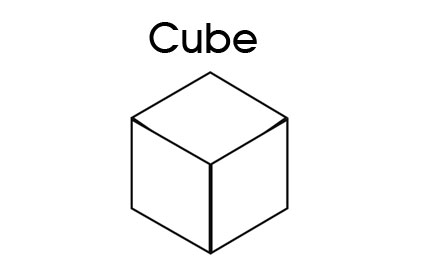
Final Year Report

Arrows

A 3D C++ First-Person Shooter Game



SB684 - 12845703

Samuel William Bowen

Andrew Blake – Aidan Delaney

Computer Science (Games) - BSc(Hons)

Table of Contents

[Introduction 3](#_Toc418688374)

[Aims & Objectives 3](#_Toc418688375)

[Responsive and smooth player controls 3](#_Toc418688376)

[Realistic bow and arrow style gameplay 3](#_Toc418688377)

[Fun and engaging gameplay 3](#_Toc418688378)

[Interesting and dynamic skill system 4](#_Toc418688379)

[Attractive graphics 4](#_Toc418688380)

[Background Research 4](#_Toc418688381)

[Which language to use 4](#_Toc418688382)

[C# 4](#_Toc418688383)

[Python 5](#_Toc418688384)

[C++ 5](#_Toc418688385)

[The external libraries available 5](#_Toc418688386)

[SDL 5](#_Toc418688387)

[GLM 5](#_Toc418688388)

[GLUT 6](#_Toc418688389)

[GLEW 6](#_Toc418688390)

[SOIL 6](#_Toc418688391)

[The Type of game 6](#_Toc418688392)

[Isometric 2.5D 6](#_Toc418688393)

[3D 6](#_Toc418688394)

[Project Scope 7](#_Toc418688395)

[Feasibility 8](#_Toc418688396)

[Analysis 9](#_Toc418688397)

[Why a game? 9](#_Toc418688398)

[Programming language 9](#_Toc418688399)

[Graphical API 10](#_Toc418688400)

[Game play 10](#_Toc418688401)

[Game style 10](#_Toc418688402)

[Methodology and Planning 10](#_Toc418688403)

[Scrum 10](#_Toc418688404)

[Design 11](#_Toc418688405)

[Aesthetics 11](#_Toc418688406)

[UML 12](#_Toc418688407)

[Development 13](#_Toc418688408)

[Setting up 13](#_Toc418688409)

[The basics 13](#_Toc418688410)

[Making Games with Ben 14](#_Toc418688411)

[Creating the 3D Camera 14](#_Toc418688412)

[Arrows & gravity 15](#_Toc418688413)

[Collision detection, jumping and terrain 16](#_Toc418688414)

[Enemy Agents 16](#_Toc418688415)

[Game Play 17](#_Toc418688416)

[Progress Assessment 18](#_Toc418688417)

[Progress Made 18](#_Toc418688418)

[Problems encountered 18](#_Toc418688419)

[Solutions 19](#_Toc418688420)

[Lessons learned 19](#_Toc418688421)

[Critical Evaluation 19](#_Toc418688422)

[Game play 19](#_Toc418688423)

[Graphics 20](#_Toc418688424)

[Artificial Intelligence 20](#_Toc418688425)

[Deployment 21](#_Toc418688426)

[Aims & Objectives 21](#_Toc418688427)

[The Project Log 21](#_Toc418688428)

[How to play 21](#_Toc418688429)

[References 22](#_Toc418688430)

# Introduction

**First person shooter games have always been popular and lots of them have been extremely successful in the games industry. There are hundreds of different types of FPS games each with their own styles, graphics and gameplay. This report is going to go through the different stages of development as I try and design my own first person shooter style game using C++.**

# **Aims & Objectives**

## Responsive and smooth player controls

* Research different input libraries, there are multiple libraries currently available for use when programming games and applications. I need to find out which one will be best for me when handing input from the player in the game.
* Research 3D camera implementation methods, there are multiple ways to implement a 3D camera. I need to find a method which will allow me to create smooth, responsive and realistic camera for my FPS game.

## Realistic bow and arrow style gameplay

* Research how a bow and arrow are used in real life. I need to be able to simulate how a person would draw and fire an arrow using a standard bow with methods and functions.
* Think about how physics applies to a bow and arrow. An arrow won’t fly forever, I need to be able to apply physics and collision detection to an arrow to make it as realistic as possible.

## Fun and engaging gameplay

* Research top played games. I need to look at all of the top selling FPS style games and try and find out how they have been so successful and then try and use some of their methods.
* Research game content. I need to research lots of different features I might be able to implement in my game, I don’t want my game to be boring and run out of new things to do within the first 5 minutes.

## Interesting and dynamic skill system

* Research current talent/skill systems in existing games. Lots of games have implemented skill trees and talent systems where a player can completely customise their experience through unique skill selection.
* Think about what types of skills and talents I could incorporate into the type of game I am going to create. I need to create skills which will fit into the style of game I’m creating.

## Attractive graphics

* Research OpenGL. I need to learn the ins and outs of OpenGL as it is the best API for the C++ language. OpenGL lots of amazing features that I can take advantage of when creating my game.
* Research 3D model loading. To be able to have some good graphics and scenery I need to know how to import 3D models that I will be able to use in my game.
* Research texture loading and mapping. I need to learn how to import and map textures to objects within a 3D world, this will allow me to give my game a better environment to play in.

# Background Research

## Which language to use

When it comes to game development there are only a few programming languages which are relevant. It is true that almost every programming langue will fill a different role, but there are a few that are especially good for game development. Each programming language is different in its own right, and all have their own advantages and disadvantages.

### C#

C# has a high-level approach, which makes it much faster to develop with, but restricts the flexibility and runtime performance of the software. C# contains lots of features that makes software development faster and easier, it does its own garbage collections and doesn't require the use of pointers (but is optional). C# is also supported in the amazing game engine, Unity. Unity is extremely powerful and allows for amazing results with minimal effort, but does not promote learning.

### Python

While researching Python I found that it would not be a suitable for a large 3D game. Python is instead, a very good scripting language for developing game logic, missions and artificial intelligence. As it is an interpreted language it is generally faster than higher level languages such as C# but will be much slower than a low level language like C++.

### C++

When researching C++ I found that it is extremely powerful and allows the user almost full control. It allows you to do anything that you can imagine, but it also allows you to potentially create any bug. With C++ it is important that you know how to clean up after yourself and amend all of your mistakes. C++ is an extremely hard language to start off with and can be very confusing and unforgiving, thanks to my previous experience with the language I found C++ to be a good language to program with.

When looking for resources for C++ I found hundreds of websites, tutorials, guides and forums dedicated to C++ game development. The amount of resources are endless, along with the amount of external libraries that are available to the C++ language. C++ also supports the two major APIs, OpenGL and DirectX.

## The external libraries available

When developing a game it would almost be impossible to do everything yourself, that is why libraries have been created, to allow developers to not have to create and re-write mundane functions. I have listed a few libraries which I thought I might need before I started my project, as well as some libraries which I found while developing.

### SDL

Simple DirectMedia Layer is a cross-platform development library which has been designed to provide easy access to low level input, such as audio, keyboard, mouse, joystick and graphics hardware via OpenGL and Direct3D [9]. SDL supports most of the major operating systems as well as other platforms. SDL has been written in C and works natively with C++.

### GLM

OpenGL Mathematics is a mathematics library which has been specifically designed for “graphics software based on the OpenGL Shading Language (GLSL) specification” [5]. It is a library which provides multiple functions which help with the design and creation of 3D programs, providing functions which can create cameras and manipulate objects in a 3D space.

### GLUT

“GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs”[8]. GLUT provides a portable API which is designed for creating small to medium sized programs which can also be deployed across all PC and workstation operating systems.

### GLEW

“The OpenGL Extension Wrangler Library (GLEW) is a cross-platform open-source C/C++ extension loading library”[7]. GLEW provides an efficient run-time mechanism for determining which OpenGL extensions are supported on the target platform.

### SOIL

Simple OpenGL Image Loader is a very small and simple cross-platform image loading library which can read a variety of useful formats, such as BMP, JPEG, PNG and others. Its primary function is to upload textures into OpenGL.

## The Type of game

### Isometric 2.5D

Isometric 2.5D games are becoming increasingly popular among indie gamers and game developers. Isometric 2.5D style of games allow the developer to create a game in the 2D world space coordinate system, but give it all of the art style of a 3D game. This makes game play development much easier as the developers do no need to account for the 3rd dimension.

### 3D

3D games have always been popular among game developers, especially first/third-person shooters and MMORPG style games. They give the game a 3rd dimension to be played in and allow for much better gameplay.

Games such as the Call of Duty franchise have used the 3rd dimension to allow for better gameplay throughout their games. They take advantage of the 3rd dimension (height) by designing maps and levels which include terrain with varying heights, to allow the players to gain a height advantage on other players or enemy agents. Games such as Guild Wars 2 and World of Warcraft use the 3rd dimension to give the players a more immersive experience. They include exploration achievements and beautiful huge open worlds for the players to explore.

# **Project Scope**

I planned to implement a single player, 3D, first person shooter style game. It’s going to be a Player versus Engine (PvE) game, where the player will fight against computer controlled agents. The game mode which I am planning to implement is called “Horde”, this is where the player fights off waves of enemies, until they lose. The game will start off slow, where the player has minimal opposition, but the “waves” of enemies will slowly get harder, through; strength, speed, intelligence or volume. Eventually the player will lose, but he will have generated a score and gathered experience from his success, depending on how fast he has cleared each wave and how many waves he managed to complete.

The game I am going to create will be implemented using the C++ programming language. I intend to build and distribute my game on Linux, using the open-source build software called Cmake. Ubuntu works extremely well with Git which I will be using to track, control and optimise my workflow. On the Ubuntu operating systems I will be able to use a g++ compiler available for C++ which will allow me to compile and run my program in the terminal. I plan to use the OpenGL application programming interface for rendering the 3D graphics in my game. I also plan to take advantage of many of the open source libraries available to the C++ language.

The game that I am planning to create is going to be single player, I have a limited amount of time and I feel that I my project is already quite time consuming. By adding multiplicity into the game I feel like I would not have enough time to complete most of the main features in my game. I have also not done any networking with C++ applications so I would need to research and learn how to implement them from scratch. Networking also include lots of security issues that I would need to look into.

I will be developing a 3D environment as I have had lots of experience using OpenGL in the past which is an extensive C++ API for graphical implementations. Having previously developed a 3D game using OpenGL it should make it easier for me to develop this game and focus on the game features instead of the graphical features. I know how OpenGL represents 3 dimensional space and I know how to manipulate it when creating a 3D world, using my knowledge I should be able to create a 3D game.

The original idea of my game was going to involve bow and arrow weaponry, I thought about what types of games I could create which would involve this and I felt a first-person shooter would best emulate using a bow and arrow. First person style games limit you to one camera position, the player’s eyes. The camera sees and displays what the player’s character can see, limiting my game to only one style of camera, whereas with a 2.5D top down style game that player would have greater control over the camera, or even a 3rd person shooter where the player would have a larger view over the character’s shoulder.

By creating a 3D game I have to create 3D map, this means that I need to design a world, or space, where the player will be able to move about freely. Most 3D games would design their worlds in 3D design software, such as 3Dmax or Blender, and then import their maps into the game. I plan to limit my game to a cube style world by using OpenGL primitives to design and create my world. As I have created a horde style game the map doesn’t need to be too large, and the map is going to be reused constantly so a simple map is ideal.

I will limit my game to only having two different types of enemies, designing the artificial intelligence behind enemy agents is very complicated and time consuming. Therefore I have opted to only attempt to create two types of enemies, this gives the game a bit of diversity as there isn’t just one enemy with the same behaviour and movement/fighting patterns. I have decided to try and create a melee (close combat) type agent which will get up close, and a ranged type agent which will try and pick off the player with their long range weaponry.

# Feasibility

With my final year project time is a major issue, I have multiple modules that I need to complete and I have a very limited amount of time to get them all done. I need to allocate my time efficiently to be able to get as much work done as possible, and make sure my final year project meets my aims and objectives. Time has been one of the major feasibility issues that I have needed to address, but my knowledge of some of game design has also been an issue.

I believe that it was for the best that I made my game single player instead of multiplayer. By not having any experience with networking in C++ I felt that it would be far too time consuming for me to attempt to create a multiplayer game. I had never done any networking before starting my 3rd year of university, and for a major part of this project to be based around networking I felt that it would not be feasible project.

Originally I wanted to design, create and animate my own 3D objects and models for all of the characters and objects in my game. I felt that if I was going to create 3D models for my project I would have to make sure that they were good enough, I didn’t want to use semi-completed, half-arsed models. I realised that creating my own models was unfeasible. I would need to create multiple models for my project, and each model can take weeks to create, along with the animation that I would have to design for each of the moving enemy characters.

# **Analysis**

## Why a game?

The first major point of analysis is why I have decided to create a game instead of something more original. The reason is that I love games, from playing to creating them, games have and will always be a major part of my life. My ambition is to get into the games development industry, and I believe that creating a game and having something to show, that I’m proud of, will help me get onto that career path.

## Programming language

My game has been created using C++, the main reason for this is that I am comfortable with C++, I’ve been using it for over a year now and feel that I can program quite well with it. There are many languages which are good for game development, but C++ is by far one of the best. Having used C++ last year to create a game I had some work that I could fall back on and revise to help me with my final year project.

C++ is a great object orientated game development language which supports multiple external libraries. It also has amazing terminal compiler, g++, which has allowed me to develop in Ubuntu using text editors, removing the need for me to use clunky IDEs (Integrated Development Environments) such as Visual Studios.

## Graphical API

Why OpenGL? The main reason for me using OpenGL is that it is the only C++ API that I have been taught. I feel comfortable using OpenGL and have had almost two years of experience with it. Although experience is important it’s not the only reason for me using OpenGL, it has a few very important features that makes it irreplaceable. OpenGL is cross-platform, making it easier for me to design a game which can be deployed on multiple operating systems.

## Game play

I decided on a game style quite quickly; horde. Horde style games are very engaging, fun and hard! The horde style game mode can really test a player’s skill, and keep them entertained, this is because you can never win. This allows the game mode to have replay-ability, as the player can never complete the game, only improve on what they have previously achieved.

A main reason I decided to create the horde game mode was that it doesn't require as much time to develop and create as many other games. A good horde game can have 2-3 unique arena style maps, which would be extremely small when compared to the maps and environments of the other popular game modes, such as RPGs. Horde games also do not require an extensive storyline which are used to guide players through the game, and give a reason to the game.

## Game style

The game style, I decided to go for a 3D style game because I thought it would be difficult and fun to create. I had already developed an extremely simple and static 3D game but did not get to explore lots of the features a 3D scene has to offer. OpenGL provides an extremely good API for developing 2D and 3D games and I was already familiar with it. When designing a game in 3D you can create a brand new world, where you can place the player and have them immerse themselves in an alternate reality which gives you a lot of freedom when designing the world and allows you to be creative.

# Methodology and Planning

## Scrum

Scrum uses real-world progress — not an uninformed forecast — to plan and schedule releases. Time is divided into short work cadences, known as sprints, typically one week or two weeks long. At the end of each sprint, stakeholders and team members meet to see a demonstrated potentially shippable product increment and plan its next steps. This allows direction to be adjusted based on completed work, not speculation or predictions.

I will not be working with anyone else except for myself, so at the end of each week I will be reviewing what I have done and been able to achieve. I will be able to determine if I am making the progress I want to be making or am falling behind. The scrum methodology will allow me full control over how the project is being managed, and will allow me to make changes on the fly.

# Design

## Aesthetics

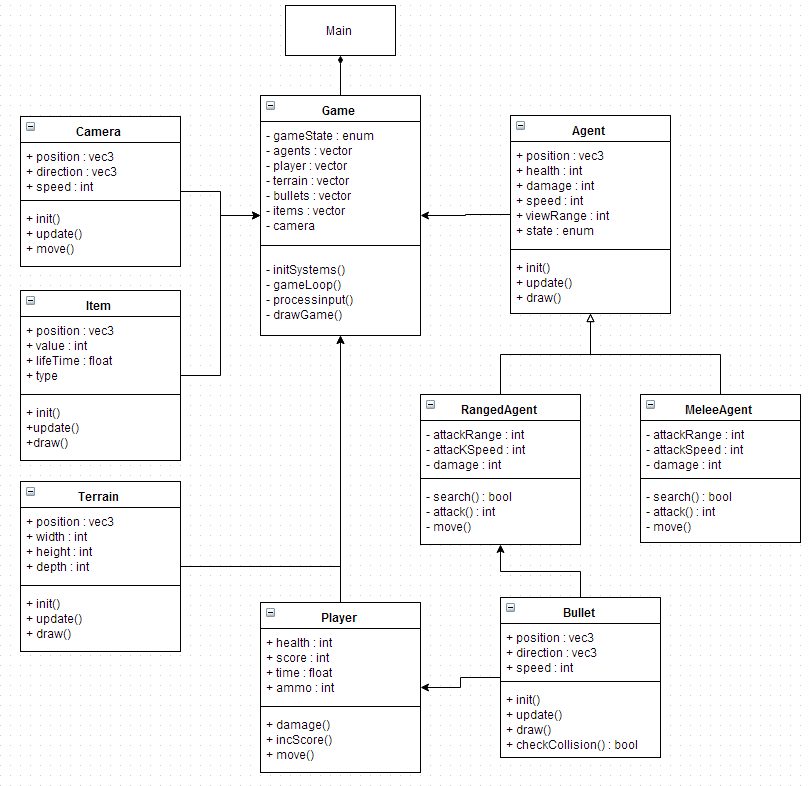
The game is going to be 3D, just like most first-person shooter games the player will be inside of the 3D world. OpenGL allows me to design a camera which can portray a 3 dimensional environment, OpenGL also allowed me to create 3D models and objects which can be used in the game as assets and enemies.

I found that it would be extremely unfeasible for me to design my own models, so I have decided to keep the style of the game simple by only using primitive types which can be easily created using OpenGL. OpenGL allows you to create 3D objects by setting their position coordinates in the 3d space, and using triangles to draw shapes, such as cubes, cuboids, and prisms.

The world itself is an arena, the player will be contained in the arena by walls and floors (or the lack of floors), the shape and size of the map has been predefined and will be static. There will be multiple floors in the arena which will allow the player to gain a height advantage.

## UML

A simple UML diagram of how I believe the classes to be structured and the relationships between them. As well as some fundamental variables and methods that might be included with the classes.



# Development

My development history can be seen at: <https://github.com/CrazyRagdoll/Arrows/commits/master>

## Setting up

I had never developed a 3D game from the very beginning before, the last time I designed a 3D game using C++ I was given a very extensive boilerplate that contained a lot of the fundamental code.   
Starting from scratch meant that I needed to learn how to implement the basics, but before I started programming I decided to add in some external features that helped me with my development.

There are multiple external programs & websites that are available to use for free when developing software. An extremely important thing to have while developing software is to have extra repositories. While developing my game I decided to use GitHub which allowed me to backup and track all of the progress and work that I do.  
GitHub managed to save my ass while developing my final year project as halfway through development my laptop broke, meaning I lost all of my progress that I had made on my laptop. Fortunately all of my work was backed up on GitHub which allowed me to pull my repository and start work without losing much progress at all.

Another good feature that I decided to use and set up was TravisCI, a continuous integration and deployment service, which works with GitHub to build the software online, on multiple platforms to see if the software works on multiple operating systems. This software has been very useful as it has allowed me to see if my builds will work on operating systems other than Ubuntu.

## The basics

After I had set up my GitHub repository and got Travis out of the way the first thing that I created was a SDL window. To do this I had to create a main.cpp file and a Game class. In main I included the Game class and created an instance of a game, I also started to design the fundamental game functions, such as initializing systems. To create the window I decided to use SDL2, an excellent C++ library, which made it extremely simple to initialize and create a window on screen.

The next feature I added to my project was game states, I edited the Game class to allow for an enum class of two game states, play and exit. I then added two new functions, processInput and gameLoop, game loop would be responsible for handling all of the game play features. ProcessInput would be used to track all of the keyboard and mouse inputs. While the game was in the game state of play the project would continue to process the game, otherwise it would quit.

Once I had the basis of the game set up I decided to add a function which would make it easier for me to debug the game. I would call the fatalError function when something went wrong, it would print a string of the error and exit the game. I initially used this to make sure that I could successfully set up the SDL window, GL context and initialize glew.

## Making Games with Ben

For some of my project I was following an extremely good series of tutorials by a programmer called Ben Arnold. Unfortunately his tutorials were for 2D games, so I couldn't really follow the whole series, but I was able to get some great features out of the tutorials. The tutorials really helped me understand C++ better, the author really explained everything he was doing, and how it all worked.

Therefore some of my game engine has been adapted from the making games with ben tutorials. The classes Errors, GLSLprogram, GLTexture, IOManager, ImageLoader, Timing and Window are the main classes which I have utilized from the tutorial, but they have all been expanded upon to make them work with my project.

## Creating the 3D Camera

For a first-person shooter game the camera is the most important part, the player will spend almost all of their game time looking through the camera. My engine consisted of a simple 2D camera which would convert screen coordinates to world coordinates to display sprites, but I wanted a 3D world and camera, so I scrapped most of the old camera class and rebuilt it.

The original 3D camera which I developed consisted of a few variables to give it a position in the world and some view parameters. It also had simple update functions to create and update the projection and view matrices. Getting my head around matrices and the 3D world space coordinate systems was quite a challenge but I managed in the end.

In the camera class is where the GLM (OpenGL Mathematics) library really came in handy, the GLM library consists of two functions which make it extremely easy to create the projection and view matrices. GLM perspective allows you to create a projection matrix from four values; field of view value, width to height ratio, the near z-plane and finally the far z-plane. GLM lookAt allows you to create a view matrix from three values; the position of the camera, the direction the camera is looking and which way is up.

I decided to treat the camera as the player, and simulate movement and collisions for the player through the camera class. I created multiple movement functions as well as a rotation function which would take input from the player, and depending on which key was pressed, change the positional and/or directional vectors accordingly and then update the view matrix. This allowed me to create a player controlled camera which would be used as the player's character.

## Arrows & gravity

After I had implemented the camera and all of the movement functions I then decided to start developing the 2nd most important feature of a FPS, the shooting. I was creating bow and arrow weaponry so I created the “Arrow” class which I use to create, draw and update the bullets. In the main game I decided to use a vector of arrows, every time the player shoots a new arrow would be created and pushed back into the vector. The vectors made it easy for me to initialize, draw and update all of the arrows at the same time.

In my game I decided to bind the shoot button to the left mouse click, whenever the player would press the LMB (left mouse button) it would fire an arrow. But I wanted it to seem much more like the player was drawing an arrow, so I came up with the idea to give the arrows power. So now when the player holds down the LMB the power of the shot increases, when the LMB is released the arrow is fired. I managed to get the direction the arrow was shot, and the position it was shot from, by using the position and direction of the camera.

I now had implemented the ability to fire arrows, but they would always fly off into the distance, so I decided to have gravity act upon them. Using simple SUVAT equations (v = u + at) I was able to take the initial velocity of the arrows and add the time multiplied by the acceleration of the arrow to find its final velocity. Applying this equation to the y-axis I was able to get the arrow to dip as it flew through the air.

## Collision detection, jumping and terrain

The movement and shooting were implemented, great! But there was so much more that needed to be added to make it a playable game, for starters I needed collision detection. As I am using primitive types it was extremely easy for me to implement collision detection between all of the different objects, I created simple a bounding box around each object using their position and width/height/depth. Whenever a part of one of the bounding boxes entered another one they were colliding.

Once I had collision detection sorted, and could add simple bounding boxes to objects I decided to implement a floor and jumping feature to the camera. Using collision detection I could determine whether the player was standing on the floor or not, if he wasn’t then he must be in free fall, so gravity should act upon him, pushing him down. Using the floor I could determine the size of the battle area, and make certain places out of bound.

## Enemy Agents

I had implemented all of the game engine and most of the functionality. I had a player that could move around freely, with the ability to shoot, I had arrows that could collides with objects, I had a world where all of this was going to take place. The next thing that I needed was to add was some game play, something to shoot at!

I created the agent class, a base class which would handle all of the initialization and drawing for all of the enemy agents. I decided on two different types of enemies, a melee agent and ranged agent, which would have unique movement and attack patterns, as well as different statistics like health and damage.

The way I have implemented artificial intelligence in the agents is by using an enum class of different the different states that the agents can be in. These states would determine what the agent would be doing; patrol, chase, scout and attack, however, what the agent does in each states is unique to the agent.

The melee agent starts in the patrol state, where he will move back and forth over a point until he meets the requirements to enter a different state. While the melee agent in patrolling he will constantly look for the player, when he finds the player he will then enter the chase state. I have used the Barycentric coordinate system to give the agents a field of view, it creates a triangular zone in front of the agent and if the player is inside that zone then the agent can “see” the player.

While in the chase state the melee agent will constantly check to see if he in range to attack the player, if not then he will move closer to the agent. The melee agent also moves at an increased speed while chasing the player. When the player does finally enter the melee agent’s range of attack he will enter the attack state.

In the attack state the melee agent performs an “attack animation” where (in a game with proper models and animations) the agent would attempt to hit the player. While in the attack state the agent has a reduced movement speed, which gives the player the ability to dodge the attack. Using a simple attack speed variable I can regulate the time which the player has to evade the attack, but if the player is still in the attack range of the melee agent at the end of its attack animation the player will take damage.

The ranged agent works differently to the melee agent, instead of using the patrol state the ranged agent starts in the scout state. While in this state the ranged agent stands idle and randomly looks north, south, east and west every couple of seconds in search of the player. The ranged agent has a much larger view distance, and can therefor “see” the player from further away. Once the ranged agent finds the player he enters the chase state just like the melee agent, but he has a much larger attack range and different style of attack.

Once the ranged agent is in attack range of the player he will enter the attack state and perform his attack animation. Instead of checking to see if the player is still in attack range at the end of the animation he will always attack, this is due to the fact that he uses a ranged attack. At the end of the attack animation the enemy agent will fire an arrow, just like the player does, if the arrow collides with the player, the player will take damage.

## Game Play

I had all of the elements of a game in place, all I needed to do now was to add an objective to the game, I needed to give the player something to achieve. I started off by creating the player status class which would hold the health, ammunition, time and score of the player, which would be tracked throughout the game. The player’s score would be incremented over time, depending on how well he is performing within the game.

I implemented the Horde game mode, this means that enemies would be coming in waves to try and kill the player, so I created a simple function which would spawn a certain amount of enemy agents, depending on which wave the player is on. By parsing in the wave number I am able to decide how many of each type of enemy spawns and by using a random function I am able to give them a random position on the map.

Once all of the enemies have been killed the player is given a couple of seconds to recuperate and get back into position to fight the next wave. Depending on how fast the player completed the wave he will generate a score, his score is also increased every time he kills an enemy agent. His life total is not reset every wave and has no way to regenerate, he will lose life every time he get hit by a melee agent’s attack, or an arrow from the ranged agent.

In the game the player has a limited amount of arrows, every time he fires his bow an arrow will be consumed. However, to keep the game playable the enemy ranged agents will drop a stack of two arrows when they have been killed, allowing the player to keep on fighting.

## Progress Assessment

### Progress Made

I have made tremendous amounts of progress, and have learnt so much from completing this final year project. I feel I have a far greater knowledge of the C++ language and have learnt multiple different programming techniques which could be used across all programming languages.

I was able to create at the basis of a first-person shooter and I am extremely pleased with what I have been able to create on my own in the past few months. Even thought I was unable to complete my project to my aim and objects I am pleased with the results.

### Problems encountered

I encountered multiple problems while completing my final year project, some were to do with my lack of knowledge of the C++ language, and some were just silly mistakes.

My biggest problem was attempting to convert a 2D camera into a 3D camera without much knowledge of actually creating a 3D camera. I spent a lot of time researching the different methods you can implement a 3D camera and tried to find the most effective and efficient method.

I had lots of trouble attempting to get my texture loader to work efficiently, but unfortunately I was unable to create a texture loader that would not completely destroy the frames per second rate of my game. This is the main reason why my game does not contain many textures in it.

I attempted to add game states into the game, where the player would be able to pause the game by pressing the esc key, and resume by pressing it once more. I managed to get everything to pause except for the enemy agents, who would still move around freely.

### Solutions

I was able to overcome the 3D camera problem by doing lots of research, I think I spent around 10 hours looking at all of the different camera implementations. I researched the different ways that I could create the perspective and view matrices, and found that the GLM library was far fear the easiest, and best ways to implement the camera.

### Lessons learned

I learnt that diving straight in and implementing without doing enough research can lead to a lot of wasted time. I must have spent hours writing code that didn’t work and could have easily fixed if I looked into it a little more. I didn’t really leave in any code that I found to be useless to me later on, but I know I should have left in the functions and methods that I attempted to implement. An example of a failed method was the rotation function that I created through some maths research for the arrow class, the idea was to rotate the arrow while in flight, to make it look like it was actually dipping with gravity.

# Critical Evaluation

Studying a games course I felt that I needed to do a project that was games oriented, I felt that I needed to take all of the modules that I have learnt over the past couple of years and try and implement that knowledge into my final year project. So, I decided to make a game.

## Game play

Game play was a fundamental part of my project, I wanted to create a game that is not just fun to play, but has great game play and controls. I wanted to create smooth and reactive player controls, I wanted people to feel like they have full control over the character. I believe that I have managed to do this, the player is free to move around the arena using the key binds available. The game responds to the player’s key presses in real time and has minuscule delay between the input and outputs.

I feel like I was able to utilize the knowledge I gained from last year’s games development module, and this year’s internet games design to really add some good game play to my game. The modules taught me how to think like a games developer, and gave me the logic that I needed to program a game.

Graphics

The graphics in my game are poor, I honestly feel that the graphics have really let me down. Initially I wanted to create an attractive game that has amazing graphics and animations, but this fell to the wayside as I really started to develop the game and realised how time consuming game development really is.

I attempted to liven up the graphics using a texture loader and giving all of the primitive types different skins, and giving the world some colour. Unfortunately my texture loader was very poorly optimized and would cause the game to have massive frame rate drops when dealing with too many textured objects.

## Artificial Intelligence

The AI module I have been studying in the 2nd semester of this year has really helped me with the development of my enemy agents, and I am actually quite proud of my AI that I have implemented into my enemy agents. I believe that they work quite well, and can become quite hard to deal with, causing the game to become increasingly difficult, but also allow for some “outplay” potential, where the player can skilfully overcome them.

I would have preferred to create much smarter enemies, such as enemies which would try to avoid fire by hiding behind terrain, or attempt to gang up on you with the other agents. I fell that I have really only scratched the surface of enemy agent artificial intelligence and could have made them much smarter.

## Deployment

I was only able to get my project to run on Ubuntu. I would have liked to get it to also deploy on windows, and then maybe attempt to port it to a mobile operating system. I didn’t spend enough time working between Ubuntu and Windows 8.1, and eventually stopped trying to develop on windows all together as I was having compilation problems with Visual Studios.

## Aims & Objectives

I believe that I have managed to achieve most of my aims and objectives to my fullest extent and am quite happy with the results. However I have not managed to complete some of them; I wanted to have attractive graphics and I have been unsuccessful in producing a game with good 3D models and animations. I also planned to have implemented an experience system & skill point system which would allow the player to customize their character and change up the gameplay. I wasn’t able to fully complete other parts of my aims and objectives so the talent system fell to the wayside.

# The Project Log

I haven’t really kept a physical log of the work that I have been doing over the past year, instead I have been using a GitHub repository which has been collecting all of the changes I have been making to my project:

<https://github.com/CrazyRagdoll/Arrows/commits/master>

I did make some notes with some ideas of what I wanted to develop at the beginning of the year and will be included with the report.

# How to play

Currently only works on Ubuntu 12.04/14.04.  
Download from <https://github.com/CrazyRagdoll/Arrows/archive/master.zip> or access from USB provided.  
Required libraries & extensions: GLM, GLEW, SDL2.0, SOIL, OpenGL & Cmake

Open Ubuntu terminal and navigate to ../Arrows/Build/  
Enter “cmake CMakeLists.txt” to build project  
Enter “make” to compile project  
Enter “/Arrows” to run the project

# References

[2] - Blog.wolfire.com,. 'Why You Should Use Opengl And Not Directx - Wolfire Games Blog'. N.p., 2015. Web. 19 Jan. 2015.

[3] - Developer.nvidia.com,. 'Opengl'. N.p., 2015. Web. 7 Dec. 2014.

[4] - GameDev.net,. 'Language Comparison - Game Programming'. N.p., 2015. Web. 2 Jan. 2015.

[7] - Glew.sourceforge.net,. 'GLEW: The Opengl Extension Wrangler Library'. N.p., 2015. Web. 1 May 2015.

[5] - Glm.g-truc.net,. 'Opengl Mathematics'. N.p., 2015. Web. 29 Mar. 2015.

[8] - Group, Khronos. 'GLUT - The Opengl Utility Toolkit'. *Opengl.org*. N.p., 2015. Web. 3 May 2015.

[9] - Libsdl.org,. 'Simple Directmedia Layer - Homepage'. N.p., 2015. Web. 3 May 2015.

[6] - Lonesock.net,. 'Lonesock.Net: SOIL'. N.p., 2015. Web. 4 Feb. 2015.

[1] - Scrummethodology.com,. 'Scrum Methodology & Agile Scrum Methodologies'. N.p., 2015. Web. 13 Nov. 2014.