SCHOOL OF COMPUTER SCIENCE

University Of Kwazulu Natal

T.E Tshezi	214543866	_____
M.O Mtshali	212534816	_____
N.B Mhlongo	213511643	_____
M Ncanana	212559763	_____

GROUP 4

UNDER THE SUPERVISION OF *Dr. D Moodley, Mr. Jude Adeleke.*

MAY 2016

An Interactive 3D World Using OpenGL
Advanced Programming (COMP315)

ACKNOWLEGMENTS

If words are considered as a symbol of approval and token of appreciation, then let the words play the heralding role expressing our gratitude.

The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. We are grateful to our Lecturer, Supervisor and Mentor ***Dr. Deshendran Moodley*** *for the guidance that helped us prepare for and deal with difficulties we have encountered throughout the whole life span of this project, Mentor and group demonstrator* ***Ms. Shabana*** *the senior Tutor* ***Mr. Jude Adeleke****, for his inspiration.*

We also thank our colleagues, all group members who have helped in successful completion of the project.

ABSTRACT

A group of 3rd year Computer Science students were given the task of designing and implementing a 3D interactive game in C++ using OpenGL.

The main objectives were to learn and have a much deeper understanding of computer graphics and animation, while doing so, also design and implement these skills into a first person shooter 3D game.

An object orientated approach was used. Multiple classes were used to implement the design. An iterative approach was also incorporated. Various aspects such as texture mapping, lighting, collision, sound, mouse movements, a very professional looking game menu with menu navigation using the mouse and Major extensions such as rain effects, Loading of OBJ files from Blender etc. have been researched via the internet, books, and other resources. YouTube video tutorials, programming forums websites have been the most helpful form of gathering knowledge and skills used in this project and also overcoming some of the most difficult problems we came across while working on the project.

The coding for the world environment was put into action before anything else was incorporated, this has been used as a basic platform as ideas continuously expanded and implemented in an iterative manner into the project. These elements were incorporated towards the final stages of development, as it allowed for easier debugging.

The project is designed to be based on a main character, whose main objective is to get to the other side of a complicated maze alive, within the maze are enemies acting as obstacles to the player. This theme was found to be popular very interesting. The game play focusses on a simple yet challenging experience of getting lost in a huge maze while defending yourself as you figure out which way to go.

As an outcome of the project we have gained and learned how to apply our skills and knowledge of C++ object orientated programming and the use this powerful programming language effectively.

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

Developing a 3D game in OpenGL is both challenging and rewarding. There are many resources and libraries available to create a very realistic and working high-end game engine.

Games are not only fun but promote learning, critical thinking and interactive skills. Computer graphics allow us experience this in the most realistic way. A maze is a pattern form composed of walls, paths and dead ends, and the whole idea of a maze is to get lost a few times before figuring out the terrain and finally finding your way through.

Students in the COMP315 class are tasked to work in groups to design and implement a 3D game in C++ using the Open Graphics Library.

Using an agile iterative approach, the game has been designed and developed to completion. The main objective being to create a game with an interactive environment using suitable data structures, where the player is able to achieve the game objectives, where the player's basic movements are aided with primary controls.

Group 4 decided to call the game The Maze Runner, a first person shooter, single player game where the player starts on one end of the maze with a 100% life span and tries to reach the opposite end before he dies. The player has a gun as ammunition and is tasked to avoid or shoot to kill monsters that he will come across in his attempt to get to the other end, as the player's life decreases with contact of the monsters and his starting position being reset every time he comes to contact with the monsters.

There are life forms that the player is bound to come across as he traverses the maize which increases his life to a 100% and his chances of getting out of the maize alive. There also exists non-player of which the player can interact with outside of the maize. The game requires the player to be decisive throughout the game because if the player is to make it out alive the best if his cognitive abilities are required.

We decided to bring the action to the maize because we wanted to bring the best of both worlds in one game and still make the game noetic.

When we were given this project we were very excited to go ahead and begin the project but we were clueless of the challenges that came with its development. The ideas we had before the beginning the development of the project were very great, and we thought we would be able to implement them very easily but that was not the case, even getting the basics functions right was not as easy as we thought it would be. Even though we were not able to implement all the ideas we successfully managed to produce a decent, up to standard working game engine.

One of the most challenging obstructions we faced was that of implementing the idea of a complicated **maze** and how it would be related to the objects around it and of course figuring a quick, efficient way of implementing a **collision detection** mechanism that would work swiftly and reliably with each and every single wall in the maze, a complicated task which led us into hard coding the mechanism for each wall, by knowing the respective explicit coordinates of each wall and each object in the world from all possible directions except coming from above. **The most challenging** was the loading of **obj Models** from blender and moving the gun concurrently with the motion of the camera lookAt, and camera x and z.

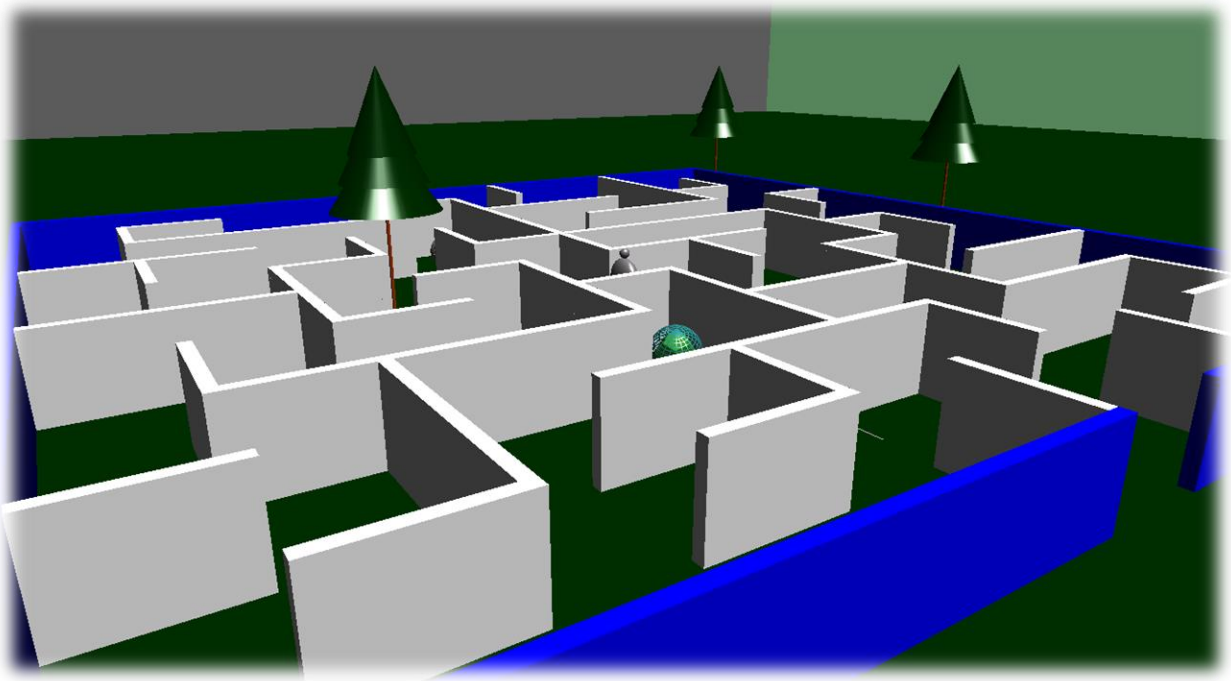

Figure 1

Figure 1 shows a resemblance of a real maze at a very **early stage** in the project, we were able to implement a very basic, small maze with the use of Glut Primitives and making use of their transformations to meet the required dimensions and orientation of the walls.

The world is contained inside a **box (cube),** surrounded by big walls. The floor was used as the basis of starting with the project. With the help of basic code from the **Snow_World** practicals along with keyboard movements.

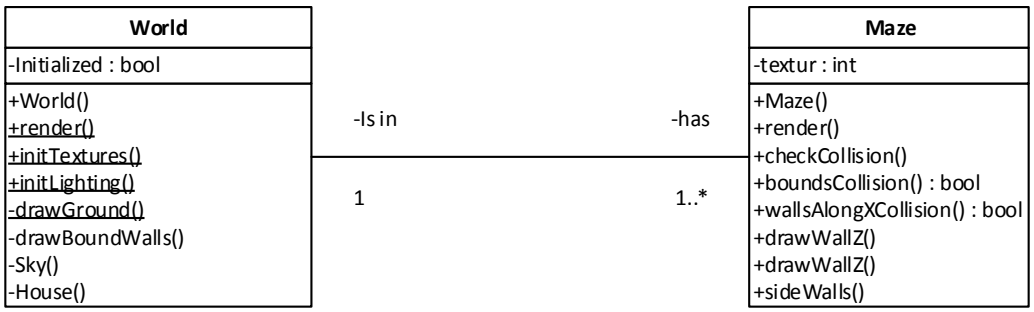

Figure 2

Figure 2 demonstrates how the world is relates to the maze, of course at this stage the maze collision functions had not been given any implementation but had only been just defined.

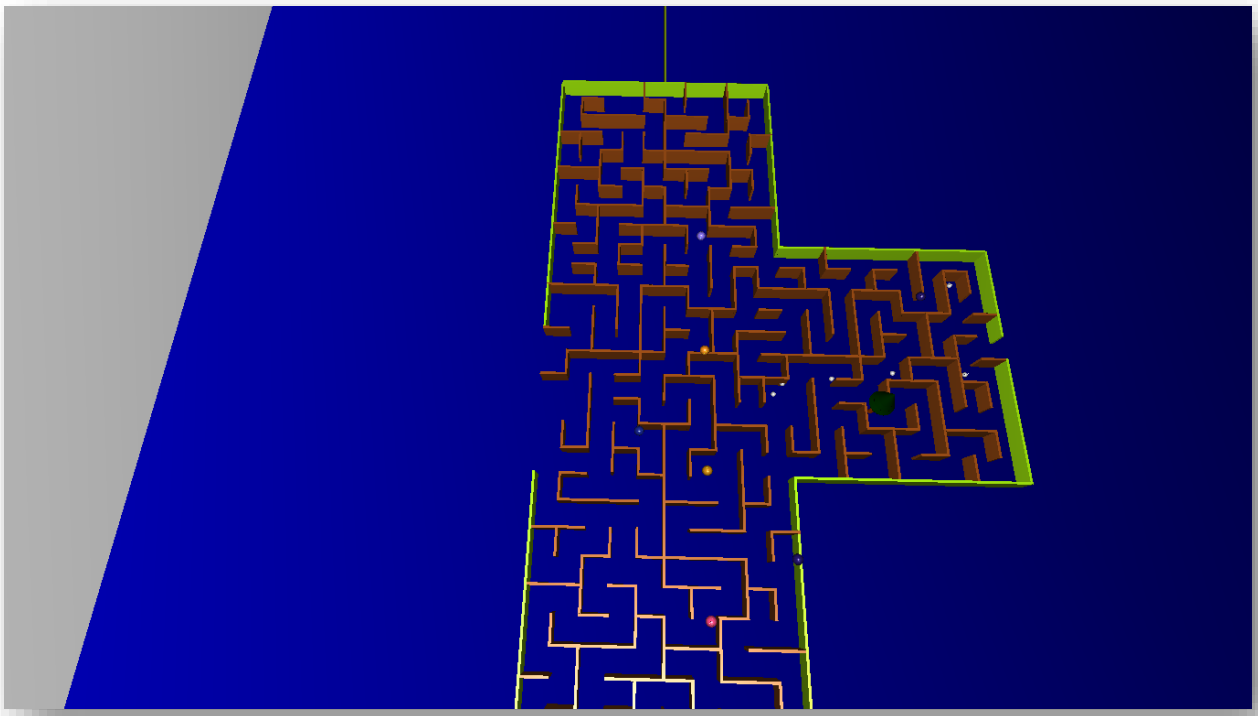

Figure 3

Figure 3 shows you the top view of the **new** modified maze, this was done by the duplication of the first maze, and so we made a **tradeoff** for this as the final. As it is big enough and has significant amount of complexity.

OBJ Loading, after a thorough researching and a lot of YouTube video tutorials on **How to Write an OBJ Loader** for OpenGL, we managed to write an obj loader, it's not perfect but it works. It does not show you the models in the world until you run the application from the **.exe** executable file in the location: bin\Debug .It **does not** export the wavefront with its **Texture coordinates** and **UV Lighting.**

Figure 4

The very first OBJ to be loaded into the project, at this stage it was **stationary** and couldn't move around along with the camera. We then overcame this challenge with the use of OpenGL default orthographic projection and simple trigonometric principles similar to those we used for mouse motions.

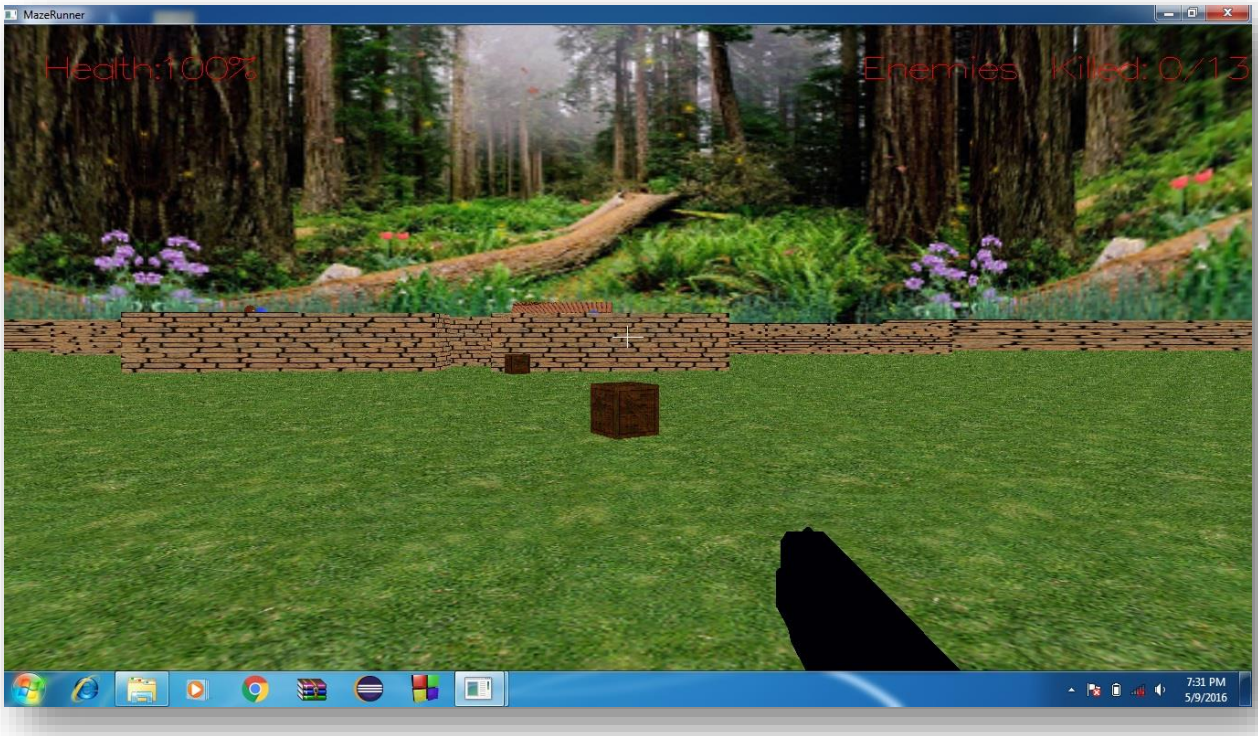

Figure 5

Figure 5, Gun motion after overcoming the problem at figure 4.

Another challenged we face was that of finding the right **audio API** that would allow us to play more than 2 sounds concurrently during game play, we found this to be very difficult as we had to manually and explicitly implement our own code and also decoding ourselves while using some these Libraries that support sound mixing. Some of the API's and external Libraries we have attempted to use include

- **OpenAL**
- **SDL**
- **PortAudio**

However we did end up using the **Playsound** function from the windows API. Had we had more time, most probably we would have managed to pull through.

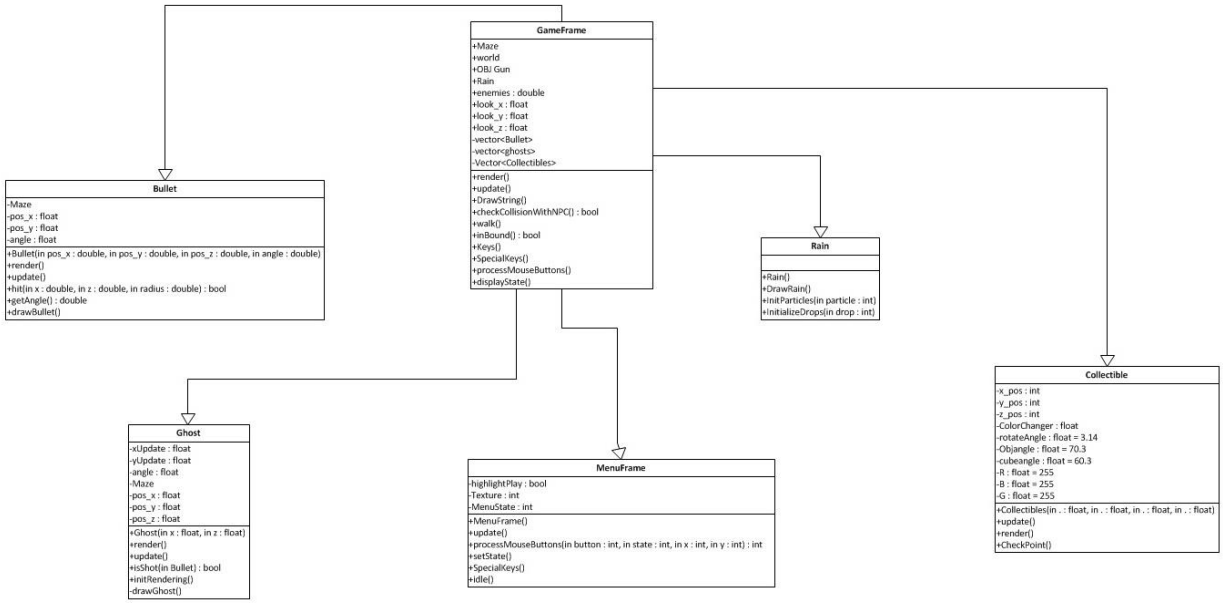

Figure 6

The Gameframe class basically does almost all the main functions of the gameplay, all the rendering of the world environment, The MenuFrame is only presented at the beginning of execution of the project. The GameFrame **uses** all classes in the project.

The implementation of **the maze** is solely based on simple Glut primitives more precisely **glutSolidCube()** and carefully calculated precise dimensions of the walls. Then playing around with **Transformations**.

```
        glTranslated(-280, 7.5, -180);
        glScalef(1, 10, 100);
        glutSolidCube(2);
    glPopMatrix();

    glPushMatrix();
     glTranslated(100, 7.5, 10);
        glScalef(1, 10, 90);
        glutSolidCube(2);
    glPopMatrix();

    //+x wall RIGHT MOST wall
    glPushMatrix();
        glTranslated(+280, 7.5, -180);
        glScalef(1, 10, 100);
        glutSolidCube(2);
    glPopMatrix();

    glPushMatrix();
        glTranslated(-100, 7.5, 10);
        glScalef(1, 10, 90);
        glutSolidCube(2);
    glPopMatrix();

    //-z +x wall
  glPushMatrix();
        glTranslated(-189, 7.5, -80);
        glScalef(90, 10, 1);
        glutSolidCube(2);
```

Figure 7

The UML design shows the various classes that the project contains. We found that separating most the game into these classes will ensure efficiency, and an even distribution of the work. We decided to design all components that have their own movement, in different classes. Although some classes are dependent on others, the use of our design made it easier to call methods from other classes, this was done by inheritance.

We made use of most of the OpenGL components such as

- The glut predefined methods,
- glut predefined methods,
- gl methods such as translates, rotates, scales, etc.
- For textures, bitmap images were used.

Problems encountered while programming:

- The Texture mapping of obj's from blender.
- Texturing sphere objects
- Collision detection of walls, there some complications between the walls at times.
- Lagging of the game, the more amount of rendering and more extensions added, the **slower** the game.
- Flickering of components, textured walls, Lighting, Positioning the light in a good position.
- Figuring the build target of the compiler, and keeping image paths consistent from one computer to another. (Absolute paths, relative paths, **image not found**)

The **rain** effect was implemented and **modified** from an existing **partial** code from a YouTube video tutorial. The idea is very simple, make particles fall down, when they get to the ground then reset them to start again at the starting position, (**loop**). Particles of the rain (rain drops) are drawn using **GL_LINES .**

```
void Rain::drawRain() {
    float x, y, z;
    glPushMatrix();
    glScaled(2,7,2);
    for (loop = 0; loop < MAX_PARTICLES; loop=loop+2) {
        if (par_sys[loop].alive == true) {
            x = par_sys[loop].xpos;
            y = par_sys[loop].ypos;
            z = par_sys[loop].zpos + zoom;

            // Draw particles
            glColor3f(255, 255, 255);
            glBegin(GL_LINES);
                glVertex3f(x, y, z);
                glVertex3f(x, (y+0.3), z);
            glEnd();

            // Update values
            //Move
            par_sys[loop].ypos += par_sys[loop].vel / (slowdown*100);//Speed
            par_sys[loop].vel += par_sys[loop].gravity;
            // Decay
            par_sys[loop].life -= par_sys[loop].fade;

            if (par_sys[loop].ypos <= -10) {
                par_sys[loop].life = -1.0;
            }
```

Figure 8

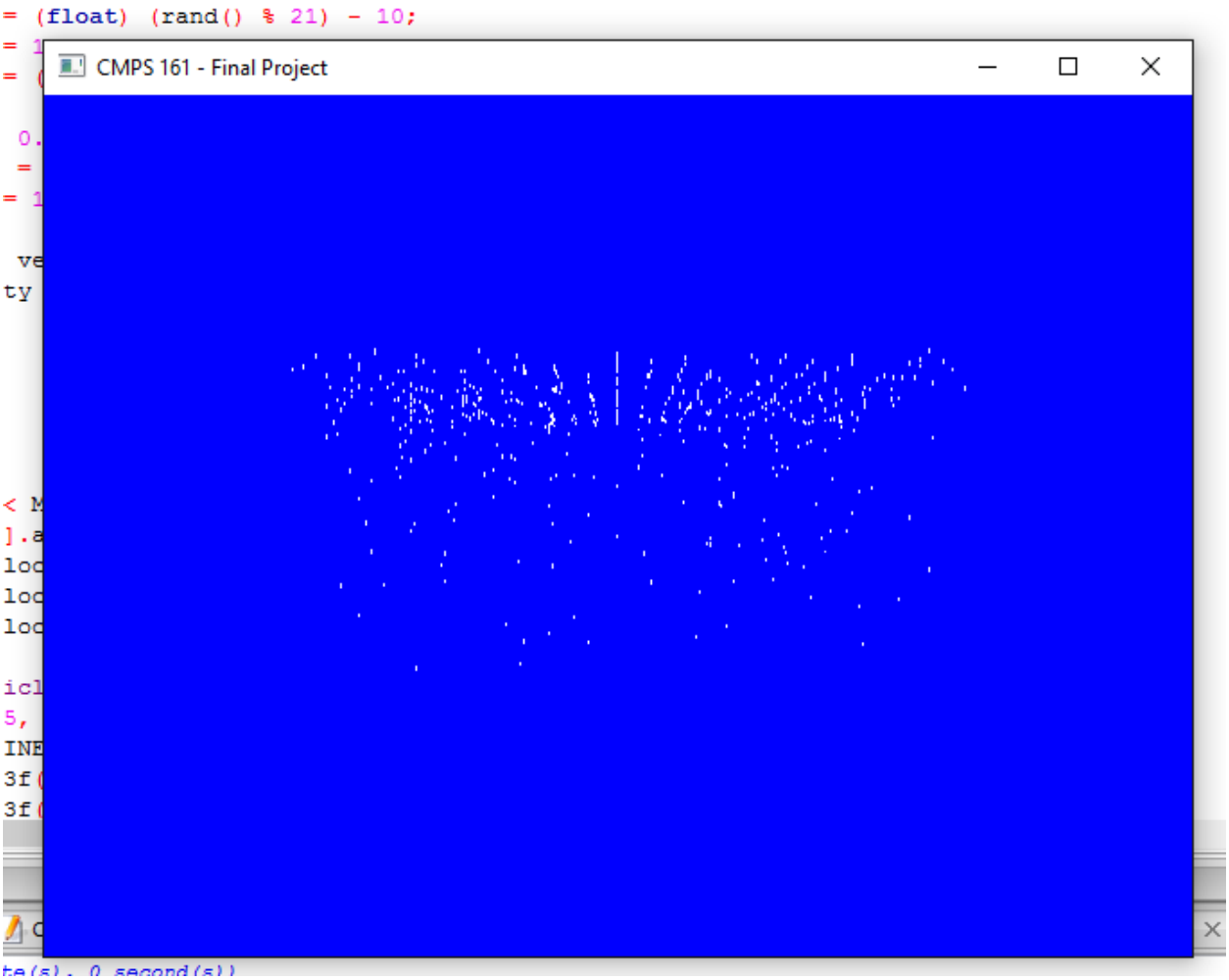

Figure 9

The rain is meant to rain only on a fixed **square area**, at first this was implemented separately on its own CodeBlocks project, then it later was incorporated into the game, we then duplicate the rendering of this animation around our world, using glut transformations.

The **OBJ loader** class is totally somebody's else's effort, idea, style of coding, We take no credit or what so ever , All credit goes to the author of the class, whom we will mentioned in this report. We watched a video tutorial, understood **how it works** with obj coordinates and how it actually meshes and renders these objects, and **How to actually use it in our code.**

Figure 10

The basic idea behind this realistic gaming menu, is switching to **Default Orthographic Projection,** and then using **2D Screen Coordinates** to detect the position of the cursor on the screen, there are 2 bitmap images involved, one highlighted on **play,** the other highlighted on **Exit,** when you move the mouse or press up and down keys, this creates **an illusion** to your eyes, that you actually navigating through the menu.

In our project we have multiple classes. Almost all objects have their own private space, their own classes, structuring it this way makes it very easy for anyone to understand the architecture of our design and also to understand the code. Separating the elements made debugging the classes much easier. The structure will allow future programmers top easily add in functionality as all they need to do is make a class with the new elements and then implement them in the GameFrame. Modification of code is therefore very simple and easy.

Our design has a lot of room for improvement and many more extensions can be added to improve the look and feel of the game. The design is open to the extension of a multiplayer game. This could be incorporated through networking. This could be done by having many player players playing against each other **in the maze**.

The most important part of achieving our design and meeting our requirements was working as a group. We all did a fair share of work and contributed immensely with ideas. During the software integration process we made use of Github and as a team member made modifications or updated code, we alerted the rest of the group using a group discussion on a social media. The use of these tools made it easier to modify code. Although we ended up with a lot of versions of our game, we found this way easier to use so that we could reference other versions and also have them more accessible if we had lost code or if our computers has crashed. Working as a group has had many advantages for different members had different strengths. Some were better in the functionality of the design, while others were better equipped for the rendering and design of the environment. This made it easy and more efficient for we all knew what we had to do. Software development was therefore effortless in terms of working in a group.

Lessons we have leant we have learned:

- How to work well in a group
- How to meet deadlines and work under pressure
- How to share code and accomplish tasks as a group
- How to solve problems and not have everyone working one that problem but maybe just two members.

The most important of all, how to code effectively, how to use what we have learned during our studies to code a fully functional game. How to code in a new language that being C++, How to resolve any conflicts between group members.

In the future:

- We will get started on whatever projects we may have much earlier
- Try and delegate work better and perhaps have two members working on an aspect rather than one
- Make use of better programming tools to have a better look and feel of our design
- Make use of better and more efficient coding techniques
- Use a better way of sharing code for modification
- Ensure that we have a clear and precise plan of action document, so that we know where we are and we are going
- Have a clear understanding of all objectives and aim of a project
- The development of this game has allowed the group to grow as programmers and work as a group.

CONCLUSION

In conclusion, developing this game has been a challenging yet informative and a good learning experience. Initially we had set out to implement many huge great ideas and functions into the game and we have successfully incorporated most of these into our final project. We have successfully designed and developed a 3-D Maze first person shooter game as described and initially planned.

We did experience difficulties, when trying to implement what we had initially set for ourselves in the beginning, for example implementing the Enemy movements around the maze in such a way that they will **come to YOU** (as the player) no matter which position you went to, they should **follow you**. We found this to be difficult, we had also planned to include **bleeding effects,** Although we think could have been easy if only we had just **one more week extra.**

However these did not affect the overall objective that we had hoped the game would accomplish and having changed a few of these extensions, we have developed the game we had intended to.

[1] Mukundan, R. (2012). *Advanced Methods in computer graphics: With examples in opengl*. Springer.

[2] Howard, T. (2010). *An introduction to graphics programming with opengl: Tutorial and reference manual* (v3.3). Manchester: School of Computer Science.

[3] Gregory, J. (2014). *Game Engine Architecture (2nd edition).* London: CRC Press.

[4] Eberly, D.H. (2006). *3D game engine design: A practical approach to real-time computer graphics (2nd edition)*. San Fransisco: Morgan Kaufmann Publishers Inc.

[5] Stack Overflow. (2013). *opengl display shape and draw text*. [ONLINE] Available at: http://stackoverflow.com/questions/15694233/. [Accessed 02 March 15].

[6] Lighthouse3d.com. 2015. *Bitmap Fonts >> Lighthouse3d.com*. [ONLINE] Available at: http://www.lighthouse3d.com/tutorials/glut-tutorial/bitmap-fonts/. [Accessed 02 March 15].

[7] Lighthouse3d.com. 2015. *Bitmap Fonts and Orthogonal Projections >> Lighthouse3d.com*. [ONLINE] Available at: http://www.lighthouse3d.com/tutorials/glut-tutorial/bitmap-fonts-and-orthogonal-projections/. [Accessed 02 March 15].

[8] Lighthouse3d.com. 2015. *Mouse: Moving the Camera |||*. [ONLINE] Available at: http://www.lighthouse3d.com/tutorials/glut-tutorial/mouse-putting-it-all-together/. [Accessed 26 March 15].

[9] Lighthouse3d.com. 2015. *Keyboard >> Lighthouse3d.com*. [ONLINE] Available at: http://www.lighthouse3d.com/tutorials/glut-tutorial/keyboard/. [Accessed 26 March 15].

[10] Lighthouse3d.com. 2015. *Sub Menus >> Lighthouse3d.com*. [ONLINE] Available at: http://www.lighthouse3d.com/tutorials/glut-tutorial/sub-menus/. [Accessed 26 March 15].

[11] Dimitrios Christopoulos . 2014. *NeHe Productions*. [ONLINE] Available at: http://nehe.gamedev.net/tutorial/collision_detection/17005.htm. [Accessed 01 April 15].

[12] Lighthouse3d.com. 2015. *Frames per Second >> Lighthouse3d.com*. [ONLINE] Available at: http://www.lighthouse3d.com/tutorials/glut-tutorial/frames-per-second/. [Accessed 11 April 15].

[13] Programming Resource. 2002. *3D Sound Tutorial*. [ONLINE] Available at: http://www.alsprogrammingresource.com/sound_tutorial.html. [Accessed 03 May 15].

[14] Fallout Software. 2015. *OpenGL Light Tutorial*. [ONLINE] Available at: http://www.falloutsoftware.com/tutorials/gl/gl8.htm. [Accessed 06 May 15].

[15] Tom Dalling. 2013. *Modern OpenGL 07 – More Lighting: Ambient, Specular, Attenuation, Gamma*. [ONLINE] Available at: http://www.tomdalling.com/blog/modern-opengl/07-more-lighting-ambient-specular-attenuation-gamma/. [Accessed 11 May 15].