriented design is a common approach to developing a game, commonly associated with programming it can also be applied to the modelling and animating aspects.

A major benefit of object oriented design is reusability as outlined in the conference proceeding (Lomow, et al., 1989) , in programming having an independent class or function that can be reused is very efficient as it can be plugged into different areas or into a new game, i.e. if there is code that adds AI to a mesh model, it can be used and applied to a different mesh model or in a different game that has its own meshes.

In animation the rigs are often reused for example, one rig would be made for a four legged animal such as a dog, when the modellers create another four legged creature like a cat it can use the same rig as the dog as their movement would be very similar and a lot of time can be saved.

However players are very quick to spot when content has been reused and it generally goes down negatively as it gives the impression of laziness on the development side since they expect new and exciting content with each new game.

2.7Designing storyboards

During the very early design stages when concept ideas are being thought off, designers often create storyboards to show the flow of their concept ideas help visualize the game world and actions (Moore, 2011).

They allow the designers to estimate the what emotions and actions a player would be doing in a particular scene of the storyboard, for example if the design involves jumping from one ledge to another, the storyboard would mention the context (such as running away), the action the player needs to perform (the jump), and the outcome (player lands successfully or falls down gap).

A simple method to classify each type of outcome a section of the storyboard produces is positive or negative, this can be tallied up and then referenced with the concept ideas, for example a horror game may want to provoke lots of negative emotions from the player such as fear and anxiety, they would then storyboard a scene and see if the actions done by the player match what they wanted to achieve.

One drawback of storyboarding is that the original vision for the game could get skewed considerably and if there is a lack of communicating between the implementation and design teams. Despite this it is a great method in helping with concept decisions; it can also be used to identify solutions to potential problems.

Chapter 3: Review of modelling tools and programming engines

There are many different types of engines that can be used to programme a game each specialising in a particular function such as rendering graphics or implementing physics (Ward, 2008), this chapter will give an insight into the major engines.

To implement these engines a development environment has to be selected and the programming language to work in, the most common in the games industry is C++ (Duffy, 2007)with companies using their own bespoke compilers, this project however will use Visual Studio 2010 (Microsoft, 2010).

The types of engines that are needed to build a game are; Graphics engine to render the scene, physics engine to show off any dynamics in the game, a sound engine for audio and a UI engine for in-game feedback and navigation of menus.

3.1 Graphics engine comparison

Graphics engine handle all the calculations needed to visually display what has been coded and is rendered on the screen, this includes the terrain, skybox, character models, shadow and lighting effects. This section will highlight the benefits and disadvantages of the graphics engines.

OGRE (*Object-Oriented Graphics Rendering Engine*)

OGRE is an open-source graphics engine (The OGRE Team, n.d.), which uses scene nodes to render objects, textures, lights and particles onto the screen in real time. Since it takes an object orientated approach not only can sections of the code be used in other OGRE projects but also can be exported to other C++ language programmes.

OGRE is also free to use if the user abides by the open source licensing conditions (Torus Knot Software Ltd, 2000).

Whilst OGRE is mainly a graphical engine; it supports an extensive list of additional libraries that can help incorporate other features such as sound and physics, (The Ogre Team, n.d.), in

some cases specialist wrappers have also been developed such as OgreBullet (Kuranes & Chaster, 2007).

The main drawback to OGRE is that it is a predominantly graphics engines, so in order to add features like physics or sound you would need to link additional libraries and set up their plug-ins correctly, this can prove to be too complex for beginner programmers.

Learning the Ogre SDK can also be a lengthy process as to implement many functions you would have to hard code them by defining the relevant parameters mentioned in their API. Many files that you would want to load will need to be exported and converted into one ogre can recognise therefore the development time is also slightly longer using this Engine.

Irrlicht

Irrlicht is a 3D engine written in C++ that can render graphics in 3D and 2D, in addition to graphics Irrlicht also has functionality that can implement physics and UI making it great all round package developed by (Gebhardt, 2003).

Irrlicht is considered a great start for beginners to start creating games due to the simple nature in implementing game features, i.e. whole levels can be generated with a few lines of code. It also includes basic physics and UI functionality therefore there is no real need to link it towards specialist libraries unless the developer is aiming for more complex collision detection or UI.

The engine contains multiple built in importers that can load common files (Gebhardt, n.d.) So there is no need to convert them as you would need to in OGRE making Irrlischt faster to use.

Whilst OGRE has an extensive library list that can be used to add additional features, for Irrlicht this is more limited, therefore it is also considered less powerful in its capabilities.

3.2 Sound engine comparison

Irrklang

Irrklang, is a high level 2D and 3D cross platform sound library, as such it can be adapted to many different graphics engines which includes Ogre 3D, this is one of the main reasons this engine was chosen. Another benefit is that it can play a larger collection of sound files that includes, .wav, .ogg, .mp3, .flac, .mod, .it, .s3d and .xm, as opposed to OpenAL.

OpenAL

OpenAl is also cross-Platform sound engine, unlike Irrklang it supports a 64bit Library. OpenAL also uses three objects, a listener, a source and a buffer, therefore it is slightly more complex to use than Irrklang where you simply need to create an Irrklang device and attach a source to it. However the benefits of these additional objects, is that OpenAL can process and project more complex sounds.

3.3 Modelling tools available

For a modeller and animator, they will need tools that can accurately manipulate objects in 3D space, these needs to be updated in real time. Similar to game engines many companies would use their own bespoke modelling or animating software to create their 3D objects, however there are two main tools that a student can access via their educational institute, they are Autodesk Maya (Alias Systems Corporation, 1998) and Autodesk Mudbox (Autodesk (Skymatter Ltd, prior to acquisition by Autodesk), 2007).

Both these tools were selected because they have an easy to use interface and is a good start for beginner modellers where an understanding of complex maths is not needed, everything is made on screen via manipulators and brushes that can be controlled with a mouse or graphics tablet.

Chapter 4: Analysis & methodology

This chapter will analyse the features discussed in the previous chapters that has influenced the design decisions in the following chapters, the methodology that the project will follow will also be discussed and justified.

4.1 Mission statement

To create a PC video game that contains a high quality model that has been crafted with passion, a free flowing narrative that draws players in and a mysterious environment that holds many secrets.

4.2 Technology analysis of PC & console

One of the first decisions in the project was to select the environment the game will be developed in, particularly the language and the platform.

There are four main platforms to create a video game on in the current industry, they are the console, handheld, mobile and PC, each have their own pros and cons that will be explained in this section.

Developing on consoles

Video games consoles contrary to popular belief have been around almost as long as the PC, with the first generation being created in the 1950's as mentioned in the journal by (Nyitray, 2011), since then 3 major companies have emerged to continue to make consoles to this day, which is considered the 7th generation, they are the Sony Playstation 3, Microsoft Xbox 360 and Nintendo's Wii.

- Nintendo Wii

Development on the Nintendo Wii can be very innovative due to their unique type of controller that allows for a lot of creativity, whilst not considered as powerful as their rivals (Thurrott, 2010); the games produced on the Nintendo Wii tend to be more social and cartoony. However creating games for this console as part of a project is very restrictive, with

the main documents and coding environment being bespoke and limited to Nintendo employees.

- Sony Playstation 3

To be able to develop for Sony you need to obtain the relevant licences (Sony Computer Entertainment, n.d.), this is readily available to educational institutes; in return access to their development centre is gained which contains documents and help forums. The language used is C++ and it can be coded in various windows compilers, however there is a lack of tutorials making it hard for beginners to grasp if they have limited knowledge with C++ and object orientated programming. However the games produced can be very advanced and complex as the playstation has the most powerful computing power out of their rivals (Thurrott, 2010).

- Microsoft Xbox

Developing on the Xbox is readily available to students through their XNA Games Studio, however payment is involved if you want to run it on the console. many example games created by users are readily available on their arcade store. The programming language involved is C# which is considered more limited than its playstation counterpart that uses C++ (Suess, 2006). There is also a lot of help for beginners with a wide variety of tutorials and help documents available online.

Developing for a Handheld

Developing on the handheld consoles is very similar to its main console counterpart such as the Playstation Portable and Playstation 3 both use the C++ language, there are two main differences however. The first is the processing power of handhelds is considerably smaller, this effects the graphics of the game as models would need as few polygons as possible making them very blocky. Battery life also has to be taken into consideration, too much processing of data inefficiently can drain the battery fast defeating the purpose of making a handheld console free roaming and truly 'hand held'. The second consideration is the navigation and movement controls, the common setups is to have character movements mapped on the directional pad that is operated by the left hand, and have action buttons mapped to buttons that are pressed on by the right hand, in addition there are shoulder buttons and even buttons behind the handled.

Developing for mobiles

More and more games are starting to be developed for the mobile phones, simply because the target audience is very large with most people owning a Smartphone of some sort (Haggerty, 2012), these games do not tend to be long or contain complex narrative, rather full of short mini games that take a few minutes to complete. One innovative feature of mobile phones is that you can use the inbuilt gyroscope as a means of navigation or movement within a game, for example, when you physically tilt the phone to the left, the on screen character will also move towards the left. Mobile games can also be easily commercialised, ad space can be sold or some sort of in-game store that unlocks bonus features can generate revenue.

Developing for the PC

Development on the PC is considered the most traditional, but due to this there are a wide array of languages and support provided making development suitable for all skill levels, this will be the platform the project will be developed on and the language chosen will be C++, With C++ being norm in the games industry it has the most potential to refine and further develop the skills that will become relevant when starting a career.

4.3 Planning the specifics of the game

Open world level

Looking back at the research done on open world games, the aim will be to replicate on a smaller scale for the project, to have an open environment in which the player can travel around and do various objectives. This would give the impression of freedom as there should be no restrictions in the movement, only the very edges of the world will be out of bounds with most of the game world in a self-contained large rectangle, where the terrain, graphics and game logic will be in.

This would also allow a lot of freedom to work with camera angles and player movement, another positive is the objectives of the game can follow the simple questing design also mentioned in the literature review chapter, were items/objectives will be scattered around the open world.

Since the open world terrain would be on a smaller scale a decision was made to only implement one type of terrain that would be ‘closed off’, this means the world would need to be self-contained and the boundaries of the edge clearly defined by the terrain. This would

rule environments such as grassy meadows or the open sea, instead a valley like terrain would be most suitable for the project, the hills would serve as the boundary of the game world and the actual valley would be where the player plays the game.

Within this Valley there would also be a mini sub-valley that would contain a boss like creature, the aim would be to navigate through the terrain and reach the boss and defeat it, from a narrative standpoint it could be a long lost temple.

In the end it was decided that the desert valley with a sandstorm would be ideal for the aims the project had in mind, this would allow certain areas to be obscured by the sandstorm which will add a sense of mystery, and it is also different from the norm.

Fully modelling a main character

Once the environment and terrain is built it would be time to populate the scene with objects and characters to bring the game alive.

Having researched about the modelling aspect of game development it was decided that the main character would be fully modelled from scratch, as this would be the perfect opportunity to make a challenging model and gain the skills and knowledge of this aspect. Creating a custom character from scratch is a lengthy time investment; particularly with the aim to get it as realistic as possible therefore only one model will be modelled from scratch, the design and implementation of this is described in detail in chapters 5 and 6.

However there are a couple key features that need to be analysed before modelling or designing can begin, they are Polygon count and N-gons.

- polygon count

The mesh will be modelled with numerous polygons, the aim is to keep this count as low as possible whilst having as much detail as possible, the more detail needed in the mesh the more polygons I will need. The count will have to be fairly low simply because when plugged into the game engine it will directly affect the performance of the game, i.e. very high polygons count will cause the game to slow down and 'lag' which is not desirable, there will also be other models such as terrains and enemies that will also contain polygons. For the main character it will be reasonable to have a high polygon count as it will be constantly close the screen that the player will see.

- N-gons

An N-gon is a 5+ sided shape, when modelling the mesh should contain only 4 sided polygons or triangles, this is because having n-gons will negatively affect how the mesh deforms when animating it. It will also cause textures to artefact and distort and cause lag in the game as the game engine only supports Triangles (it will automatically convert 4 sided polygons into tris, having n-gons will complicates this conversion).

Main character will be fully animated

Once the modelling of the main character is completed the next step is to animate the model and bring it to life, here the research done on bone structure and muscle kinematics will be applied to make it move in a realistic manner.

Since this is also large time sink and usually done in a spate department, the model will only contain a couple of important animations with two currently planned, they are the run cycle to show movement of the main character, and the attack cycle for when combat has been implemented.

Further animations can include the model being idle when the player is not inputting any commands or block which would which would be a part of the combat, this will also allow for different game mechanics to come into play. Finally the last main animation would be to jump, but as there is no core game mechanic revolving around the character jumping in the game, this is of a low priority.

There are two main parts in getting a model to animate; the first is to rig the mesh, this involves creating a skeleton and joints, the number of joints does not always have to accurately represent real life, it simply needs enough joints to serve as ‘control points’ of the mesh.

Once the mesh has been rigged the skeleton is bounded onto the mesh, with each joint controlling a certain surface area of the mesh that it will deform, this is important to note as if the rig is not bound correctly the mesh may distort drastically making its animation difficult to implement, i.e. a joint that is incorrectly oriented will rotate the wrong way than intended.

Once the rig has been correctly bound to the mesh it is time to animate the model. Here various artistic techniques come into play, traditionally animation was done on flip books

where the drawing would be ever so slightly different on each page and when the user flips through the page it gives the illusion of a moving picture (Flippies Inc., n.d.).

Animating a 3D model is very similar to this, instead of a book Key frames would be set at certain time intervals, the model and its joints will be posed in a certain way, and there can be a number of key frames the amount depending on how accurate the movement to feel when the animation is played back.

For game models the animations are in short chunks as they are simply looped through over and over again as such the run and attack animations that are planned for the project will only need to be 4-6 seconds in length, when the player presses a movement key the run animation will continuously loop, the same applies for the attack animation when the attack button is pressed.

Planning the texturing of model

Once the mesh is finished the character will need to be textured, this involves applying a texture or colours over a section of the mesh, UV coordinates are used to identity which section of the mesh is due to receive the texture.

Each vertex of the mesh in addition to having a 3D coordinate also has a UV coordinate that is in 2D, this shows where the vertex is on the mesh.

This 2D representation is also called a UV Map, it is very similar to unfolding the mesh and laying it out flat as described in the book by (Mullen, 2012) , this is also considered another time consuming aspect of creating a model as the default UVs generated when a mesh is made are not suitable and causes textures to distort when applied.

In order to apply a good texture to the mash in the project I will also have to map out its UVs, here I use a test texture that consists of a series of boxes, this is then projected onto the mesh, the aim is to keep the boxes as square as possible when it is on the mesh, where it is distorted will also mean the texture will distort here when applied, so it is very important to map the UV’s correctly.

(see Fig 1. In appendix for an example fo a test UV texture)

Populating the world with misc. scene objects

The project will also include many other peripheral scene objects that will bring the terrain to life. This will include objects like rocks, half buried statues and some npc character.

These will all be outsources from third party sources, so they will already be modelled, textured and animated and would just have to be plugged into the game world.

However there are a few additional objects that may not available on these model marketplaces and as a result will be built from scratch, but they will have considerably less complexity than the main character, this will mainly include the temple door that will be used to bar of a section of the terrain until the player reaches a certain point in the game as described in the design chapter.

Adding a 3rd Person camera

The game will also need to be played at the correct perspective to show off the model in full.

The initial perspective will be in 3rd Person, as mentioned in the literature review chapter having a 3rd person point of view in a open world terrain allows the player to associate and gain a sense of scope of the environment, there will also be no movement involving the hands.

However one dilemma that this perspective presents is that the front half of the model will not be seen since in fixed 3rd person there is no incentive to see the front of the model, as a result there are plans to make a more dynamic camera.

There are two main ways this could be implemented; the first is to have multiple cameras at certain angles in the scene that can be turned on an off, this could be on a specific condition such as being in the vicinity of the camera or at a press of a button, whilst this gives the advantage of the optimal view for that section of terrain, it will still feel quite restrictive as it would be in a fixed position, also it may disorientate the player in terms of how to move their character when viewing from a different angle.

The second method is to detach the camera and have it free roaming, this has the benefit of the player making a conscience choice on which angle they feel is best when playing the game, further to this the developer can add more features that can only be used with this type of camera system, for example high up in a alcove there could be treasure, the only way to

see this would be to fly the camera up to spot it, this would ordinarily have been missed if the camera was in static 3rd person.

This also allows the player to dynamically view the scene, it will also showcases the model that has been created from all angles, the player may also wish to use this leisurely, i.e. perching the camera to get a wide view of the character and the scene making a picturesque moment that the player is likely to remember.

Allow for full navigation along the terrain.

The project will include full 3D movement via pressing keys, as it will be on the PC it will utilise the computer keyboard layout with the most common setup being on the arrow keys, For the camera, which is planned to become detached and free roaming the movement will be bounded in a similar manner but to the keys WASD, the switch between the movement and camera movement should be seamless and revert back based on which keys are pressed. For example, pressing the up arrow key it will move character forward, when they press W the character stands still and camera starts to move forward only, but when the up arrow key is pressed again it would snap back to the character and move forward again.

Narrative told from main character perspective

The project will also need some sort of narrative, taking into account the type of terrain is a desert valley and that the player is controlling one main character; it would make sense to tell the narrative from the main characters point of view. Through text and dialogue placed on various objects like rocks it would explain the reason why the player is there in the first place and what they must do in order to 'end' the game. With the terrain having a sandstorm affect it will enable certain objects and areas to be just outside the field of vision and be obscured; this will then add the sense of mystery to the narrative.

Adding enemies to increase difficulty

With the main aim of the game being to explore the terrain and experience the narrative there needs to be some difficulty that would try and prevent the player from reaching their objectives. Therefore the game will also include some enemies that will be suited to the desert environment, they will attempt to attack the player and try to kill them and should the player die the game will end.

The game will also include a boss type enemy that would serve as the main objective to reach and defeat it, this enemy will be located in the obscured area that is shrouded by fog and blocked off by a temple style wall, when the player reaches the wall they will need to find a way through to reach the enemy, this is further explained through concept sketches in the design chapter.

There will also be little scarab critters that spawn in an erratic manner, this will help further populate the game world, these scarabs would have a patrolling pattern and the aim for the player would be to either kill or avoid these scarabs.

4.4 Analysis of modelling techniques

As touched upon in the literature review there are there are three main types of modelling techniques that would be suitable for creating my model, to reiterate they are traditional Polygon modelling, virtually sculpting digital clay and using Motion Capture, this section will analyse these techniques and select a suitable one to use as part of the project.

Motion Capture

The first type is motion capture, this is commonly done using motion capture cameras or motion capture suits, an action or person is recorded which is then digitally translated and viewed on 2D or 3D on the computer, this technique has the advantage of modelling and animating very complex features such as facial movements that often require very subtle movement to make realistic, another scenario would be to model a humanoid type fantasy figure, one of the most famous being Gollum in the Lord of the Rings trilogy (Scott, 2003), where an actor replicates the movement.

Another benefit is the speed in which animation can be done, since the data received is almost in real-time there would be no need to set and measure key frames, then replaying it back to check if it is realistic which can be time consuming.

On the other hand motion capture equipment is very expensive and often limited in its access making it only viable for large companies. There are also a couple of limitations that reduce its effectiveness; the first is the size of the area that is being captured, i.e. if you wanted to capture a action sequence involving multiple people, you would only be able to capture a few at a time depending on how big the room may be. The second drawback is you cannot capture

anything unrealistic; everything needs to obey the laws of physics, and in many cartoon type animations there is often a large amount of exaggeration that break these laws (Ant, 2007).

Virtual sculpturing

The second type of modelling is virtual sculpturing, here you are presented with virtual clay that you can manipulate with the mouse or tablet sculpting the mesh into the desired shape, no animation can be done however, so it is purely modelling. The benefit this has is you can add in fine detail without fearing of breaking the mesh i.e. it will ensure there are no N-gons which as mentioned are very undesirable in modelling. The main drawback of this technique is the mesh will have a very high polygon count in, this can be increased even further depending on the detail needed, and this will then cause performance issues when the mesh is plugged into the game such as lag or screen tearing.

Polygon modelling

The last type is traditional polygon modelling which then leads to traditional key frame animation, here you create and draw individual polygons and have full control over the placement of faces, edged and vertices. This is considered a good start for beginner modellers as the basics are learned on how polygons work and how to create a 'clean' mesh, the main aim when making a model for a game is to keep the polygon count low as possible. This will be the selected technique that will create the model as being a programmer modelling is not a skill I have gained, and this is a good start for a beginner. In addition once the model has been created the animations will be set using key frames.

4.5 Moscow requirement list

With the platform, language, modelling techniques and detailed description of the game features outlined, the next step is to draw up a requirements list for the project, for this Moscow analysis will be used, it help outline key features that need to be implemented and depending on which section a feature in, the time needed to be invested in it can be estimated.

Must do –

- Model a realistic character mesh

There needs to be something for the player to control and it will be the main protagonist of the game, this will be custom modelled from scratch, it will also be the main focal point that

the camera focuses on and the point of view the narrative will be told from, therefore must be something tangible that the player can see which is this main character.

- Rig the mesh.

In order for the model to have life it will need to be animated, before that however a bone structure will need to be implemented otherwise known as a rig, since it will be a human character a human bone structure would be sufficient, this will involve rigging the legs, arms, body and head. These bones are a must do on the rig, the other aspects of a human skeleton are optional.

- Animate the mesh

Once the bone structure is in place via the rig, two main animations must be created, one for when the character is running and a second for when the main character is attacking, this will give nice visual feedback to the player that an action is being carried out.

- Texture the mesh

The main character must be textured to make it look like a game character as opposed to a plain mesh or with a text texture applied to it. There should also be a minimum amount of distortion so the UV's of the mesh must be laid out carefully.

- Have mesh functioning in game world

Once the model mesh is created, it must function correctly in the game world, this means that animations must play and loop correctly when its corresponding button is pressed, it must also be able to traverse the terrain correctly i.e. not falling through the floor when moving.

- Create a full game environment

A terrain and skybox must be constructed to represent the game world, the terrain must resemble a desert valley with high hills along the boundaries of the game world, the skybox must be as seamless as possible and give the impression of a real sky moving across. The empty 'void' beyond the skybox and terrain must not be visible to the player.

- Implement 1 extra type of enemy

There must be at least 1 extra type of enemy in the game; this will allow the game to have more of a narrative and end goal that the player can strive towards, combined with the combat and collision it must also be killable.

- Implement combat

A form of collision detection must be implemented; this will coincide with the attack animation and have the enemy correctly despawn when it is 'killed'. This should then also add a big of challenge and reward to the player and enhance their experience.

- A UI that tells player what to do

A UI implemented, here the UI will tell the player which keys are used to move and perform certain actions or functions, they must not be obtrusive and so located along the side or top of screen so they are out of the way. The UI must also show messages when a action is performed, this is good feedback to the player.

The narrative must also been shown in the UI in the form of text on the screen when an object is clicked.

Should do:-

- Have block and jump animations

Additional animations should be implemented such as block and jump this will then allow for other features of the game to be implemented such as obstacles to jump over or more complex combat involving blocking.

- Create an intro menu

Into menu should be created as it would give the game a more finished and polished feel, with a start and options button as seen in many other games at the very start, in addition there should be a pause button that will freeze the game state and resume when the player presses the resume button.

- Add music

Music should be added to the game, in particular a sound track and some ambient sound that will continuously play in the background, should they wish the player can turn this off or on at the will. This would add a lot of atmosphere to the game and allow the player to be more immersed into the game play and scenery.

- Implement 2nd type of enemy

A second type of enemy should be implemented to add a bit more variety to the game; this should be in addition to the end boss and be roaming around the scenery, thus being more of a nuisance than pivotal to the progress of the game. This second type of enemy should spawn slightly randomly and in a random pattern to add an additional layer of variety to the game every time it the game is started up.

Could do :-

- Have idle animation

Having an idle animation play when no buttons are pressed would further add to the realism that it is a living breathing character you are controlling as opposed to a character that only comes to life when a button is pressed, this will further allow the user to relate to the main character.

- Have 3rd type of enemy

Having a 3rd type of enemy would be the limit in terms of a variety of NPC's in one area in my opinion, so a 3rd enemy could be added and weaved into the narrative if it makes sense for the enemy to be present in that environment.

- Have powerups/hp

Having this would add additional objectives for the player to complete, it will also enhance the combat experience the player will have, this will need the additional animations of block in order to function.

Won't do:-

- Make it compatible for a different platform

The game can be ported and adapted to make use of a different platform, this would not only develop additional skill as the syntax of the code will be different, but it will also open up more of a target audience that will play the game.

- Add networking

Multiplayer functionality can also be incorporated into the game, this could be in the form of a 2nd main character that accompanies the first, or the enemy boss could be controlled by the player in the form of a arena style area.

- Different movement

Different input methods can support in addition to the standard keyboard, this includes joysticks or specialised gamepads, this has the benefit of making certain actions more fluid and realistic, i.e. flying would work very well with a joystick.

4.6 Understanding agile development methodology

The most suitable method to developing a game with all these features is the agile software development methodology; here the emphasis is on constant iterations and small incremental developments of a task called sprints (Layton, 2012).

These sprints are given a time frame in which to complete and it is then evaluated afterwards, changes that usually arise are then added to the next iteration of the task and it is evaluated again, this cycle repeats until the release date of the game or until the development team are happy with the results.

An advantage of this in game development is that bugs can be identified very quickly and work can begin on fixing them as soon as possible, such as an enemy not spawning or a graphical glitch that distorts a model.

One feature that is commonly being integrated into this methodology is beta testing by the public, in the past this was done internally and was just another iteration cycle in the development of the game. Nowadays it is publicized more and is open to the public or chosen select members that receive a beta key, the role is still the same, to find and fix bugs or flaw before the game is released.

This would have been very restrictive or not possible if the waterfall methodology was used as bugs would needed to have been fixed after the game was released in the form of multiple patches, this generally does not go down well with the players as they would feel they have paid for a half-finished game and thus not likely to play another game from the same studio.

Therefore choosing the right methodology is not only affects the design process but also the commercial success to the game.

However there are drawbacks to using agile, the main one being if the project has a fixed budget and timeline, with multiple iterations and sprints this can cause projects to go over budget or miss a deadline. Another minor drawback is when the client asks for a very specific product here the waterfall methodology would be more suited, however this is rarely the case and modifications to the original design occur most of the time (Finlay, 2010).

4.7 Analysis of design process through placement

During my Placement I gained a practical insight on how agile methodology was applied in the industry, as they created the online MMORPG RuneScape (Jagex, 2001).

Their process was split into three main aspects, Design, Implementation and testing, each aspect has a main brief that contained a list of tasks and dates to show when it is due to start and end, as it is planned months in advance.

The design brief is the first to be created, here the content developers, artists, modellers and sound designers decide the concept ideas of a new quest or feature they want to implement, it is then translated into the implementation brief where the programmers make the concept functional in the game.

If a feature cannot be implemented then it will go back to the designers and another iteration of the task is made to try and resolve the issue.

The testing brief is mainly done by the QA team who play test the code, this brief mainly contains feedback and suggested changes that is passed on to both the designers and implementers.

This is a very fluid and flexible pipeline that allows them make frequent updates to their game while keeping it very polished and bug free.

Chapter 5: Design

In this chapter the specific designs of the game will be mentioned, the specific concept ideas and how they will be incorporated into the game. There is a strong base in place after the research done in the literature review and the information received from the analysis in previous chapters. Tips and tricks learned from my placement will also be applied such as the recommended tools mentioned by my former employers.

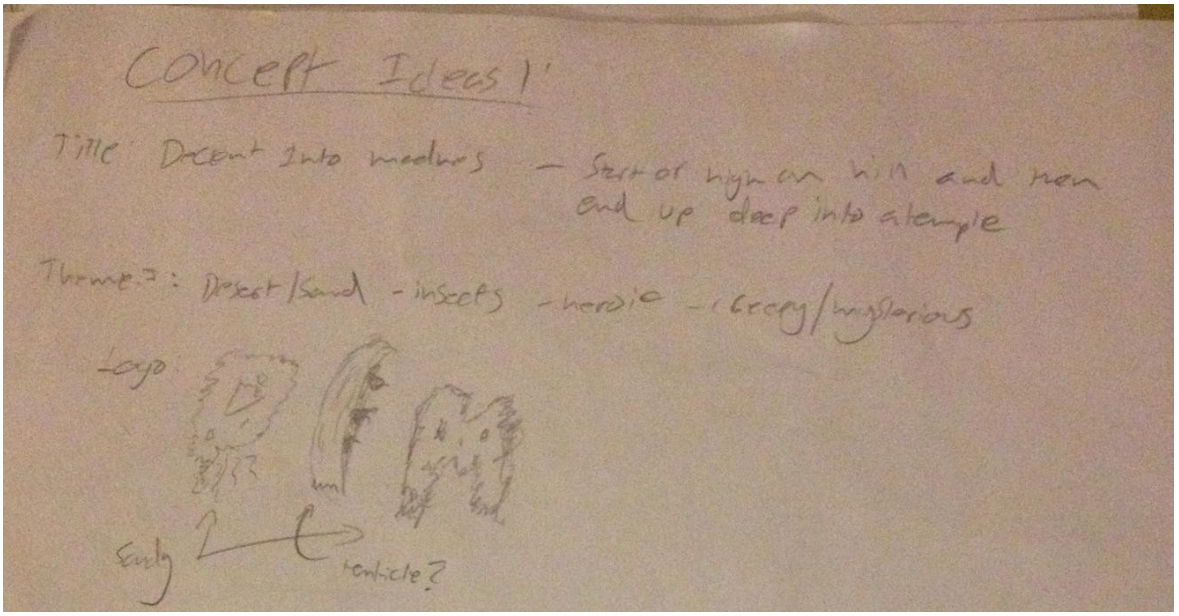
Design brief - Concept ideas

The first step is to draw up concept ideas, these are very important in outlining the 'vision' of the game, such as how do the characters look, how do the enemy's look, how the terrain will be set up in the game, how certain viewports will look in the game.

With this in mind I drew up some early concept ideas on paper (full scan see fig, 3-5. in appendix), trying to answer some of the questions mentioned above and keeping in mind the information gained from previous chapters.

5.1 Concept Sketches

- **Determining Game Title and setting out the specific themes**



The very first idea was to think of a working project name and its logo, the aim in the name was to portray a very brief narrative to the player who may not understand it at first but will once they have finished the game.

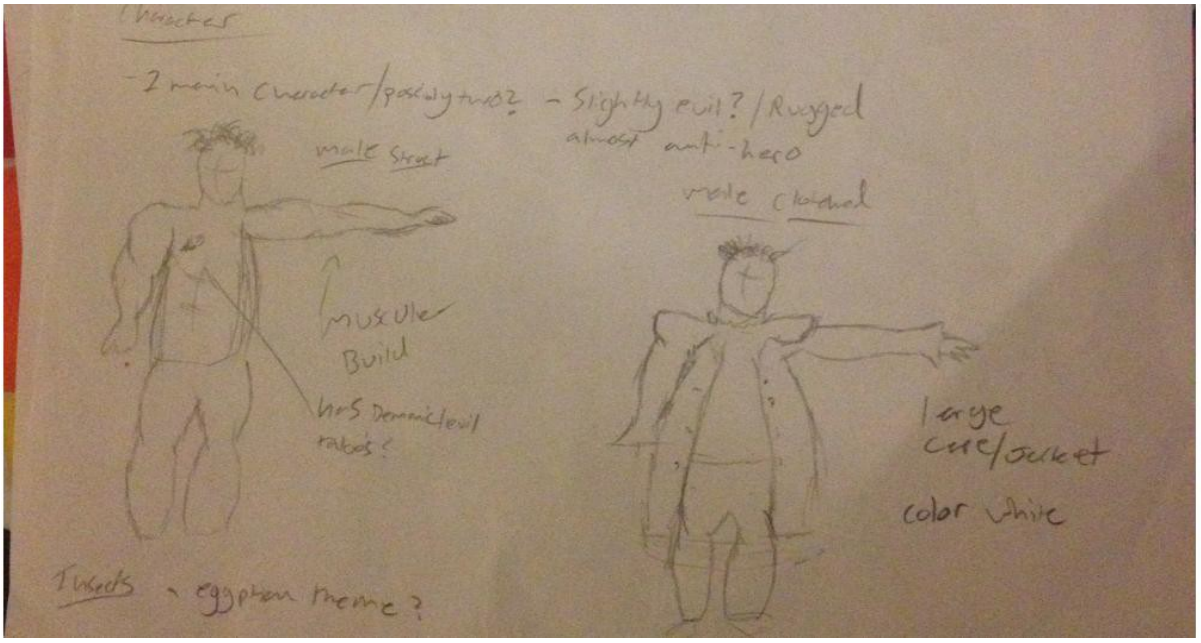
The title chosen is ‘Decent into Madness’ (or abbreviated to DiM).

The title briefly shows how the journey of the player will play out, they will start off at a slightly raised point like a small hill and gradually make their way down and navigate their way to the end where there will be an enemy initially hidden, this is the ‘madness’ part of the title.

Also in this first part of the concept ideas was outlining the themes that would be running through the game and narrative. The main theme is the environment in which the player is located which will be a desert/sandy valley, this hen would be very fitting place for insects to be enemies such as scarabs or scorpions. In this environment there will be a sandstorm effect, this will intentional obscure scenery and give the game a creepy feel where the player is encouraged to reveal its mysteries.

- **Character Design**

Now that the themes have been outlined, the character development could begin, the first character designed was the main character who would also serve as the main protagonist in the main narrative of the game. Since this will be the model that would be built from scratch as determined in the analysis section above, the advantages and pitfalls had to be taken into consideration such as not to make it to complex or it would have a high polygon count.



Above is a scan of the concept art sketch of the main character in two different styles, they both have a similar build in terms of a humanoid structure and height and width. The main difference is the outfits of the character; on the left he has a more evil feel with a very

muscular build and tattoos suggested on the upper torso, on the right however they have a jacket giving it a more modern feel.

- **Enemy Design**

The game will also be populated by multiple enemies, three different types was found a ideal for a single setting environment as determined in the analysis, with at least one being implemented from the Moscow analysis.

Above are the three different types of enemies that were sketched, they all keep to the desert theme of the game and were inspired by Egyptian creatures such as the scarab and desert wasps, the third sketch on the right is depicting the boss of the game who will have a more humanoid figure but with the modification of claws to instantly tell the players this character is to be avoided or killed.

- **Combat items designs**

With a main character and enemies designed the next step was to connect them and draw ideas on how the combat may look like such as what items may come into play.

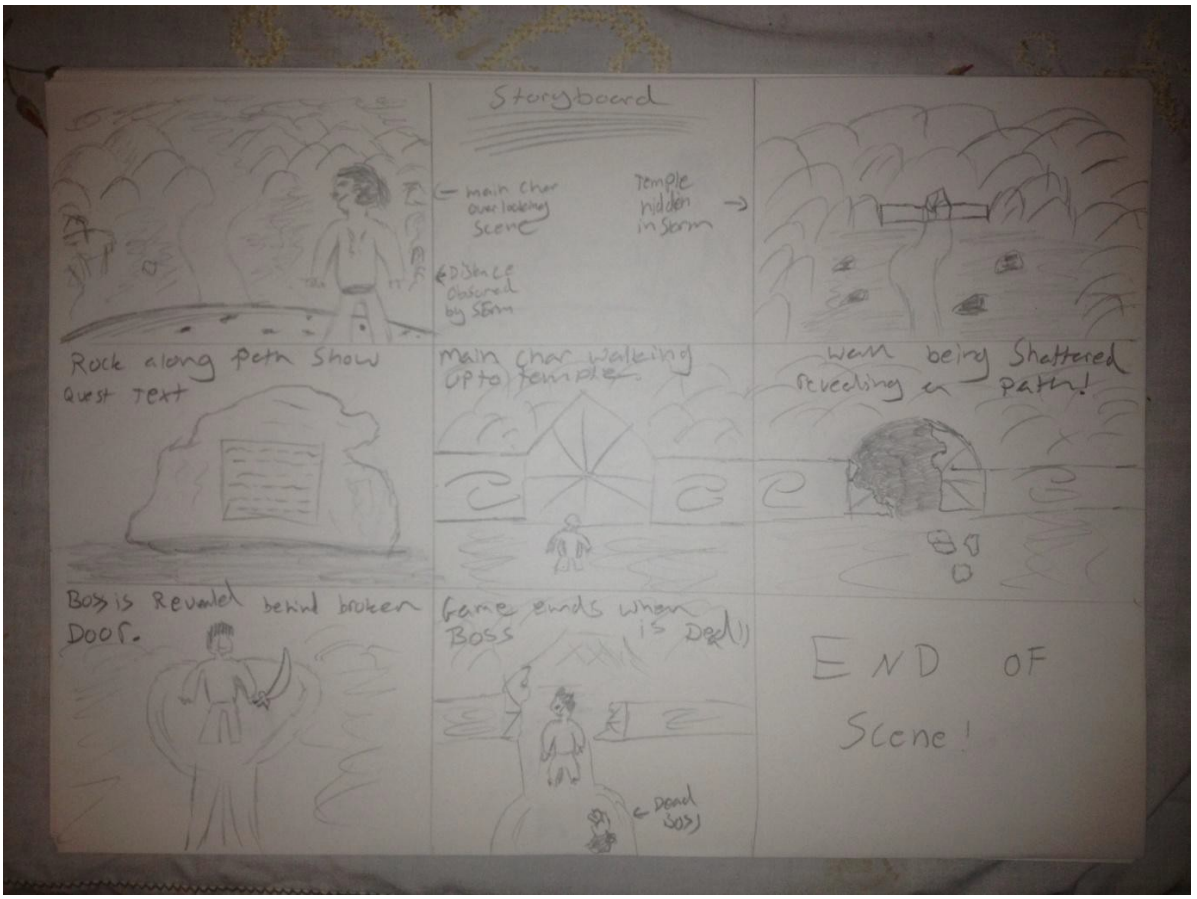
Above are the designs for the basic items that the player may need in combat, this includes a shield for protection and a sword to attack, also dotted around would be potions that can give buffs and recover HP.

- **Terrain Design** (see fig 5. in appendix)

There were two sketches designed in fleshing out the game world, the first is how the desert valley would look like, it will be sounded by high hill along the borders that will restrict the player to the middle game area. The second sketch is of a temple door that would hide the end boss and how it is expected to look from the player's point of view. In the aerial sketch the location of the start of the player is shown on a little hill towards the right of the terrain and the temple is on the left hand side.

5.2 Storyboard

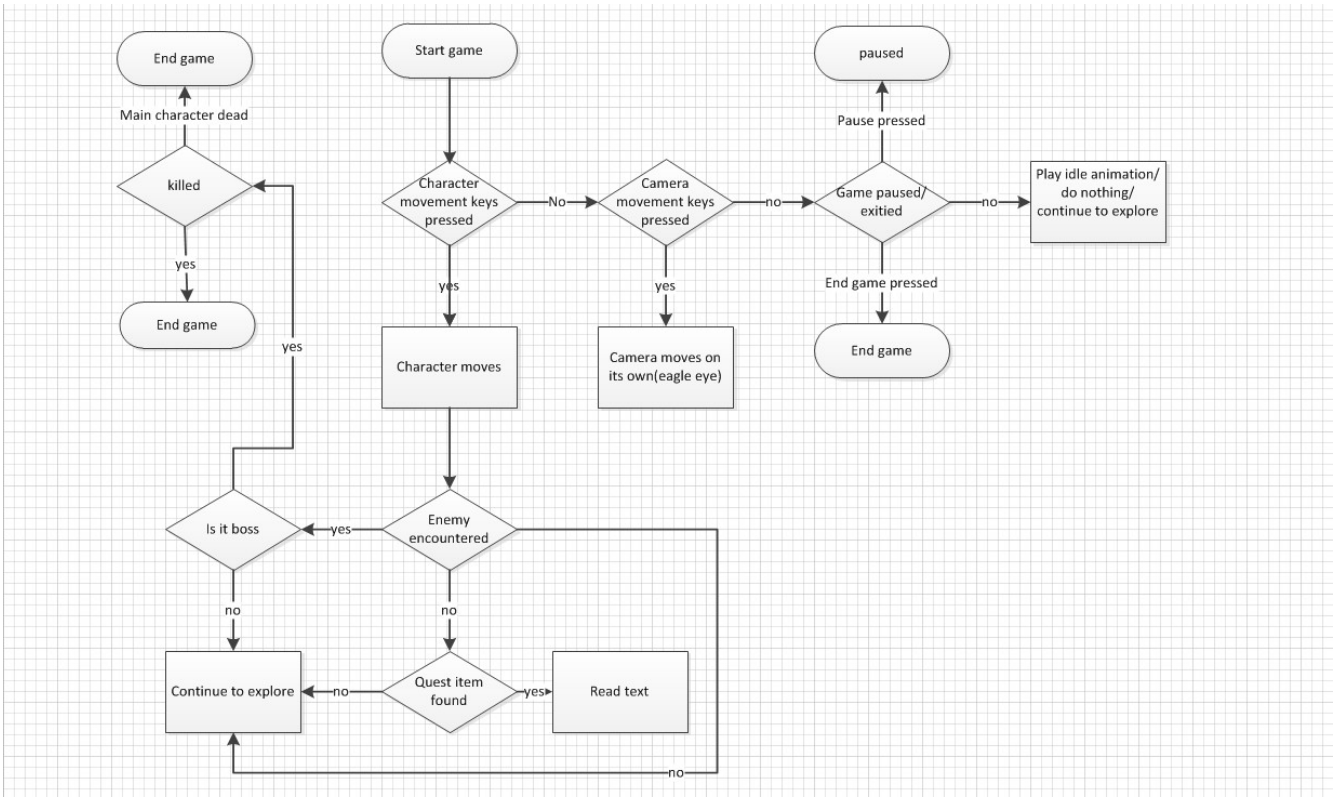
With the concept ideas being sketched the next step was to storyboard the main events that would happen through the game, this would help give an overall vision of the game when implementing the features and gain an idea on how the narrative will flow.



Technical brief

With the design brief made, the project can now move onto the technical brief, in this section the implementation methods will be designed will aid the project when programming of the game and modeling of the assets starts.

5.3 Data flow diagram of game playthrough



5.4 Game controls and navigation

Game controls and navigations are pretty standard across most games with usually the arrow keys moving a character and the mouse to move the camera around. However this project takes a slightly different approach, with the custom designing of the model the aim was to show it off and give the player a suitable camera perspective, for example if the camera was static 3rd person then it would only show the back of the model. This lead to the design of a new feature called 'Eagle eye' mode, the idea of this design is that the player can manually detach the camera from the model using alternative movement controls, this will give them a lot of freedom when moving the camera but it will keep the model still and when the movement keys are pressed for the model the camera will simply snap back into the normal 3rd person. This will also allow for additional features to be opened up, for example having treasure in a high and obscure place that would normally be missed by the player if they kept

with the third person camera, but when they use eagle eye mode they would see it and it would make them think about how to reach the treasure, this then adds another dimension to the gameplay.

5.5 Pseudo code for character movement and eagle eye mode.

To gain an insight into how the Eagle eye mode would function with the Ogre enigne, some psudo code has been written, it takes into account Ogres use of nodes to manage scene objects like models and cameras, this high level code will come in very usefull when it is implemented.

```
New mNode = main node
New cNode = createchild node mNode ->character node;
New camNode = createchilde node mNode-> camera node;
camNode->setposition(mNode->getposition()+offset z);

if up_arrow key is pressed {  mNode->translate.x +=1; }
if down_arrow key is pressed {  mNode->translate.x -=1; }
if right_arrow key is pressed {  mNode->translate.z +=1; }
if left_arrow key is pressed {  mNode->translate.z -=1;
//Move camera else attach it back to main parent node
if W key is pressed {  cNode->translate.x +=1; }  else {camNode->setposition(mNode…}
if S key is pressed {  cNode->translate.x -=1; } else {camNode->setposition(mNode…}
if D key is pressed {  cNode->translate.z +=1; } else {camNode->setposition(mNode…}
if A key is pressed {  mNode->translate.z -=1; } else {camNode->setposition(mNode…}
```

Here we are attaching both the model and camera nodes to a main node, we then translate this main node and the child nodes will inherit the movement, we when say if WASD are pressed move the camera node on its own else set the position of camera back behind the main node.

5.6 Engines selected

As analyzed in the literature review section there are multiple engines that I can use to design a PC game each with many benefits but also limitations.

The graphics engine that will render the game will be Ogre 3D, this was chosen because it is considered more advanced than Irrlicht as you get to hard code main features as opposed to have it already made for you, such as the camera, in Ogre you would do it traditionally such as setting the position, viewport and clipping distances, therefore using Ogre will enhance programming skills more.

The sound engine that I will use is Irrklang due to the amount of file formats it can support and its ease of use compared to OpenAL.

Chapter 6: Creating the Model

6.1 Modelling in Maya

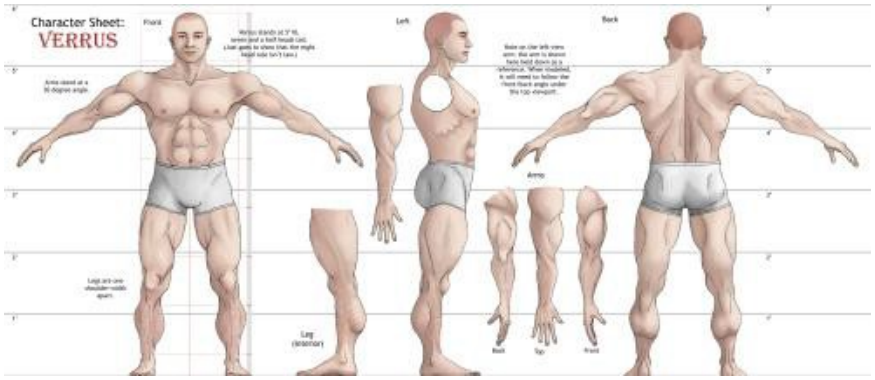
As discussed in my design, I wanted to try and make a fully fledged custom character model for my game, to experience the modelling/animation aspect of game design.

Firstly I need to create a mesh, the two main tools I researched where Blender and Maya, both where available for free and are widely used in the industry to make a mesh.

I decided to go with Maya for two main reasons, one I had brief experience with it before doing this project and such I was already accustomed with the controls and the capabilities of Maya, secondly it provides plenty of support to export my model and its animations/textures into my game engines, such as having a plug-in called 'OgreMax' that allows me to easily export my mesh and have it coded into Ogre to render it in game.

6.2 Choosing a reference image

To start of modelling I would need a reference image that I can use to effectively 'trace' around in 3D space, as doing it without a reference image would be incredibly difficult, particularly getting the proportions right.



Terms of use accepted: ©2011-2013 ~ElizaWyatt

Above is the reference image (ElizaWyatt, 2011) I Imported into Maya, It provides a clear views of the front and side, which were the two main views that I needed in order to start modelling, the proportions are also accurately represented.

The body would be modelled in smaller parts, starting with the torso then going into the legs and arms, the head would then be done last, this is a good way of modelling as each part can be exported separately if I wanted to use it later in a different model.

Also only 1 side of the model would be modelled as the other half would then simply be duplicated; this cuts the modelling time in half.

6.3 Modelling torso, legs and arms

To start with the torso I started off with a simply polygonal rectangle rotated 90 degrees, in the front view I could then simply extrude (extending one side of the rectangle) the sides to form a torso shape, I would then go into side view and align the torso side on.

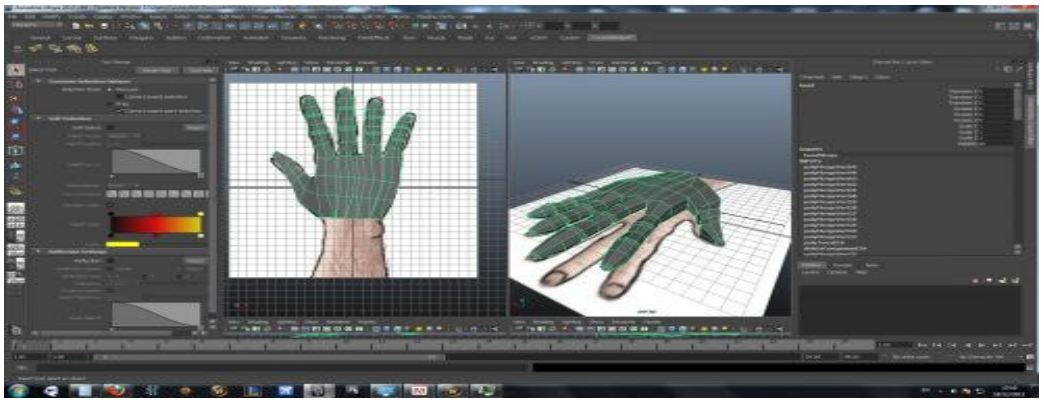
Using the 'duplicate special' tool in Maya I could easily model 1 side whilst getting the other side updated in real time as opposed to duplicating it at the end.

The legs and arms were then done in a very similar manner, once they were modelled I would snap the edge vertices of the legs and arms to the torso, I would then merge the vertices so it became 1 whole object.

6.4 Modelling the hands

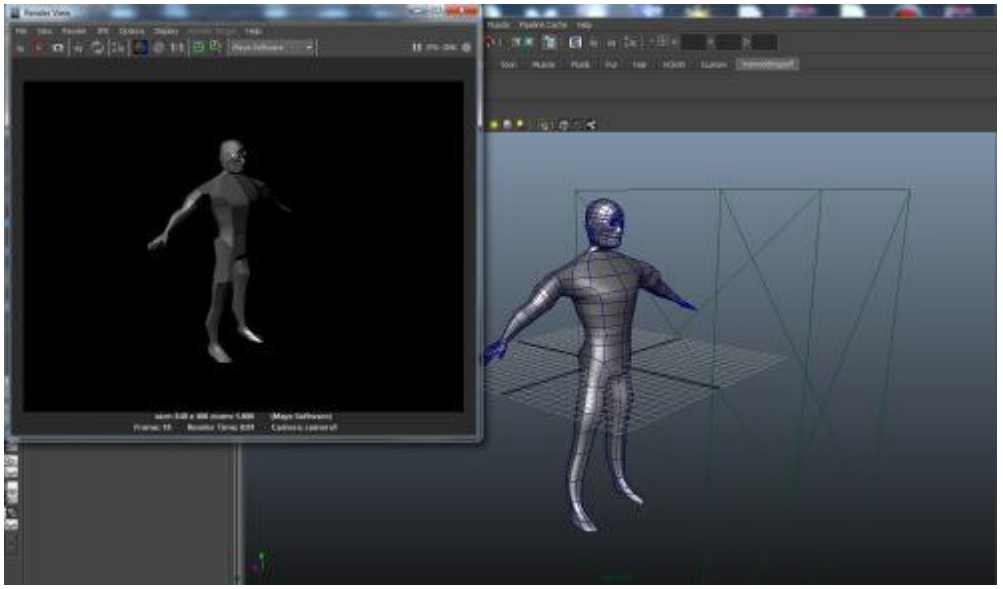
With the reference images in my main file it was difficult to model the hands correctly as I only had a side and front view, therefore I decided to model the hands in a completely separate file so I could use a difference reference image, the top down view of a hand. Same as the torso I started off with a polygonal rectangle and after many extrudes and moving of vertices I had the palm modelled. For the fingers I used cylinders that were subdivided (extra polygon faces added) then attached it to the palm, finely all that was left was to export this

hand and import it back into my main file, then attach it in the same way as the legs and arms, the only difference being that I also had to scale the hand to appropriate size.



6.5 Modelling the head and final result

Lastly the head was left to model; I could start off with two shapes, a sphere which I would then cut faces from or I could start with a plane and then extrude the sides to build the head. I decided to go with the plane as it would give me a cleaner polygon structure, particularly around the eyes, this would then help me texture the model.



Above left was the final result after modelling and attaching head to the body, this is currently very low resolution and thus has a very low polygon count, as a result it is very ‘blocky’ which is undesirable as it does not look very realistic. I then decided to apply a ‘smooth’ to my mesh (bottom right above), this creates additional polygons and averages out the vertices to create a smooth effect.

However this still did not give me the desired effect, as I wasn't satisfied around the leg area or the neck, and the torso did not have as much definition as I liked.

There were two alternatives to solve this little problem, I could either go back into my mesh and fine tune it by adding further individual polygons and vertices to give more detail, or I could look at another modelling programme.

6.6 Sculpting in Mudbox

I decided to go with the second option as I came across a perfect programme for making a character mesh, this was Mudbox. Similar to Maya it is also a modelling tool but more specifically it is a digital sculpting tool. An easy way to picture this is to imagine digital clay, using the mouse you can 'sculpt' in details to this 3D digital clay, the actually object still consists of polygons but the more detail you want the more polygons you will need.

A big benefit with Mudbox was that it already provide me with a basic character figure to start with, similar to the one I already modelled in Maya with very basic features outlined, i.e. a small bump to indicate a nose. With this I can 'sculpt' on features using tools such as 'wax' that adds extra layers on top of my mesh, 'knife' to make sharp indentations onto the mesh, 'bulge' and 'flatten' to raise and lower areas of the mesh.

(Toolset for Mudbox)

As I already had a preset character mesh, I would have to instead sculpt on features such as muscles, eyes, nose etc… for this I researched on the human muscle structure:

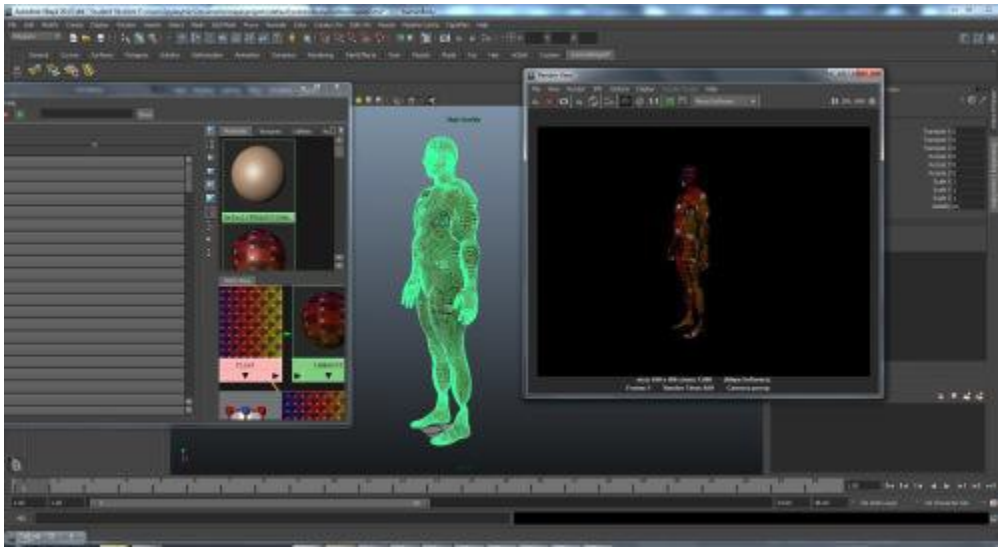
The picture to the left is the one I used as a reference point, I noticed that on the upper part of the chest there was two main chest muscles that would bulge out, on the lower part of the chest there was a section of 6 muscles. There were also muscles around the neck area which I did not know before. Using this picture I sculpted the model below.

In addition to the muscles I sculpted on his hair features and belt, I also flattened out the leg area to give the appearance of him wearing trousers.

One main downside to using Mudbox is the number of polygons generated as part of my mesh is quite high, around the 33,000 mark, however this is standard for a main game character and I am very satisfied with the result, as I have enough detail to make it realistic. I also have a very clean mesh, by this I mean that all the polygons are four sided and so it will be easier to texture and I do not have to worry about the mesh deforming unnaturally.

6.7 Exporting from Mudbox to Maya

In order for me to texture and animate my Model I had to export the mesh into Maya, as Mudbox is only a sculpting tool, there was no problems to this as both software were compatible with each other. Below you can see my mesh back in Maya with a test texture applied to it. (note the difference in polygons between previous model, but also the detail)

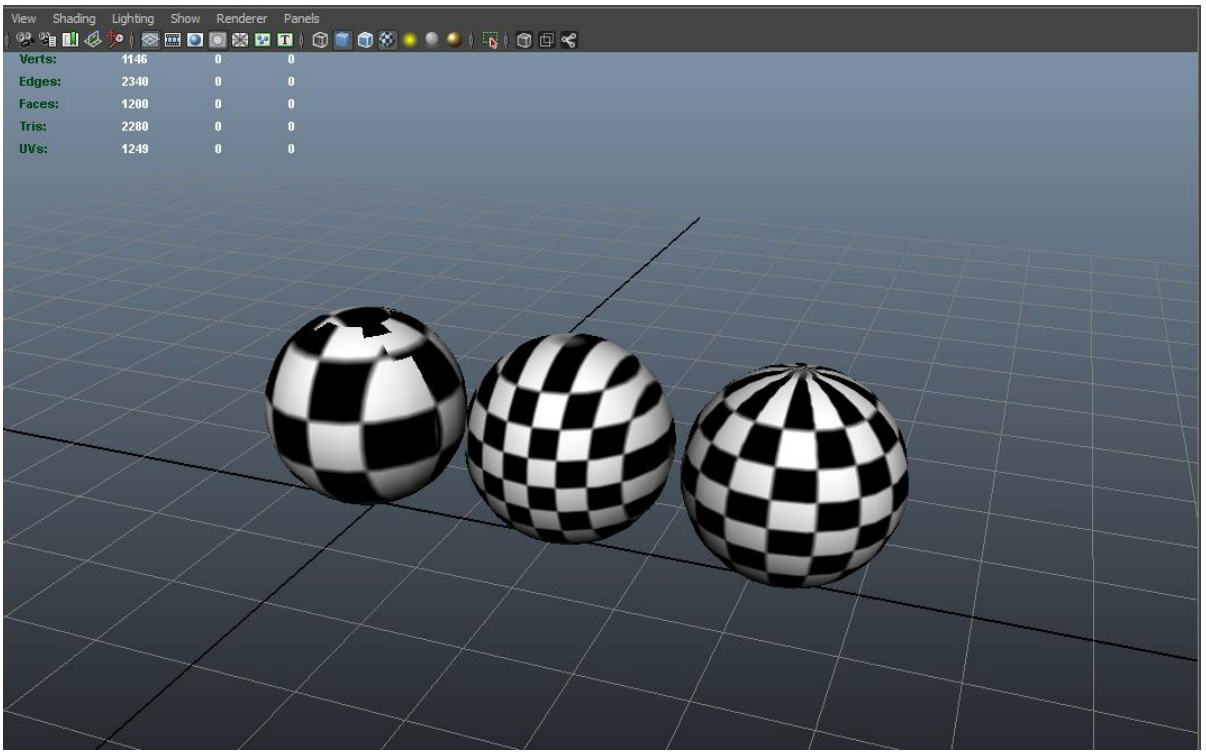


6.8 Texturing the model

The main method of texturing a mesh in Maya is to UV texture, this involves making a 2D representation of my 3D mesh, I can then use photo editing software to apply textures on the 2D representation which then automatically translates onto the 3D mesh.

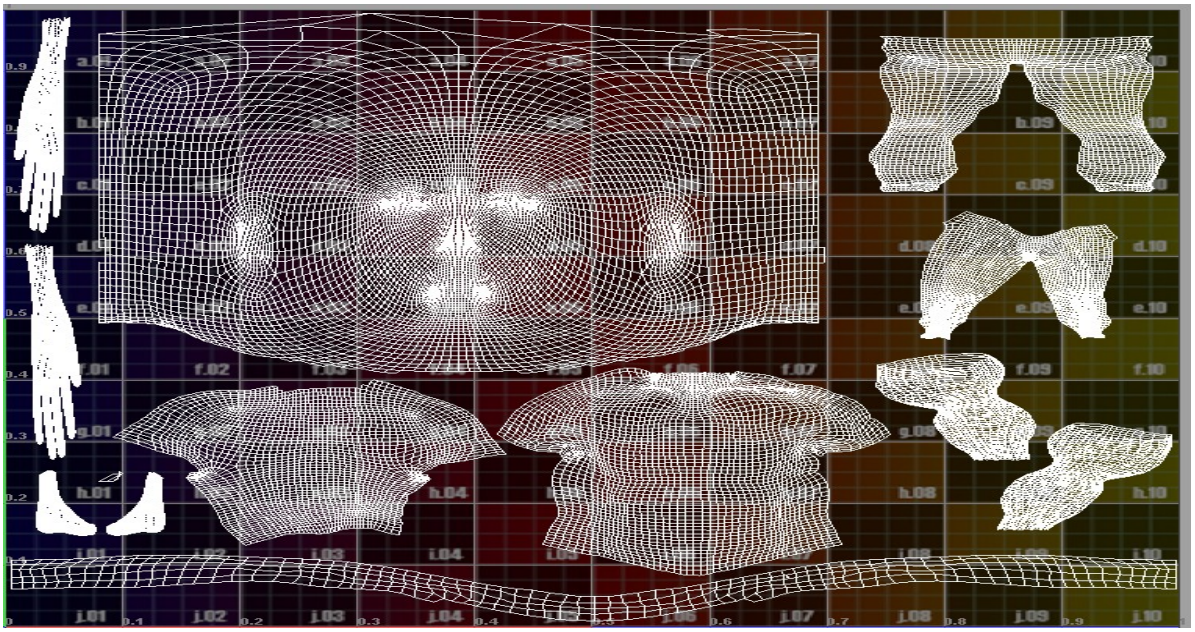
The UV are created via a projection, in Maya there are 3 main projection types that I needed to be aware of, these are Automatic (projection from 2-8 sides), Planer (projection from X,Y or Z direction) and cylindrical (projection in a cylyindal shape around the sleected mesh).

Below you can see the effects of how the texture is projected on depending on the type selected.



Projections from left to right, Automatic, planer(z) and cylindrical.

In order for me to texture my mesh correctly I will need to use a combination of all these types at select parts of my mesh.



Above is the final layout of my UV's, I used cylindrical mapping for the head and arms, and planer mapping for the legs and torso.

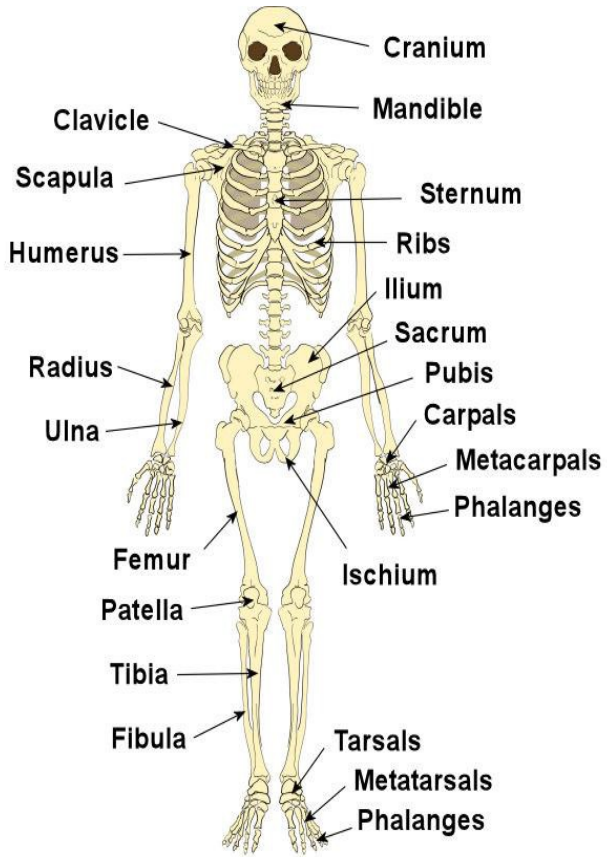
The aim was to keep each section of the mesh together in terms of their UV's i.e. the head, and whilst the default projections were more than good to get the job done, for it to be perfect I had to manually move and sew UVs together.

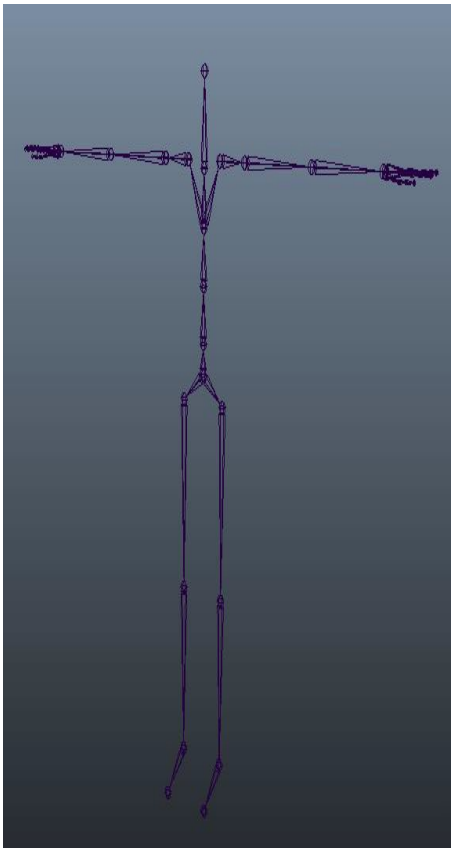
Such as on the belt and torso, the UV's were overlapping each other, this lead to some texture distortion. This test texture would remain until the game is close to finished as I would have to check for distortions when running animations

6.9 Rigging the model for animation

After modelling of the mesh and texturing it, the next step is to start to breathe life into it and animate, but first I would need to create a skeleton or 'Rig', that would allow me to naturally move my mesh and its joints when animating.

The Rig is essentially a skeleton within my mesh and is made up of multiple joints in a hierarchy, the structure of my rig follows closely that of a human skeleton except the actual joints are simple spheres and there is only one 'bone' connecting these joints.

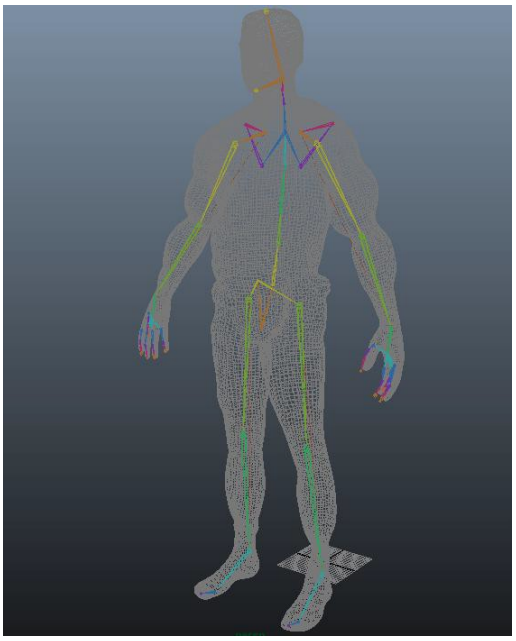


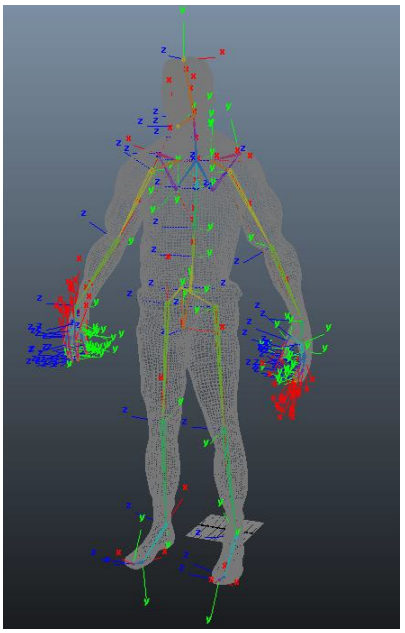


Above is the reference Image I used to create the basic outline of a rig in Maya on the right, all the necessary movement joints are present and those that do not promote movement such as the ribcage have been omitted.

One important factor I had to take into account when making my rig was to make sure each joint was orientated in the correct way, the X axis of the joint always aims in the direction of the next joint in the hierarchy, the Y and Z should be aligned in the same direction, doing this orientation correctly will allow the joints to rotate accordingly when I animate and will also make the IK handles I will discuss later easier to work with.

Below is the final version of my rig, I further developed the scapula and clavicle area of the rig, and also added some joints near the pelvis, this would give me a better structure when I bind the skeleton to the mesh and start to animate it. On the right picture you can also see how the joints have been orientated, the Z axis all points towards the left.

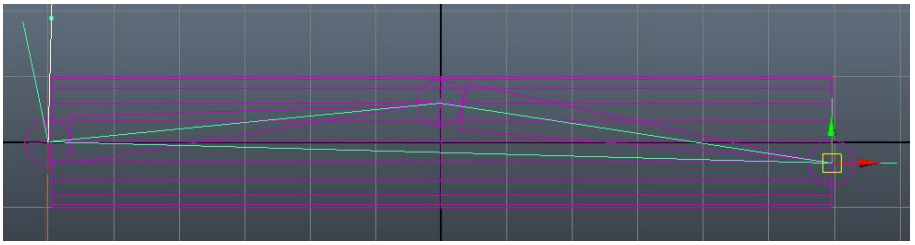




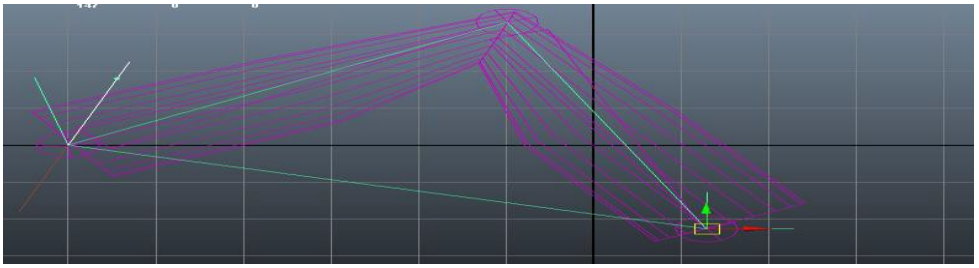
IK handles

IK (Inverse kinematics) handles in Maya is a great tool that helps making a natural animation, it calculates how a series of joints should move in between key frames of animation. I used IK handles in 4 key places on my rig, In the left and right arms, and in the left and right legs, this would then allow me to simply move the IK handles instead of each individual joint when animating.

Below is an illustration of this with 3 joints, similar to an arm or leg to demonstrate how IK handles work. (The handle is located at the end joints and Is an independent node to the other 3 main joints)



I can then simply move the IK handle and it will automatically deform the 3 joints realistically, therefore I only need to set a key frame for one handle as opposed to 3 joints.



6. 0.1 Animating my model

With the rig completed animation on the model could begin, this is also done in Maya by switching to the animation toolset it provides.

The settings of the animation will be set to PAL (25FPS) this means each second of animation consists of 25 frames, PAL is the main video format we use in the UK.

Below is a screenshot of the model in a run pose in the middle of its animation, video is available on the project blog. (URL found in appendix see fig.8)

To make it animate you would pose up the model and create a key frame at a specific point in the timeline, generally the first and the last frames are the same, and this allows the animation to loop naturally. With the use of IK handles I could naturally pose my mesh, instead of manually moving each joint in the rig.

Chapter 7: Implementation of code

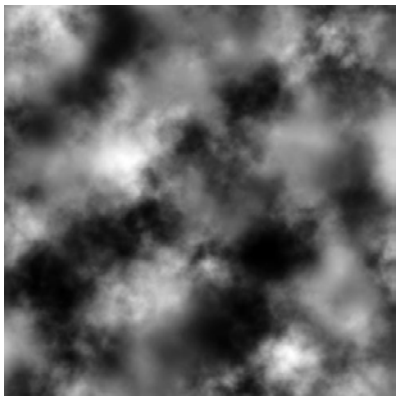
How the game will be implemented

This chapter explains how the code has been implemented to create the game world and load in assets to the engine such as the model there will be code snippets that will show evidence of this. This chapter will also show how some of the concept ideas outlined in the design chapter have been implemented, this includes features like terrain and sky.

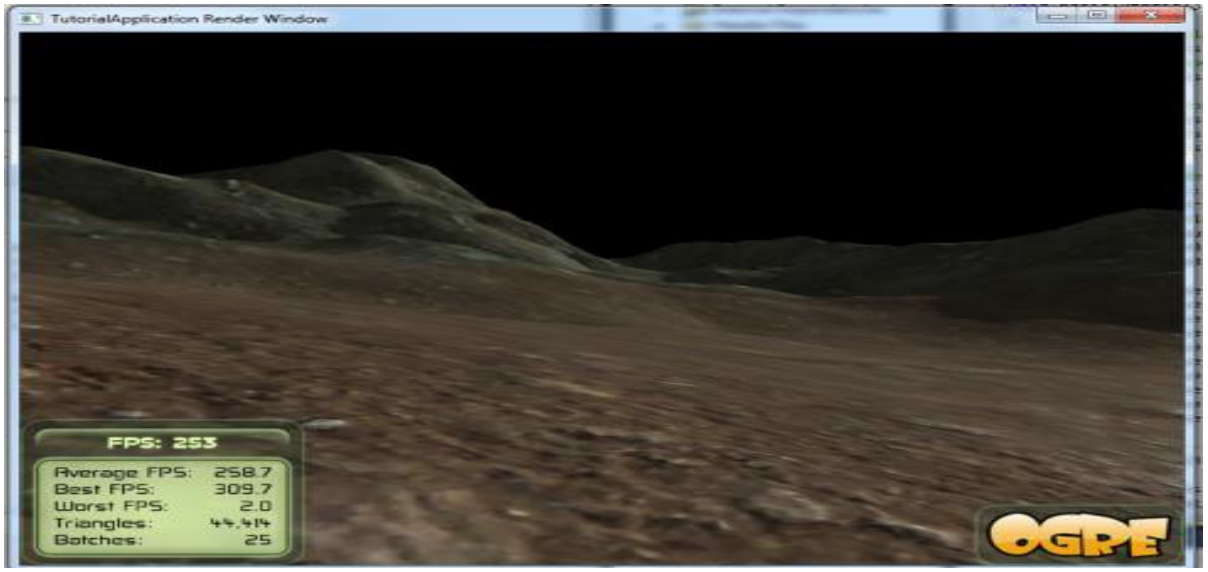
This project will use primarily the OGRE 3D graphics engine and the Irrklang sound engine, 3rd party assets will also be used mainly for the misc. scene objects and textures.

7.1 Creating the Terrain

For Ogre to create a terrain it needs to load in a height map, then it can load in various texture, bump and specular maps to give it more realism. The greylevel values of each pixel translates into how high a certain vertex is on my terrain, if it is black, then the terrain will be lower than if the pixel value is more white.



Here a pre-made height map already provided by Ogre, the height map size itself also effects how large the terrain will be, in this case it is 512x512 (there is a variable that can change the terrain size aswell).



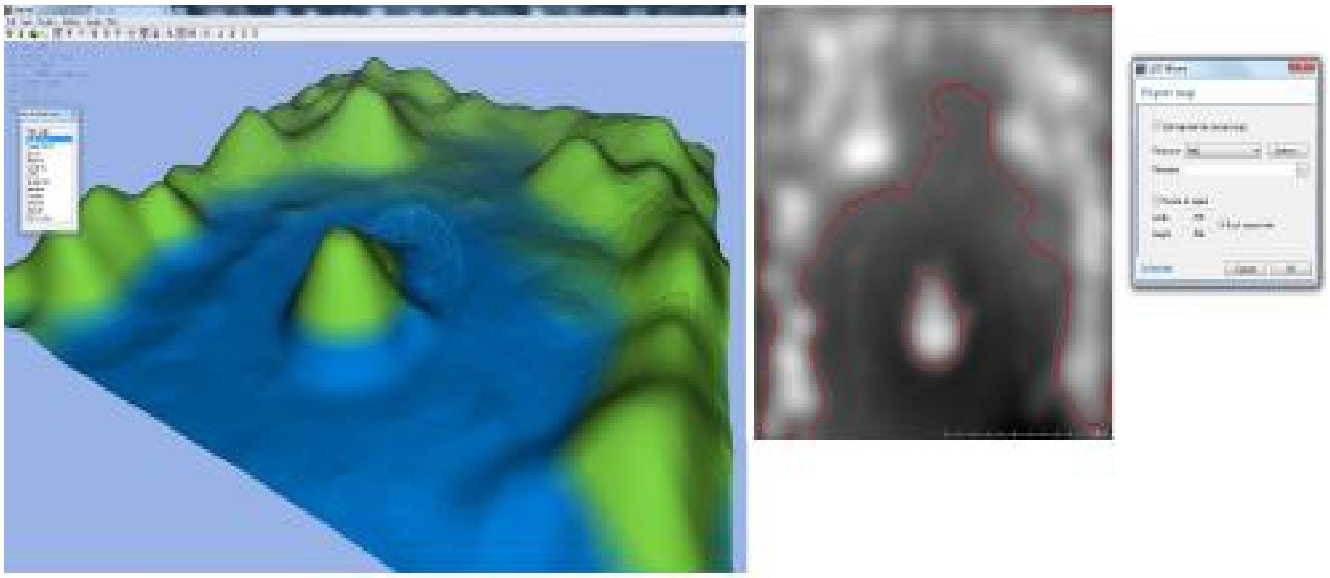
Above is how the terrain looks when the game is run (including texture and bump maps), the brown parts correspond to the darker areas, while the green parts are the lighter areas in the height map.

The current terrain did not match my original concept ideas as I wanted a flat middle with raised up ring around the edge to give it a valley desert feel. This does mean altering the height and texture maps in place; this can be done in photo editing software.

However this did present me with a problem, for me to create a smooth height map I would need to make sure each pixel was correct in its greylevel, this was very tedious to do in photo editing software, and would have needed constant loading and building of the terrain in Ogre to keep checking how it would turn out.

How I solved this was via a brand new software that I discovered, a height map generator called L3DT (Torpy, n.d.), this meant I no longer had to guess heights in Photoshop, with this tool I can gain a 3D representation of my terrain in real time, so I do not have to keep loading it into Ogre as well. Once I was finished I could export the height map in my desired size and format and load it in the getTerrainImage function shown below.

```
void getTerrainImage(bool flipX, bool flipY, Ogre::Image& img)
{
    img.load("terrain.png", Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME);
    if (flipX)
        img.flipAroundY();
    if (flipY)
        img.flipAroundX();

}
```



Above is me working with L3DT tool, it provides a sphere manipulator that is used to alter the terrain, it can raise and lower sections easily allowing me to get a terrain closer to my design concepts.

7.2 Adding a sky to the game

Now that I had my terrain I needed to add a sky, there were 3 ways of implementing this with Ogre, I could use either a SkyBox, SkyDome or Sky Plane. (Ogre 3D tutorials, 2010)

- The skybox is giant Cube that is created around your terrain, a texture is then projected onto each face to create the sky effect, this is in the middle in terms of the processing speed it will take to render after the terrain.

  This type of skybox is mainly used if you wish to look in all directions when playing a game with no tangible terrain present, an example of this would be a game set in space where you would 'float' around in all angles.

- The second type of sky available is the SkyDome, this is similar to the skybox in that it is a giant cube, with the main difference being the texture is projected in a spherical manner as opposed to on each face of the cube. In addition the bottom half of the cube is left un-textured, so some form of terrain is needed to hide this.

  This is also the most intensive in terms of rendering of the sky due to the calculations needed to project the texture, it also causes some waste in the form of the lower part

of the giant cube which is left blank, the benefit of this however is the texture of the sky is seamless and very realistic.

- The SkyPlane is very different from the previous two types of sky, here a single plane is created and faced downwards, the plane is then split up into multiple tiles, the frequency I can define. The texture is then repeatedly projected across each of the tiles, this gives the illusion of a moving sky.

I decided to use the SkyPlane for my game as it is the least intensive in setup, as I already have a fairly large polygon count from my terrain and model, this type of sky is also perfect for the type of terrain that I have as a major drawback of a SkyPlane is that when you reach the edge of a terrain you can clearly see the empty void and the gap between the plane and terrain, this is resolved by having a valley terrain as the player would not reach the edge, in essence the valley serves as the other size to still create a cube.

Another way to lessen this drawback of the SkyPlane is to give the plane a slight curve but it would also increase the processing needed. The SkyPlane also works very nicely with Fog effects that I will implement next. Below is the code snippet that shows how to add a SkyPlance, if I wanted to make a box or dome I would change “setSkyPlane” to “setSkyBox or “setSkyDome” and fill in the parameters.

```
Ogre::Plane plane; //create the skyplane
plane.d = 300;
plane.normal = Ogre::Vector3::NEGATIVE_UNIT_Y; //face pnae downwards

//add texture and have it tile across, the curviture is also defined
mSceneMgr->setSkyPlane(true, plane, "Examples/CloudySkyDark", 500, 20, true, 0.5, 150, 150);
```

Above is the result of me adding a SkyPlane to my terrain, the valley edge that I have, obscures the gap I have between the plane and the terrain.

7.3 Implementing Fog

Fog can be implemented in my game to give a great atmosphere effect such as the desert sandstorm that I wanted during my concepts. In Ogre fog is a filter that is applied to objects in the viewport instead of it being a separate entity, this means that if we turn the camera to face nothing, there will be no fog present; this is great to save on some processing power.

There are also two main types of Fog that Ogre supports, this is linear fog and exponential fog.

- Linear Fog gets thicker in a linear fashion, increases at a set rate, this means that regardless of distance travelled in the game the fog will remain constant to the camera at all times.

  Therefore this type of fog has a predefined start and end point from the camera, i.e. 50 is start and 500 units is the end point, if you are 500 units away from the camera there will be no fog present, but since the player is playing from the camera perspective this will not be an issue.

- Exponential fog on the other hand does not have a start or end point, instead the thickness is defined, this will give a more realistic fog at the expense more processing power.

Below is the code snippet that implements the fog, the colour is set to a dark brown giving the impression of a sandstorm, the thickness that is defined is ideal to obscure far places in the terrain but clear enough to see what is in front.

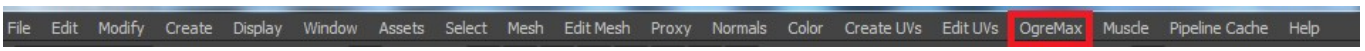
```
    //fog

    //set colour RGB
    Ogre::ColourValue fadeColour(0.2, 0.1, 0.1);
    //type of fog and the thickness
    mSceneMgr->setFog(Ogre::FOG_EXP, fadeColour, 0.0006);
    mWindow->getViewport(0)->setBackgroundColour(fadeColour);
```

7.4 Loading the model into the game

Once the model has been animated and textured in Maya, the steps to import it into the game can begin, the first step would be to export the Maya scene into a format Ogre can recognise and load up in its pipeline.

There are three types of files Ogre needs to load up a model, these are .mesh, .skeleton and .material files, in .mesh all of the polygonal information is stored about the mesh, in .skeleton information about the rig and the animations are stored, finally in the .material file the UV and textures are stored.

To create the mesh and skeleton files a plugin needs to be installed for Maya, it is called OgreMax and it is a specialist Ogre scene exporter, once these files are made they will simply need to be stored in the media folder in the game directory.



For the textures a script has to be written with a .material extension, this script works similar to effect files that are used by other engines. In the script you can define features such as the ambiance or secularity of the texture and select a specific texture file to load as this is not exported with OgreMax.

```
material lambert2
{
        technique
        {
                pass
                {
                        ambient 1 1 1 1

                        texture_unit
                        {
                                texture uvtemplate001-lg.jpg
                        }
                }

        }
}
```

Above is my .material script for my model.

In the code only the .mesh file will need to be loaded, for my game I am creating the main model node to be a child of the 'main node' this is because of the camera system that I will implement later on in this chapter. Below is the code snippet that loads the model into the game once it has been exported correctly.

```
//create a main node
MainNode = mSceneMgr->getRootSceneNode()->createChildSceneNode("MainNode",(Ogre::Vector3(2100, 50, -1250)));
//create a entity for my model
mainCharEntity = mSceneMgr->createEntity("MainChar", "maincharv3.mesh");
//create a child model node of Main node
Ogre::SceneNode *PlayerNode = MainNode->createChildSceneNode("MainCharNode");
//attach the model
PlayerNode->attachObject(mainCharEntity);
```

A screenshot of my model with its test texture is shown in the appendix, see fig.8

7.5 Implementing Camera and navigation system

Creating the camera in Ogre requires two functions, the first is the createCamera() function that creates a camera object and attaches it to the scene manager, the position is then set in the scene and the lookat area defined, an extra parameter that can be added is clipping distance this simply tells the camera how far or near to render the scene objects.

The second function is to create the view port, in this function it simply sets the window sizes and alters the cameras aspect ratio to match the viewport. Below is the create camera function.

```
void BaseApplication::createCamera(void)
{
    // Create the camera
    mCamera = mSceneMgr->createCamera("PlayerCam");

    // Position it at 500 in Z direction
    mCamera->setPosition(Ogre::Vector3(0,0,80));
    // Look back along -Z
    mCamera->lookAt(Ogre::Vector3(0,0,-300));
    mCamera->setNearClipDistance(5);

    mCameraMan = new OgreBites::SdkCameraMan(mCamera);   // create a default camera controller
}

void BaseApplication::createViewports(void)
{
    // Create one viewport, entire window
    Ogre::Viewport* vp = mWindow->addViewport(mCamera);
    vp->setBackgroundColour(Ogre::ColourValue(0,0,0));

    // Alter the camera aspect ratio to match the viewport
    mCamera->setAspectRatio(
        Ogre::Real(vp->getActualWidth()) / Ogre::Real(vp->getActualHeight()));
}
```

The viewport function above.

To navigate you can either use the default camera movement system provided by Ogre called SdkCameraMan, or create your own inputs that will handle the camera translations, for this project it would use a mixture of both, this is due to the eagle eye mode that will be implemented.

His essentially will use different key inputs to detach the camera and have it translate according to the player's wishes; however it will still maintain its normal functions such as rotation and orientation.

To do this feature I will need to attach the camera object to a node, then have the camera node attached to a translation node, in this case it is called 'main node', the model is also attached to the main node therefore both will inherit the movement of the main node.

Below is a code snippet of creating a camera node, when the users press's the input keys the character and camera will move with the camera always being offset, when WASD are pressed it will transform the character node only, but when the arrow keys are pressed, the camera node will offset back towards the main node and continue to inherit its movements.

```
//create camera node that is a child of main node, positioned behind the main node
CameraNode = mSceneMgr->getRootSceneNode()->createChildSceneNode("CameraNode", (Ogre::Vector3(2250, 175, -1300)));
CameraNode->attachObject(mCamera);
//offset the lookat to make it into a 3rd person view
mCamera->lookAt(MainNode->getPosition() +Ogre::Vector3(0,50,0));
```

(See fig 6. in appendix for code on the various inputs that the game takes)

7.6 Enemy models and its movement

In addition to a main boss who would be added in a similar manner as the main character node but without any character controls, the game has small scarabs that spawn and patrol in an erratic fashion.

The scarabs are also created and attached to node just like the main character and camera, a walk list is then created that defines points in the world, these points when looped through will give the impression of the scarab patrolling around.

Below are the snippets on how the scarabs are created and the walk list positions being set.

```
    //create scarab
mEntity = mSceneMgr->createEntity("scarab", "scarab.mesh");
mEntity->setCastShadows(true);

//starting position
snode = mSceneMgr->getRootSceneNode()->createChildSceneNode("ScarabNode",Ogre::Vector3( 1750, 20, 2000 ));
snode->attachObject(mEntity);
//scaling the scarab
snode->scale(2,2,2);
//oreientating the scarab
snode->yaw( Ogre::Degree( 90 ) );
snode->roll( Ogre::Degree( 180 ) );

// Create the walking list, its patrol points
    mWalkList.push_back(Ogre::Vector3( 1750, 20, 2000 ));
    mWalkList.push_back(Ogre::Vector3( 1800, 20, 2050));
    mWalkList.push_back(Ogre::Vector3( 1900, 20, 2000 ));
    mWalkList.push_back(Ogre::Vector3( 1750, 20, 2000));
    mWalkList.push_back(Ogre::Vector3( 1880, 20, 2050 ));
    mWalkList.push_back(Ogre::Vector3( 1900, 20, 2100));
    mWalkList.push_back(Ogre::Vector3( 1900, 20, 2000 ));
    mWalkList.push_back(Ogre::Vector3( 1750, 20, 2000));
    mWalkList.push_back(Ogre::Vector3( 1750, 20, 2000));
```

The function nextlocation() shown on the next page, loops through the walk list and normalises the direction the scarab is heading.

```
bool DiM_Game::nextLocation(void)
{
    if (mWalkList.empty()) return false;
    mDestination = mWalkList.front();  // this gets the front of the deque
    mWalkList.pop_front();             // this removes the front of the deque
    mDirection = mDestination - snode->getPosition();
    mDistance = mDirection.normalise();
    mWalkList.push_back(mDestination); //loop back through walklist
    return true;
}
```

The next few snippets will show how the scarab will walk from point to point and attempt rotate when it has to turn around so it keeps facing the direction it is heading.

First is to enable time allow for animation, in Ogre you can set the speed manually as opposed to making shorter key frames in Maya.

```
    //enabling time to enable animations +5 makes the animation faster
    mAnimationState->addTime(evt.timeSinceLastFrame+5);
```

Next is to enable the scarab walk cycle to loop only when it is transitioning from point to point.

```
    if (mDirection == Ogre::Vector3::ZERO)
    {
        if (nextLocation())
        {
            //Set walking animation
            mAnimationState = mEntity->getAnimationState("scarabwalk");
            mAnimationState->setLoop(true);
            mAnimationState->setEnabled(true);
        }
    }
```

A move variable is then created that will be translating the scarab.

```
    else{
        Ogre::Real move = mWalkSpeed * evt.timeSinceLastFrame;
        mDistance -= move;
        if (mDistance <= 0.0f){
            mDistance +=move;
            mDirection = Ogre::Vector3::ZERO;
```

Rotation of the scarab node to make it keep facing the direction it is moving. After this we

```
        }else{
            //Rotation Code
            Ogre::Vector3 src = snode->getOrientation() * Ogre::Vector3::UNIT_X;
            if ((1.0f + src.dotProduct(mDirection)) < 0.0001f) {
                //snode->yaw(Ogre::Degree(180));
            }else{
                Ogre::Quaternion quat = src.getRotationTo(mDirection);
                //snode->rotate(quat);
            }//else
```

7.7 Creating a UI

Whilst there are UI engines out there as discussed in the engine review at the end of literature review chapter, for the project I opted against using one of these as there was no need to make custom widgets to add into my game. Ogre allows you to display messages via labels and this was sufficient to display any feedback in the game to the player.

Below is the snippet that shows the labels being used to display a couple of messages including when eagle eye mode activates.

```
//main welcome label
mTrayMgr->moveWidgetToTray(mInfoLabel, OgreBites::TL_TOP, 0);
mInfoLabel->show();
mInfoLabel->setCaption("Welcome to DiM");

//eagle eye message being displayed
if (mKeyboard->isKeyDown(OIS::KC_W)) // Forward
{
//displayed message at top of screen just below the welcome message
mTrayMgr->moveWidgetToTray(mInfoLabel2, OgreBites::TL_TOP, 0);
mInfoLabel2->setCaption("Eagle eye mode");
mInfoLabel2->show();
----------------
}
//hide label when eagle eye mode deactivates when the key is released
else
{
mInfoLabel2->hide();
}
```

Below is the result of the code being run in the game.

7.8 Raytrace terrain collision

With terrain and character movement in place, the next step is to prevent the character from falling through the world, to test this I have added additional movement inputs to the model that moves it directly up and down on the O and U keys, when the character does not go down when O is pressed would mean the terrain collision is a success. I would also need to do this with the camera as it should not go through the floor when eagle eye mode is enabled.

A way to do this in Ogre is to use ray collision, here a ray is beamed out from the object specified in the specific axis. Below is a code snippet showing this, the ray is beamed out from the Y axis of the camera.

```
//create a camera ray
Ogre::Ray cameraRay(Ogre::Vector3(camPos.x, 50000.0f, camPos.z), Ogre::Vector3::NEGATIVE_UNIT_Y);
//enable ray to intersect terrain
Ogre::TerrainGroup::RayResult rayResult = mTerrainGroup->rayIntersects(cameraRay);

//if hit collide
if (rayResult.hit)
{
    //ypos of terrain
    float fHeight = mTerrainGroup->getHeightAtWorldPosition(Ogre::Vector3(camPos.x, 0.0f, camPos.z));

    if ((fHeight + 2.0f) > camPos.y)
    {
        mCamera->setPosition(camPos.x, fHeight + 2.0f, camPos.z);
    }

}
```

7.7 Adding sound

Sound has also been implemented into the game with the help of the Irrklang sound library, once it is linked into the project and the plug-in added to the main directory, it is straightforward to add sound.

First you would need to call the engine and create a new Irrkland device, and then you simply tell it to play a 2D or 3D sound with the file path specified. Below is a snippet showing this implemented.

```
//make new irrklang device
irrklang::ISoundEngine* engine = irrklang::createIrrKlangDevice();
//play sound with desired file attached
engine->play2D("C:/Ogre/irrKlang-1.4.0b/irrKlang-1.4.0/media/getout.ogg",true);
```

Above an .ogg file is being played, the last parameters is set to true so the file will keep looping over and over, this is good for some background music to my games.

Chapter 8: Critical Review

8.1 **Summary of what has been achieved**

From the start one of the main aims of the project was to gain skills that I would not normally gain as a programmer and to refine skills that I already possessed, through this project I can confidently say that I have met these aims and have enjoyed it immensely.

I have gained useful skills in the design aspect of making a computer game, with my ambition to be a content developer I learned on my placement that I needed both design and programming skills. In the design aspect I have learned how to make creative concept ideas and art on paper such as designing my main character and the environment in which they move in, I have learnt how to storyboard effectively to help give me a overall vision on the project. Granted one further aspect I can improve on is my drawing ability to make the concept sketches a bit clearer and make the storyboard closer to what I had imagined.

I have also learnt how to implement these design features into the 3D world, first by using Maya to learn the basics of polygonal modelling, then moving onto the world of sculpting in Mudbox that produced a very realistic mesh, further to that I learnt that there is much more than simply modelling to create a main character model. I found that the UV's had to be carefully mapped to apply textures onto the mesh, something that I was not aware of before I started this project. Another new feature I learned that Rigging and animating are two very separate skills that you can gain, at first I thought they would go hand in hand simply make a basic skeleton then move some joints around, I quickly gathered that you needed basic knowledge on the anatomy you are modelling to create a skeleton that will rotate and deform in a correct manner, in the animating feature I am pleased to have achieved a basic run cycle and an attack animation, although I feel if my rigging was more advanced in having more controls to move and twist joints I could have implemented more complex animations.

A Major aspect I feel I have achieved was to combine the design and implementation together, a very proud moment being when my model was properly animating the run and attack cycles in the actual game, this also made me more conscious about the performance aspects of the game where inefficient modelling can lead to a noticeable drop in frames per second when I plugged in my high polygon mesh into the game, the FPS went from 300 to

75, so I can improve on this by slightly toning down the amount of polygons I used such as using a lower subdivision level to create the mesh.

On the programming side I am also very proud of what I have achieved in that I implemented roughly 90% of the features that I wanted to from the must do section of the Moscow requirement analysis.

8.2 Lessons learned over the course of the project

In hindsight after critically reviewing the project there are a few aspects that I would have done differently, some to make the project more efficient in terms of time saving and performance and others to add additional features to the project.

The first instance where I was not pleased was the first iteration of my model that was done purely in Maya through traditional polygonal modelling, this lead to the change in tools to Mudbox where it was very efficient at not only building a full human body but it felt more natural to sculpt in the fine detail such as around the face and hair. However with a better reference image to work from and more experience I could have created a model almost as good but with considerably less polygons than purely sculpting in Mudbox.

Textureing my character model is another area that I could improve upon greatly, at present there is only a colour map attached to the mesh, with the use of other texture maps like bump, specular and normal maps I could have ‘faked’ more detail into the mesh without having to sculpt in additional polygons, this would then make my game run at a higher FPS.

In the rigging aspect I could have added more controls to the mesh, there are currently slight irregularity’s in how the mesh rotates it arms but this is due to a lack of control around the elbow area to make it twist more realistically, some of the joints are also oriented in the incorrect manner such as the X and Y axis on the hand area this leads to the hand rotating in a unrealistic way, one way to correct this would be to run a Maya script that would automatically orientate the joints, but that was outside the scope of the project.

I would also have preferred to expand my Eagle eye feature to make it integral to the gameplay, as currently if feels like an extra way to navigate around the scene but with the camera only, for example I would programme objects or enemies that would be in a location only reachable via Eagle eye mode, this gives an more direct incentive for the player to use this feature.

Currently my game has pretty basic physics attached, whilst the initial plan was to utilise a physics engine in the end the collisions where hard coded making use of Ogres bounding sphere functions, the reason for this was that I would have to have exported my mesh into a format that could make use of the dynamics that the physics engines offered but there was no justification to add this except to improve combat visuals. For example there is no instance where I would need to use the ragdoll effect on my model since my combat is not advanced.

8.3 Game rating

In the UK the PEGI rating system is used to assess the suitability of the game to a particular age group, this is determined by the themes and feature present in the game.

For this project the game would receive a PEGI rating of 12+ (Pan European Game Information, n.d.) , this is because there is some violence in the form of killing the little scarabs and then the end boss who bears a strong resemblance to a human being. There is also a sense of fear which is triggered by the sound effects and the scenery being a stormy desert valley with an eerie temple being concealed.

Fear
Game may be frightening or scary for young children

Violence
Game contains depictions of violence

8.4 Ethical/legal/social impacts

The common ethical question bought up is ‘is killing right in video games?’, there are many references in the media that suggest violent video games produces violent people (Richardson, 2012), where they aim to replicate what they play in the game into real life. My stance on this issue is against this argument as I feel this only affects the minority but they would be the same people who would ignore ‘do not try this at home’ messages they see on TV or film.

Many people play games as an escape from the stresses of real life for a brief amount of time where they can get lost in a fantasy world or experience feelings that would usually be against the law such as killing or stealing, there are people who argue that gaming takes away time from important aspects such as education or social interaction with peers, but in every booklet or manual the developers always state to play the game in moderation and to take regular breaks, those who ignore this do so at their own risk, I see this as similar to people

who smoke, the labelling of the side effects are very clear and at times graphical but people are aware of this and still decide to smoke, whilst some get very addictive many do it in moderation, the same applies to gaming.

Whilst it is true that violent video games are more prominent now as opposed to when the industry was young, this is due both to the popularity of game features such as shooting a gun in a war and the millions of dollars publishers spend to promote the game through every possible media outlet. Evan with the rating system minors would still attempt to play the game that is above their age, this is hard to prevent with the only realistic means at the point of sale, but the question arises why would they want to play this type of game? What makes it attractive? The answer would lie by understanding the common stereotype where it is perceived that young males are the ones that predominantly play games (Billieux, et al., 2013) in particular more prone to play violent games that involve killing. This likely stems from the ancient prehistoric times where men were considered the prime hunters who would provide for families and defend any threats; this is similar in games that involve killing or defending by combat hence a main reason to why this type of game is attractive.

8.5 Future plans

This project has been just a start of the grand design that I have made, right now I have implemented just one scene and scenario to the game, the next step is to refine what I currently have and look to expand the game into new scenes and environments that will further progress the narrative.

In my current game I will plan to refine the combat system by including a physics library, this would enhance the animations that I already have in place and allow for different animations to be implemented such as the block and jump that I had planned for in the Moscow requirement analysis.

I also have plans to take the modelling side further, particular sculpting additional characters in Mudbox, for example I would like to create a female version of the main character and have a character select screen at the start where you could choose between the models, I can then further expand this by having different skins for each model, this also is a great commercial aspect where custom skins are usually charged for.

Difficulty levels is another feature that I would like to implement in the future, such as making the terrain harder to navigate and having the enemies hit harder and with more advanced AI. This would then go hand in hand with a reward system such as new items to use in combat or additional areas unlocking.

8.6 Final conclusion

All in all this was a very challenging individual project to take on; however this is very real to how a present day game studios would conduct their projects, the main difference being that it is spread out over various specialised departments.

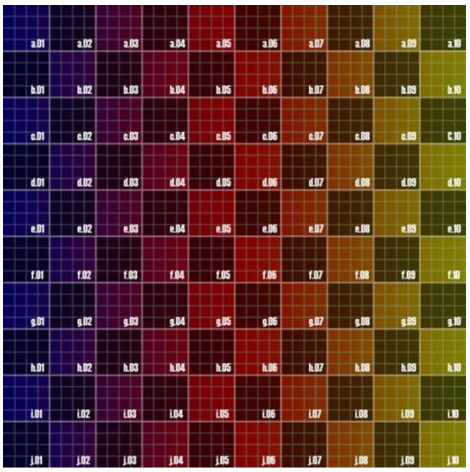
Due to the amount of new skills I have gained particularly in the design aspect of making a game I consider the project a success and have enjoyed it thoroughly, this will be a great project to add to my portfolio to enhance my future job prospects.

I also enjoyed keep a running log of my project via my blog, which was a fantastic suggestion by my supervisor, not only did it serve as my lab book where I could keep tabs on what tasks I have done but it also gave great motivation to keep adding features. I will surely use this again for future projects.

A link to my blog can be found in the appendix fig 7.

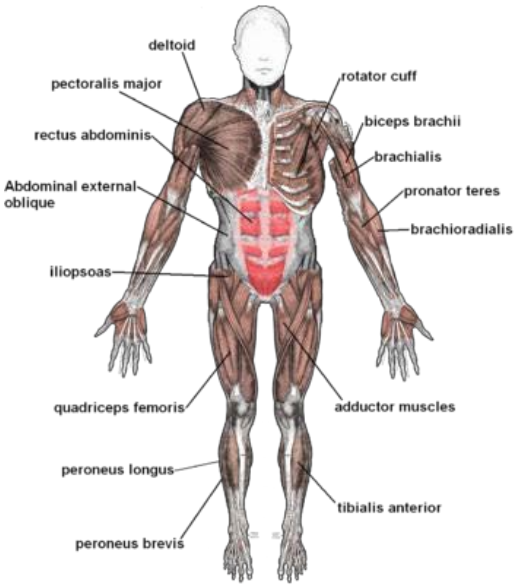
Appendix

Fig.1 UV texture template



Sourced from: Simpson, L, 2012, uvtemplate001-lg, Computer Generated Imagery and Sound TS3230-ALL_SEM1, Kingston University, Unpublished.

Fig 2. Muscle structure template



Wikipedia, n.d. *Muscle.* [Online] Available at: https://upload.wikimedia.org/wikipedia/commons/thumb/e/e5/Muscles_anterior_labeled.png/350px-Muscles_anterior_labeled.png

[Accessed 18 10 2012].

Fig 3. Concept sketch page 1

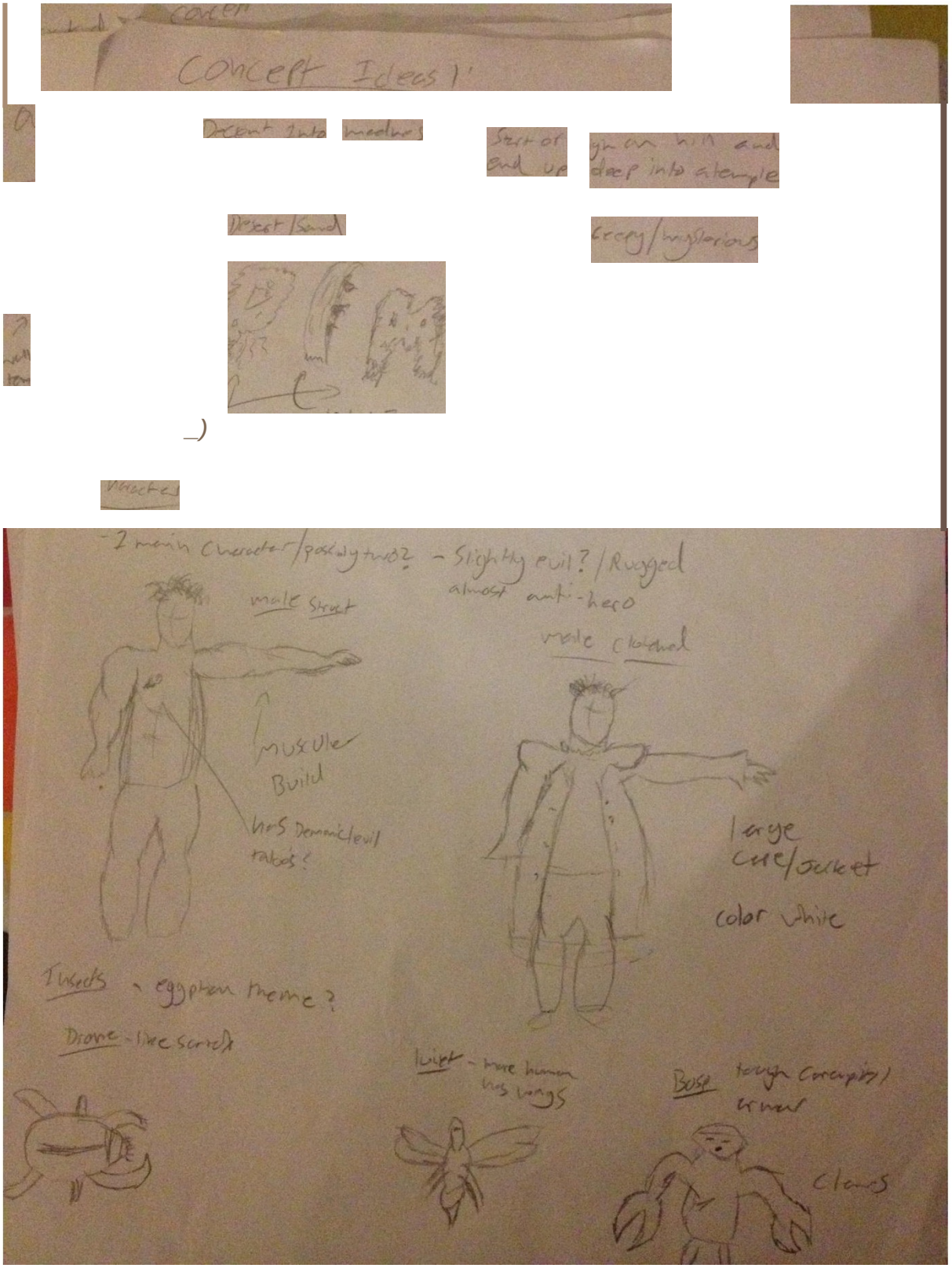


Fig 4. Concept sketch 2

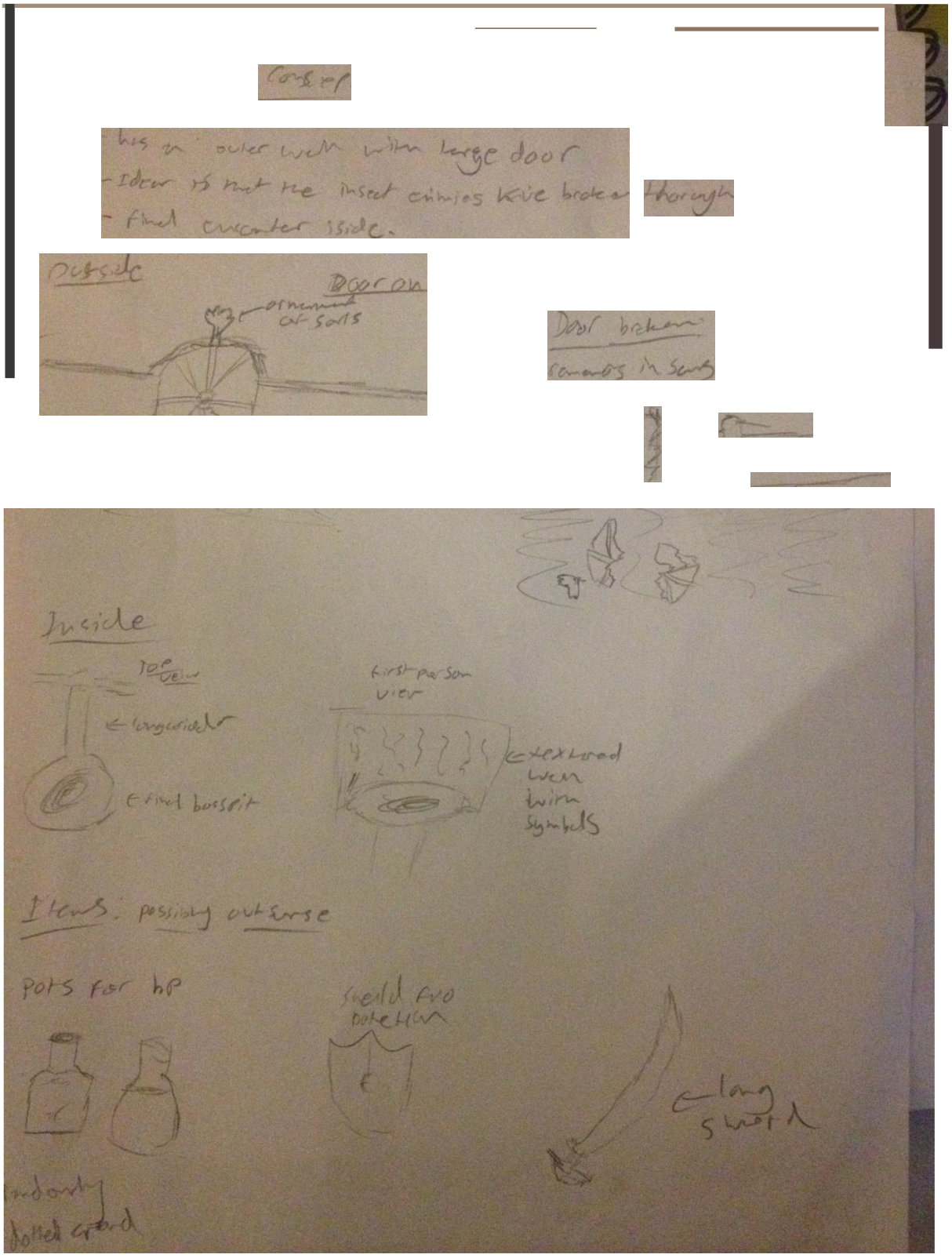


Fig 5. Concept sketch 3

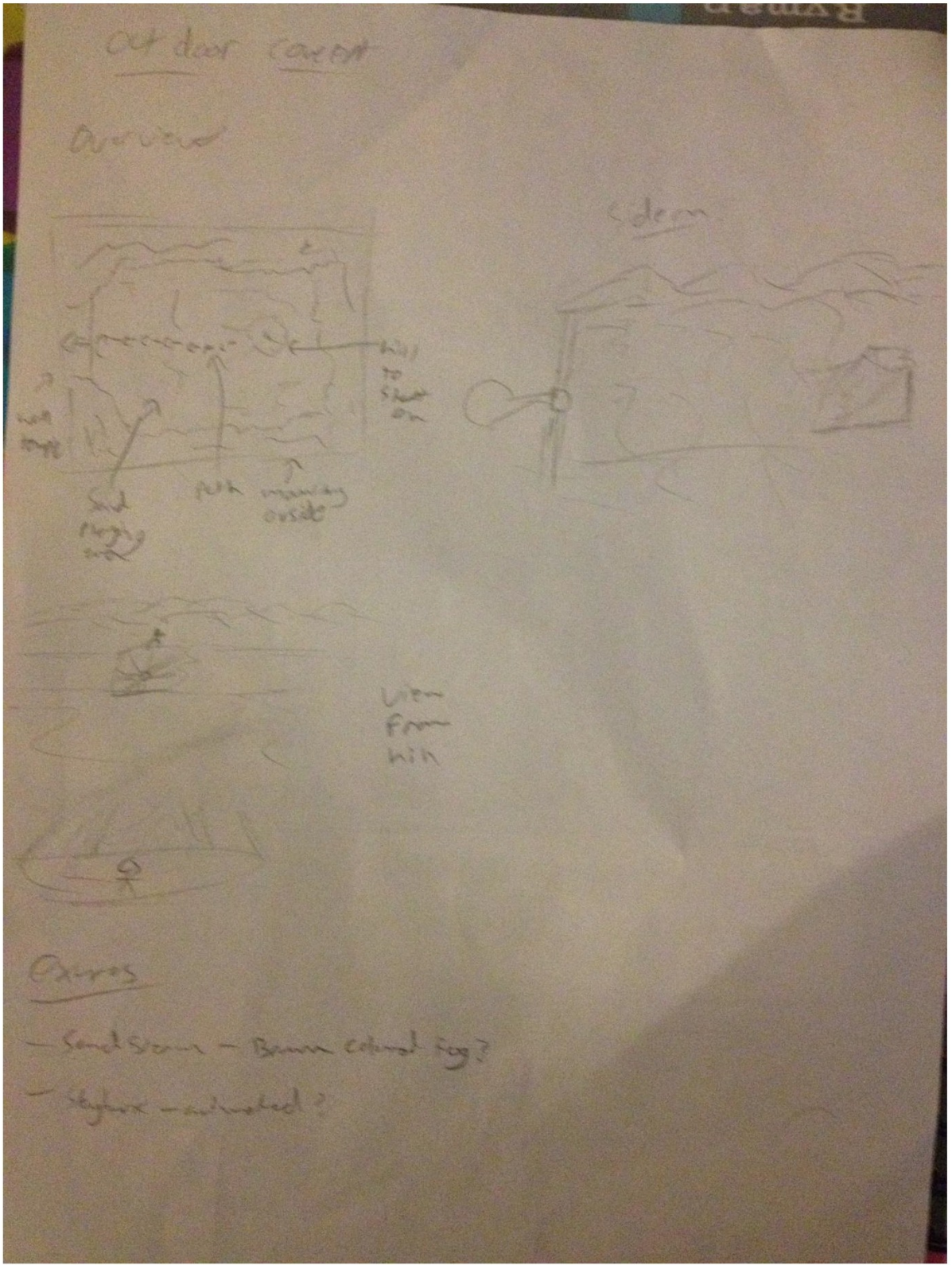


Fig 6. Character movement

```
    ///translating mesh
    Ogre::Vector3 transVector = Ogre::Vector3::ZERO;
    if (mKeyboard->isKeyDown(OIS::KC_I)) // Forward
    {
        transVector.z -= mMove;
    mSceneMgr->getSceneNode("CameraNode")->setPosition(MainNode->getPosition()+Ogre::Vector3(0,170,200));
    mCamera->lookAt(MainNode->getPosition() +Ogre::Vector3(0,75,0));
    }
    if (mKeyboard->isKeyDown(OIS::KC_K)) // Backward
    {
        transVector.z += mMove;
        mSceneMgr->getSceneNode("CameraNode")->setPosition(MainNode->getPosition()+Ogre::Vector3(0,170,200));
    mCamera->lookAt(MainNode->getPosition() +Ogre::Vector3(0,75,0));
    }

    if (mKeyboard->isKeyDown(OIS::KC_J)) // Left - yaw or strafe
{
    if(mKeyboard->isKeyDown( OIS::KC_LSHIFT ))
    {
        // Yaw left
        mSceneMgr->getSceneNode("MainNode")->yaw(Ogre::Degree(mRotate * 10));
        //CameraNode->rotate(MainNode->getOrientation(),Ogre::Node::TransformSpace::TS_PARENT);
    }
    else
    {


    {
        transVector.x -= mMove; // Strafe left
        mSceneMgr->getSceneNode("CameraNode")->setPosition(MainNode->getPosition()+Ogre::Vector3(0,170,200));
    mCamera->lookAt(MainNode->getPosition() +Ogre::Vector3(0,75,0));
    }
}
if (mKeyboard->isKeyDown(OIS::KC_L)) // Right - yaw or strafe
{
    if(mKeyboard->isKeyDown( OIS::KC_LSHIFT ))
    {
        // Yaw right
        mSceneMgr->getSceneNode("MainNode")->yaw(Ogre::Degree(-mRotate * 10));
        //CameraNode->rotate(MainNode->getOrientation(),Ogre::Node::TransformSpace::TS_PARENT);
    }
    else
    {
        transVector.x += mMove; // Strafe right
        mSceneMgr->getSceneNode("CameraNode")->setPosition(MainNode->getPosition()+Ogre::Vector3(0,170,200));
    mCamera->lookAt(MainNode->getPosition() +Ogre::Vector3(0,75,0));
    }
}
//translate
mSceneMgr->getSceneNode("MainNode")->translate(transVector * evt.timeSinceLastFrame, Ogre::Node::TS_LOCAL);
mSceneMgr->getSceneNode("CameraNode")->translate(transVector * evt.timeSinceLastFrame, Ogre::Node::TS_PARENT);
```

Fig 7. URL of project blog

http://dimnikunj.wordpress.com/

FigS. Screenshot of my modelingame with its test texture applied.

References

Autodesk (Skymatter Ltd, prior to acquisition by Autodesk), 2007. *Autodesk Mudbox,* San Rafael, California, U.S: Autodesk.

Rockstar North, 2004. *Grand Theft Auto: San Andreas,* Edinburgh, Scotland, UK: Rockstar Games.

Alias Systems Corporation, 1998. *Autodesk Maya,* San Rafael, California, U.S.: Autodesk, Inc..

Ant, T., 2007. *Advantages Disadvantages and Applications of Motion Capture.* [Online]
Available at: http://www.articlesbase.com/technology-articles/advantages-disadvantages-and-applications-of-motion-capture-217465.html
[Accessed 2 5 2013].

Bethesda Game Studios, 2011. *The Elder Scrolls V: Skyrim,* Rockville, Maryland, U.S.: Bethesda Softworks.

Billieux, J. et al., 2013. Why do you play World of Warcraft? An in-depth exploration of self-reported motivations to play online and in-game behaviours in the virtual world of Azeroth. *Computers in Human Behavior,* 29(1), pp. 103-109.

Blizzard Entertainment, 2004. *World of Warcraft,* Irvine, California, USA: Blizzard Entertainment.

Bruck S, W. P., 2009. Cybersickness and Anxiety during Simulated Motion: Implications for VRET. *Cyberpsychology & Behavior,* 12(5), pp. 593-593.

Duffy, J., 2007. *Game Programming, An Introduction.* [Online] Available at:
http://www.gamecareerguide.com/features/412/game_programming_an_introduction.php
[Accessed 11 03 2013].

ElizaWyatt, 2011. *Verrus, 3d Reference Sheet.* [Online]
Available at: http://elizawyatt.deviantart.com/art/Verrus-3d-Reference-Sheet-193351526
[Accessed 16 10 2012].

Filipowich, M., 2013. *In the Third Person: Against Player Authority.* [Online] Available at:
http://www.gamasutra.com/blogs/MarkFilipowich/20130124/185358/In_the_Third_Person_Against_Player_Authority.php
[Accessed 15 April 2013].

Finlay, I., 2010. *The drawbacks of Agile vs Waterfall.* [Online]
Available at: http://theisguy.wordpress.com/2010/03/29/drawbacks-of-agile-waterfall/
[Accessed 3 5 2013].

Flippies Inc., n.d. *History of Flip books.* [Online]
Available at: http://www.flippies.com/flipbook-history/
[Accessed 2 5 2013].

Gebhardt, N., 2003. *Welcome to the Irrlicht Engine.* [Online]
Available at: http://irrlicht.sourceforge.net/
[Accessed 1 5 2013].

Gebhardt, N., n.d. *FAQ.* [Online]
Available at: http://irrlicht.sourceforge.net/faq/#advantages
[Accessed 1 5 2013].

Haggerty, M., 2012. *The State of Mobile Game Development.* [Online]
Available at: http://www.gamesindustry.biz/articles/2012-11-28-the-state-of-mobile-game-development
[Accessed 1 5 2013].

Harris, J., 2007. *Game Design Essentials: 20 Open World Games.* [Online]
Available at: http://www.gamasutra.com/view/feature/1902/game_design_essentials_20_open.php
[Accessed 20 April 2013].

Jagex, 2001. *RuneScape,* Cambridge, United Kingdom: Jagex Games Studio.

Kuranes, T. & Chaster, 2007. *OgreBullet.* [Online]
Available at: http://www.ogre3d.org/tikiwiki/tiki-index.php?page=OgreBullet
[Accessed 19 4 2013].

Layton, M. C., 2012. *Agile Project Management For Dummies.* New york: John Wiley & Sons .

Lomow, G. et al., 1989. Winter simulation: Proceedings of the 21st conference. *Object-oriented simulation,* Volume 21, pp. 40-146.

Microsoft, 2010. *Microsoft Visual Studio 2010,* Microsoft Redmond Campus, Redmond, Washington, U.S: Microsoft.

Moore, M., 2011. *Basics of Game Design.* s.l.:A K Peters/CRC Press.

Mullen, T., 2012. *Mastering Blender.* 2nd ed. United States: Sybex Inc .

Myerson, R., 1997. *Game Theory: Analysis of Conflict.* New Ed ed. s.l.:Harvard University Press.

Nyitray, K. J., 2011. William Alfred Higinbotham: Scientist, Activist, and Computer Game Pioneer. *IEEE Annals of the History of Computing,* 33(2), pp. 96-101.

Ogre 3D tutorials, 2010. *Basic Tutorial 3.* [Online]
Available at: http://www.ogre3d.org/tikiwiki/Basic+Tutorial+3
[Accessed 25 March 2013].

OgreMax, 2013. *OgreMax,* s.l.: Ogre Max.

Pan European Game Information, n.d. *About PEGI?.* [Online]
Available at: http://www.pegi.info/en/index/id/33/
[Accessed 3 5 2013].

Patrick Salamin, D. T. F. V., 2006. The Benefits of Third-Person Perspective in Virtual. p. 4.

Richardson, H., 2012. *Pupils 'made more violent by computer games'.* [Online]
Available at: http://www.bbc.co.uk/news/education-17600454
[Accessed 3 5 2013].

Roughgarden, T., 2010. Algorithmic game theory. *Communications of the ACM,* 53(7), pp. 78-86.

Scott, R., 2003. Sparking life: notes on the performance capture sessions for the Lord of the Rings: the Two Towers. *ACM SIGGRAPH Computer Graphics,* 37(4), pp. 17-21.

Sony Computer Entertainment, n.d. *SCE DevNet.* [Online]
Available at: http://www.scedev.net/
[Accessed 1 05 2013].

Suess, M., 2006. *C++ vs. C# - a Checklist from a C++ Programmers Point of View.* [Online]
Available at: http://www.thinkingparallel.com/2007/03/06/c-vs-c-a-checklist-from-a-c-programmers-point-of-view/
[Accessed 1 5 2013].

The Ogre Team, n.d. *List Of Libraries.* [Online]
Available at: http://www.ogre3d.org/tikiwiki/tiki-index.php?page=List+Of+Libraries
[Accessed 4 3 2013].

The OGRE Team, n.d. *OGRE.* [Online]
Available at: http://www.ogre3d.org/
[Accessed 14 11 2012].

Thurrott, P., 2010. *Xbox 360 vs. PlayStation 3 vs. Wii: A Technical Comparison.* [Online]
Available at: http://winsupersite.com/product-review/xbox-360-vs-playstation-3-vs-wii-technical-comparison
[Accessed 23 04 2013].

Torpy, A., n.d. *L3DT,* s.l.: Bundysoft.

Torus Knot Software Ltd, 2000. *licensing-faq.* [Online]
Available at: http://ogre.svn.sourceforge.net/viewvc/ogre/trunk/COPYING?revision=9087
[Accessed 17 1 2013].

Ubisoft Montreal, 2007. *Assassin's Creed ,* Montreal, Quebec, Canada: Ubisoft Montreal.

Visceral Games (formerly EA Redwood Shore), 2008. *Dead Space ,* Redwood City, California, USA: Electronic Arts.

Ward, J., 2008. *What is a Game Engine?.* [Online]
Available at: http://www.gamecareerguide.com/features/529/what_is_a_game_.php
[Accessed 17 04 2013].

Wikipedia, n.d. *Muscle.* [Online]
Available at:

https://upload.wikimedia.org/wikipedia/commons/thumb/e/e5/Muscles_anterior_labeled.png/350px-Muscles_anterior_labeled.png
[Accessed 18 10 2012].

Wolf, M. J. P., 2008 . *The Video Game Explosion: A History from PONG to PlayStation and Beyond.* London: Greenwood Press.