EECE 478
Game Project – Crystal Maze
Final Report

Instructor: **Dr. Lee Iverson**

Team Members:	**Student Number**
Abdel Hamid Ismail Ahmed	83551044
Abdullah Alqattan	14248033
Frankie Shum	88415047
Joaquin Valdez	82370040
Seung Hyun Park	85563047

ABSTRACT

This report describes the design and implementation of a 3D game called Crystal Maze. The game is created using OpenGL where the player works his/her way through a maze consisting of nine rooms. Each room introduces certain tasks the player must perform in order to progress through the game. The objective of the game is to navigate through the maze and perform the tasks required to obtain four crystals and win the game.

Table of Content

List of Figures

1.0 INTRODUCTION

Creating a 3D game in OpenGL is both challenging and rewarding. There are many resources and libraries available to create a very realistic and high-end video game. We were able to make use of these resources to create a 3D maze game called the Crystal Maze. This game requires navigating through a 3D maze as well as solving riddles and puzzles to finish the game. By implementing advanced features, the overall game play quality went up as well as the look and feel of the game. We learned all the basics to designing and implementing a 3D game in OpenGL as well as some of the more advanced techniques used when creating such a game. Much of our time was spent on adding features to the game as well as improving the storyline of the game. We were able to create an interesting game that was a mix of 3D maze game and other role playing games.

2.0 GAME OVERVIEW

The game idea chosen for our project is a 3D maze. The first reason behind this decision is that in a 3d maze we can explore any area we would like to develop models for. We can construct furniture, animals, space objects, wizards, different themes in different rooms. The idea seemed very practical for this project, and left a large room for creativity. The second reason we all unanimously accepted this idea is that we were all apparently affected by games of this style such as Final Fantasy, Resident Evil etc... The idea behind the game is to make the maze full of puzzles and mysterious themes.

2.1 Game Concept and Goals

The main objective of the game is to collect all four crystals by performing the steps and hints given in each room. The player may have to visit a room more than once as the steps must be performed in sequence for the player to obtain the crystals. There are a total of nine rooms where each room represents a new challenge for the player. Each room has its own theme to match the action to be performed. Hints are provided by writings on walls as well as player interaction with the models in the game.

2.2 User Controls

Game is very hard to play if it is difficult to control the player; we implemented easy and intuitive user controls to help ease the learning curve of the game so that anyone can pickup on the game quickly. By using the mouse to look as well as point toward the direction the player will move toward, it makes it very easy for the user the control their player. On top of the basic movement controls, the player is capable of performing several tasks throughout the game. The actions that can be performed include; talk, pickup, open and use. Following is the summary of actions allowed in the game:

Mouse Left-Click – Look

Mouse Right-Click – Use from inventory

Arrow Key Up – Move forward

Arrow Key Down – Move backward

Arrow Key Left – Strafe left

Arrow Key Right – Strafe right

‘t’ – Talk to model

‘p’ – Pickup object

‘o’ – Open object

‘r’ – to ring the bell

<u>2.3 Scene layout</u>

The first thing we did is that we tried to draw up a scene plan. This took much iteration before we reached the one we have now. However, it was important to have such blueprints of the maze, because it made it easier in communication between groups. When it came to the translation of objects, and the development of the restrictions this plan proved very useful. The diagram below shows the layout. The color codes represent different aspects of the code. The side axes represent the x-axis and the z-axis in the game.

The color coding can be described as follows:

Grey: Restricted area

Yellow: The rooms

Purple: Horizontal corridors

Sky blue: Vertical corridors

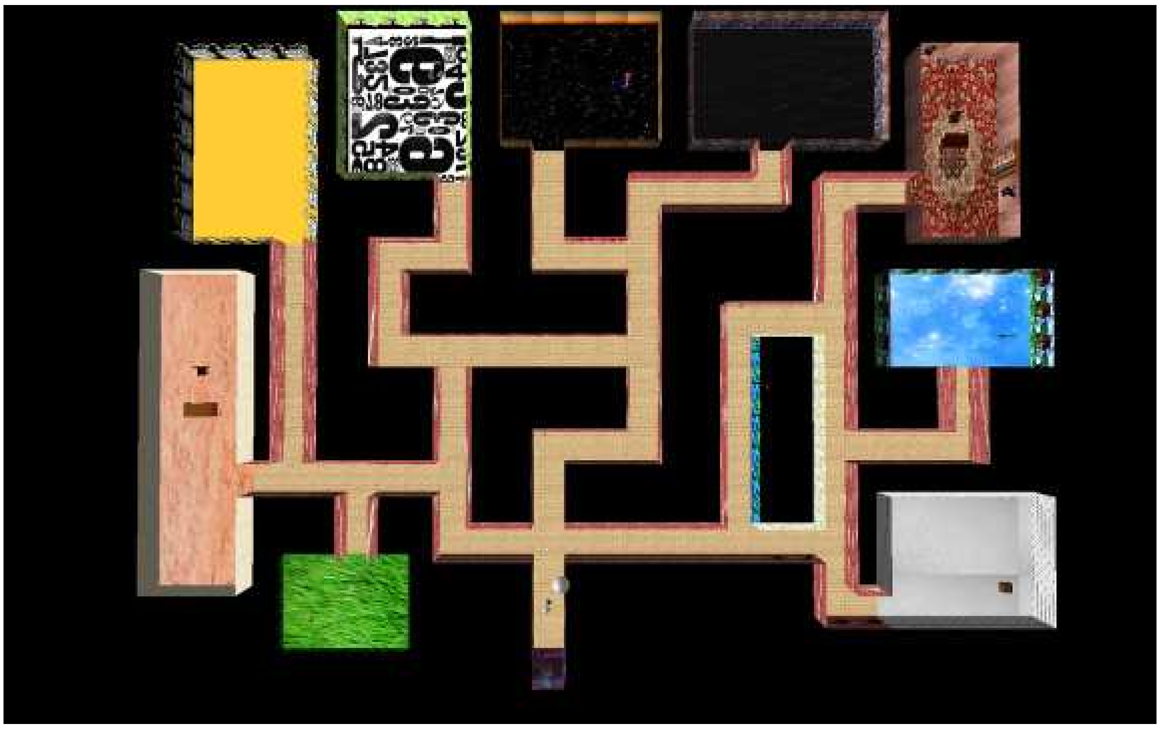

Figure 2.3.1 World Plane Layout

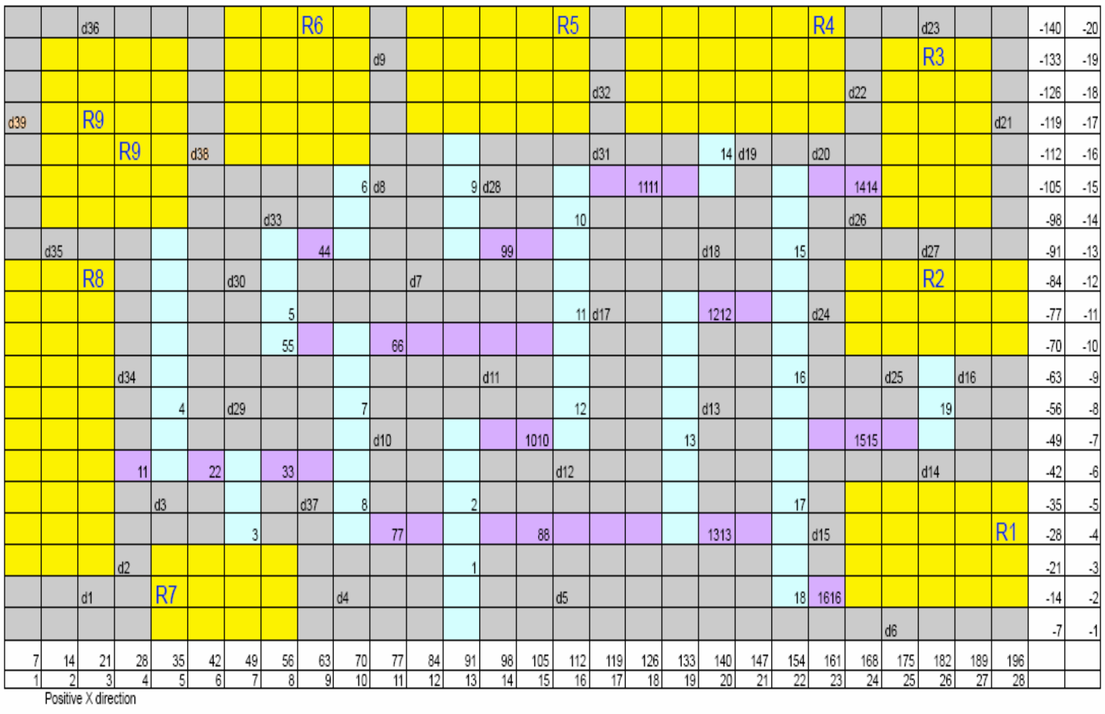

Figure 2.3.2 World Plane Coordinates

3.0 SOFTWARE ARCHITECTURE

The main objective of our architecture was to ensure scalability and maintainability. This was done by creating a world plane with easily identifiable coordinates that were both modifiable and scalable. Changes to objects and functions do not depend on other objects and functions and are defined according to the world plane.

Following is the main structure of our software architecture:

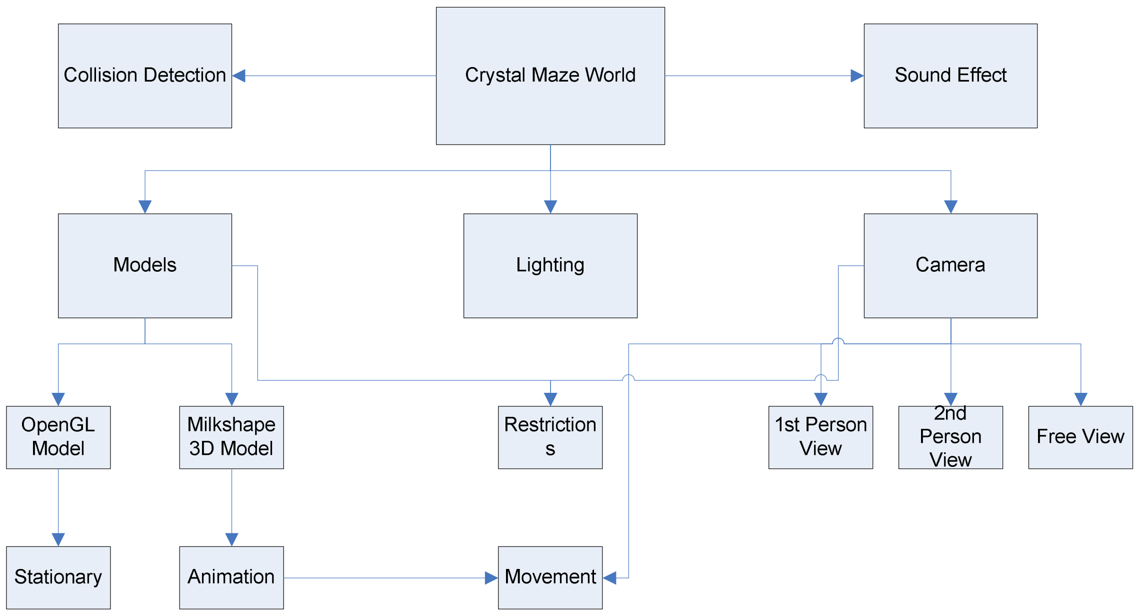

Figure 3.0.1 Architecture Block Diagram

Interactions between each object and function are as follows:

Crystal Maze World – This is the main world plane which interacts with Collision, Sound, Lighting, Camera and Models.

Models – Models are split into two types: OpenGL Model and Milkshape Model. OpenGL Models were created using the functionality provided in the Glut library. Some of the models were used in animation such as the player model and the worm model. Other models were stationary models. The Milkshape Model were created or imported through Milkshape3D and they were free models found in [1].

Camera – There are three camera model views available to the user: 1st Person, 2nd Person and Free Camera. The camera also interacts with the movement function in which player movement was followed by the camera depending on the current view of the camera.

Restrictions – This object created boundaries in which the models were enclosed within. Camera also conformed to the restrictions provided by the restrictions object to ensure that neither the player nor the camera is able to go through objects or walls.

Movement – This object controlled the movement of the player with interaction with the animation and the camera functions.

Other objects not mentioned above are the Milkshape3D, Model, Texture, OpenAL objects. These objects were used to supplement the functionality required by their corresponding objects.

Our software architecture ensures that each major object or function does not depend on other objects. These objects can be used in several instances without any conflicts and can be used in different projects as well. This approach enables our game to be easily modified. Model and objects can be imported very easily by a simple call to the function. Sound can be added or changed by another simple function that calls sound. Restrictions can be modified depending on the requirements of the world plane. These are just some of the easily modifiable and reusable code in our program.

4.0 MAIN FEATURES

4.1 Basic Features

The most basic features of any 3D computer game are the dependence on the three-dimensional structure of the environment. The 3D aspect of the game has direct consequence on the game play. Our game follows through with this idea and includes the following basic features; 3D Viewing and Moving Camera, 3D Models, Lighting, and Texture Mapping.

3D Viewing and Moving Camera

The player can play in two different views: 1st person and 2nd person view. We also have a third camera view that is restricted to the developers of the game and players can unlock this view with a secret character. The third view is a free moving camera. The player can toggle between 1st and 2nd person using the ‘7’ key.

Free Moving camera:

The free moving camera view was the first view to be implemented. We wanted a free camera in order to move through the game as we develop it and inspect objects from all angles, so we first made this view.

The free moving camera view was achieved using the gluLookAt function. We control the position of the camera using the gluLookAt, and the camera is always looking infront of its position.

The free moving camera view consists of six keys and mouse rotations of the view. The four arrows are used to move to the four directions and the numbers 9 and 3 were used to move the camera upward or downward. Left clicking the mouse and moving it will either rotate the view around the y axis or the x axis according to which direction the mouse is moving. Therefore, the player can freely use the camera to look at any direction and can move toward any direction.

The camera rotations were achieved using the mouse callback to detect the dragging of the mouse event, and then two angles where changed according to how much and in which direction the user moved the mouse. These two angles will rotate the view so that the user can look at any direction. The rotations were implemented by translating the camera to the origin and rotating the world using these two angles and then moving the camera back to its original location. The upward and downward movements of the camera were achieved simply by increasing or decreasing the y parameter in the gluLookAt function. The other four remaining directions are, however, not constant like the upward and downward movements since the player can rotate the view and look at any direction. For example, the forward direction will change according to which side the user is looking at. To implement these four directions movement, we have came up with a function for each direction. The functions are called moveForward, moveBackward, moveRight, moveLeft. What happens inside the functions is that the two rotations angles are examined and then according to their values, the forward direction, for example, is calculated using proportions of the x and z axis. The calculations is very similar to blending two values, so, basically the two edge cases are computed and then any values in between are a blend of the two edge cases according to how close the angle to one of the edge cases. The following is an example:

```
                if(angle <= 90 && angle >= 0){
                        float i = angle / 90.0;
                        params.move_camera_z-= cameraSpeed - (cameraSpeed
* i);

                        params.move_camera_x+= cameraSpeed * i;

                }
```

Here, we are first checking if the angle is between 0 and 9, if it is, then the forward direction of the player isb between the +z direction and the +x direction. Therfore, we calculate how close the player is to each directions and increasing respectively to obtain a movement in the exact directioin the player is looking it.

The result proved accurate, and the camera would move exactly in the forward direction upon rotating the camera to any direction. Figure 4.1.1 below shows the free camera view looking at different directions.

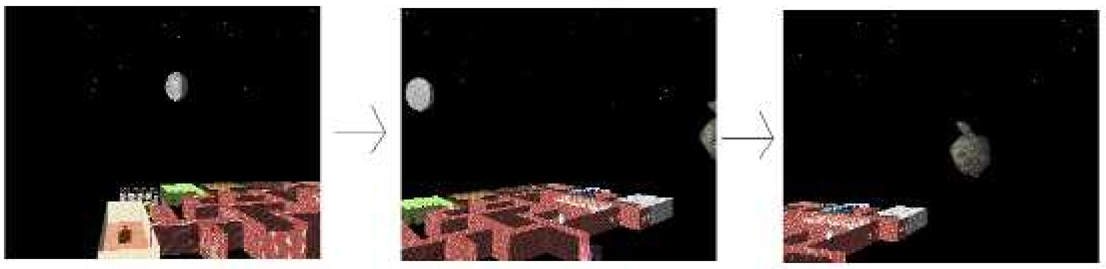

Figure 4.1.1 Free Camera View

First Person View:

The first person view is achieved by placing the camera at the eye of the player. The player can look at any direction just like a human and can move in four directions which are the directions of the arrow keys. To have the player look at any direction, a similar method to the free moving camera is used, rotating the whole world around, but this time the player is translated to the origin and the rotations are made around the player. For the movement of the player, a similar approach to the free moving camera is used. The forward directions, for example is calculated, and the player and the camera are moved in this direction. Figure 4.1.2 shows the first person view.

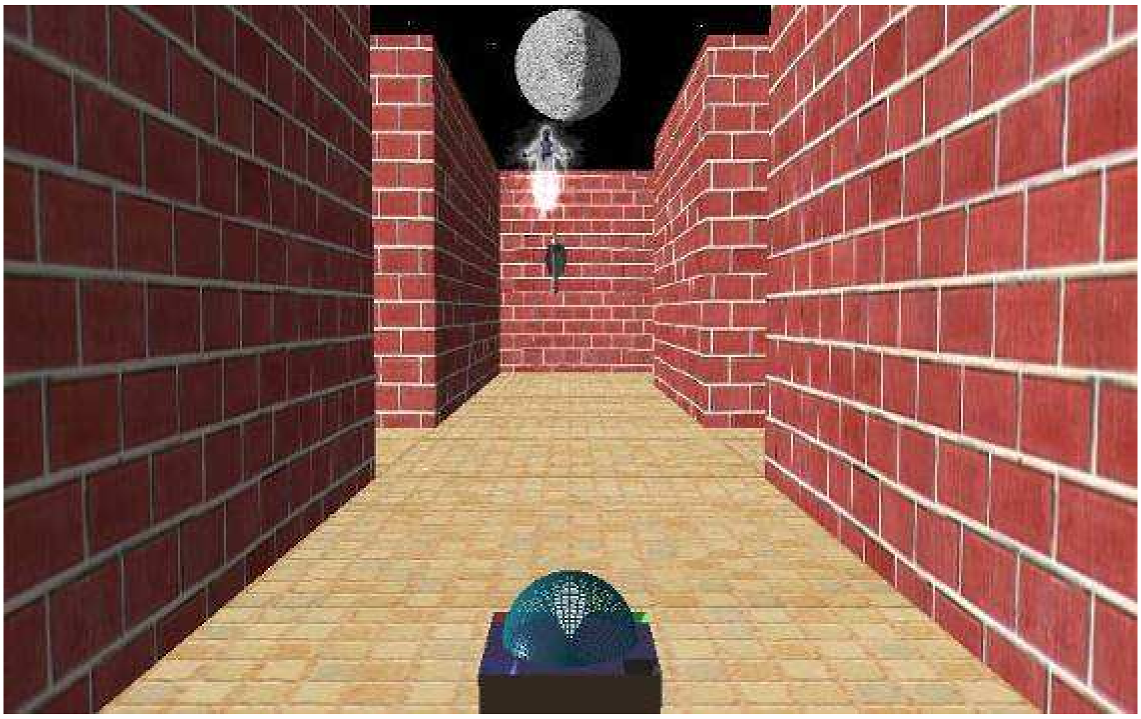

Figure 4.1.2 First Person View

Second Person View:

This view is achieved by taking the first person view and fixing the position of the camera behind the player in order to have the whole body of the player apparent on the screen. The movement and the view is controlled in a similar manner to the first person view, but the vertical rotations are restricted to eliminate rotations that would view the world from outside of the boundaries (See Figure 4.1.3).

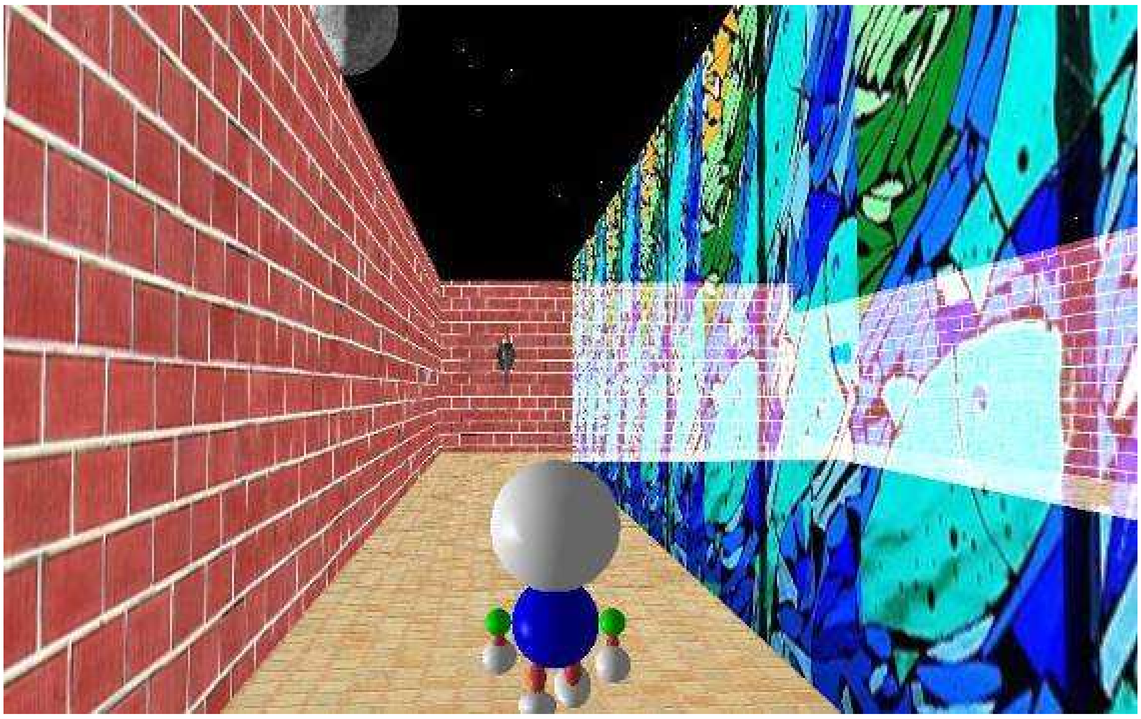

Figure 4.1.3 Second Person View

3D Models

3D models in our game were created using simple polygons, cylinders and spheres. Other more detailed models that could not be created using GLUT were imported from Milkshape3D models that were available on the internet. This section describes the 3D models that are available in our game.

The crystal stand and the crystals:

Since our game is built on collecting crystals and placing them on the crystal stand, we decided to place the crystal stand at the beginning with a big crystal in the middle of it that is covered with a wired sphere. Once the four crystals are collected from the game and placed into their places on the stand the wired cover of the big crystal will be removed and the player will obtain the big crystal and win

The crystal stand was made of several colored polygons. The cover for the big crystal is a wired sphere that is controlled by a state variable to detect the time to remove the cover. We wanted the places of the crystal to be obvious and look interesting, so we made an engraved area for each crystal on the top of the stand. These areas were made to fit the crystals exactly and colored with their colors. With the lighting effects, they appear as engraved positions for the crystals in the stand. The way these areas were made is by leaving four holes into the top of the stand and drawing in each hole four triangles to make a flipped over pyramid with a hollow body. The materials for these areas were made shiny (See Figure 4.1.4).

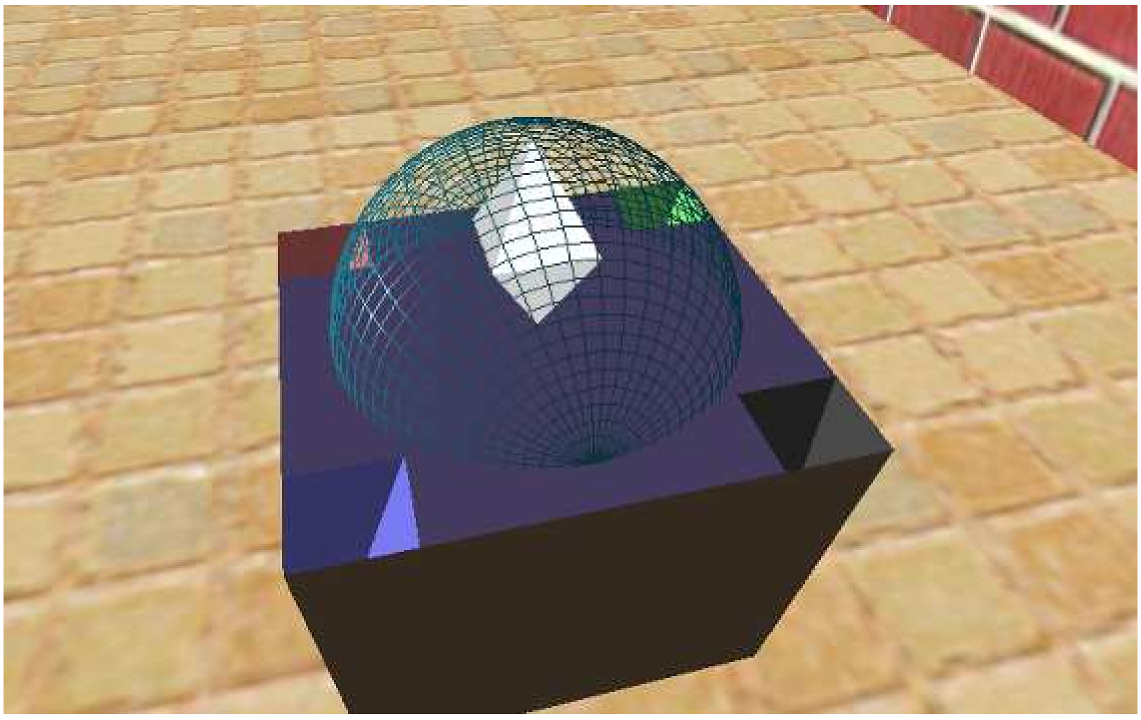
Figure 4.1.4 Crystal Stand (a)

The crystals each one of them is made of two pyramids that were constructed in GLUT using simple triangles. Then, colored so that one side would be more shiny than the rest to imitate a crystal look.

The four crystals were placed into different rooms. The position of the crystal is controlled each by a state variable. For example, if the state variable for one of the crystals is 0, the crystal is drawn in the hidden place where the player supposed to find it. When the player finds a crystal, we change the respective state variable to 1 to indicate that the crystal should disappear, and at this point we add an entry in the right click menu of the player with the name of the crystal to indicate that the player owns the crystal. Once the player chooses the crystal from the menu while the main character is at the crystal stand, we change the state variable and the crystal will be translated to its position on the stand (See Figure 4.1.5).

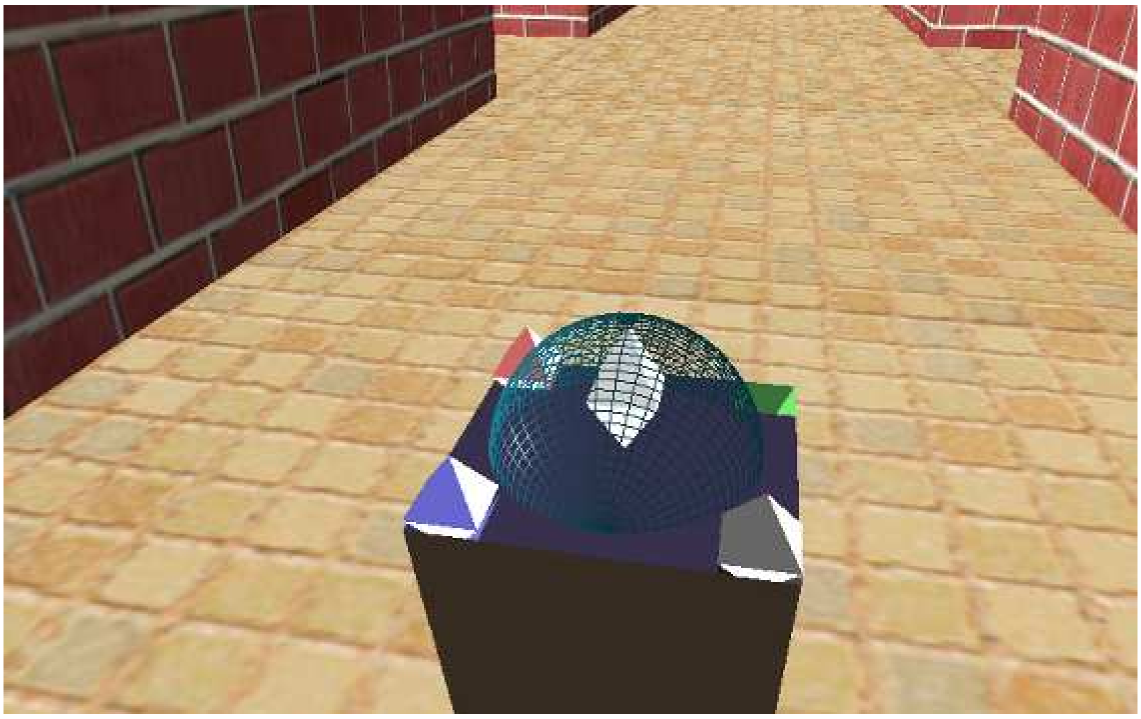
Figure 4.1.5 Crystal Stand (b)

Main character:

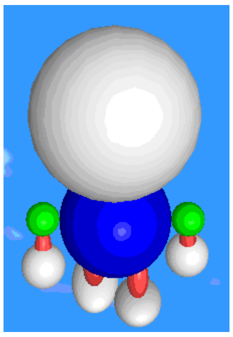
Figure 4.1.6 Main Character

The main character is composed of a head, a body, two arms, and two legs. Each part is modeled by using OpenGL. For example, the arm is composed of two spheres and an ellipsoid, and the leg is made of two ellipsoids. After the structures of all part are finished, they are joined together.

Worm:

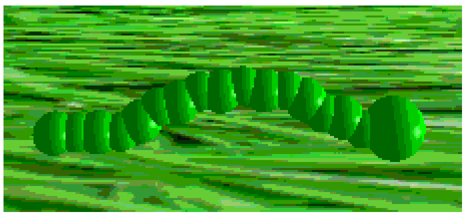

Figure 4.1.7 Worm

The worm is constructed of nine parts such as a head and eight identical division of the body. Each part is modeled by using OpenGL.

Book:

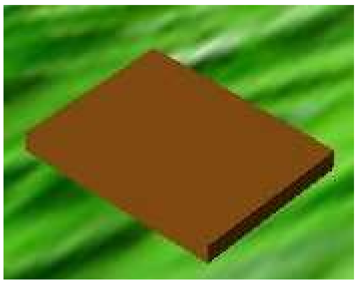

Figure 4.1.8 Book

The book is made of two tetrahedrons which are joined together. It is modeled by using OpenGL and is placed in room 7. In the beginning, this book can be opened and closed by the player, so the player can read the book. Although this feature is disabled for now, it can be enabled later if needed.

Safety box:

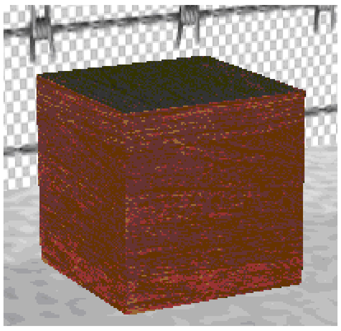

Figure 4.1.9 Safety Box

The safety box is a cube with a top which can be opened or closed. It is modeled by using OpenGL; it is place in room 1. If the player input a correct code into the box, then it will open, so that the player can see and take the element inside the box.

Door:

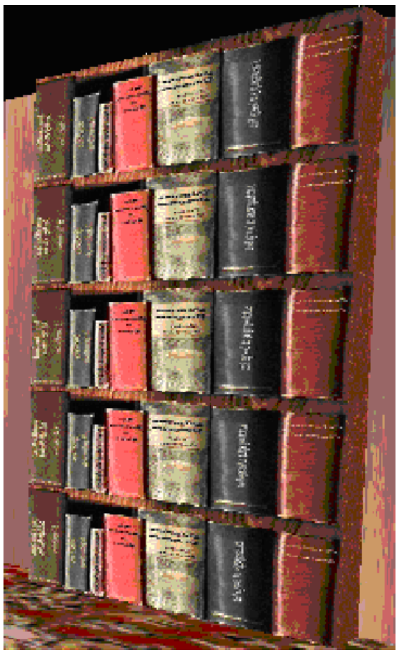

Figure 4.1.10 Hidden Door

The door is made of a tetrahedron which can be moved or rotated; it is placed in room 3.

It is modeled by using OpenGL

Key:

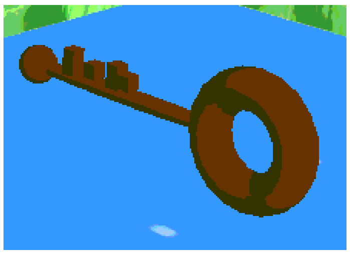

Figure 4.1.11 Key

The key is composed of two torus, four cubes and one sphere; it is modeled by using OpenGL.

Cylinder:

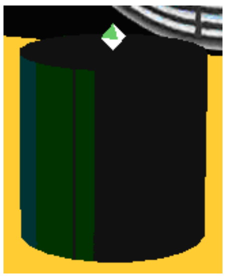
Figure 4.1.12 Cylinder

The cylinder is modeled by using Quadric() in OpenGL. It is composed of a cylinder and a disk. The cylinder is place in the music room; it will appear in some conditions.

Fishes:

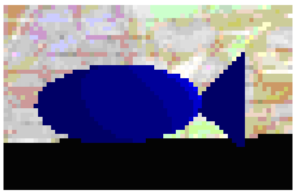
Figure 4.1.13 Fish

Each of the fishes is composed of an oval and a cone with certain scale. They are modeled by using OpenGL. They are placed in the aquarium.

Table with a Drawer:

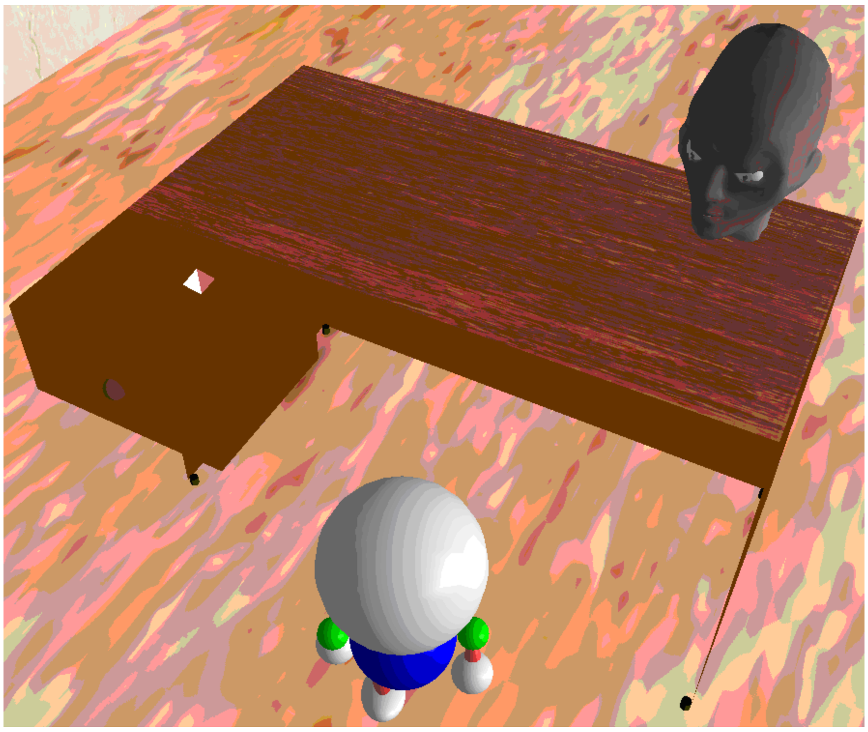

Figure 4.1.14 Table with Drawer

This table is made in GLUT using simple shapes (polygons, cylinders, and spheres). It was made to have a drawer that can be opened. A crystal has been placed in the drawer.

Milkshape3D Models:

Milkshape3D models were imported to our game to for better 3D graphics in our game. There was a limit in what GLUT can do to make complicated 3D objects and therefore, external source was required to make the game more complete. In order to import

Milkshape3D objects we used an open source code that made it simple to import any MS3D objects. The open source codes we used were; MilkshapeModel object and Model object. All our Milkshape3D models were available as open source on the internet and the specific reference can be found in our reference page. [5].

For our 3D models from Milkshape3D, we have the following; Wizard, Snail, Wooden Table, Wooden Chair, Spaceship, Earth and Eye. Texture were added to these objects and some objects, such as the Wizard interacts with the player by talking to the player.

Lighting

Lighting is achieved by placing one light at the left side of the world. The light used is a point source with full capacity white light. Also, a global ambient white color is used too. We wanted to have a dark view to go with the theme, and wanted the earth and moon to be lit from one side to create a shading effect. The shading model used is smooth shading. The moon and earth are half lit imitating the sun light in reality. Some of the materials property used are set to white color with the original color obtained by enabling glMaterialColor, other materials properties were set to have the object show colors produced by the ambient, diffuse, and specular components. Different objects have different materials colors some objects were made shiny such as the crystals and the wired sphere cover, and some objects are set to emit white light such as the candles on the walls and the ghost at the beginning.

Texture Mapping

We used texture mapping to create the complicated colours for the walls, floors and some of the objects. When it came to design the walls and the rooms we needed a quick way to write the code. So we developed a couple of for loops for this.

The first one creates the vertical walls. The second one creates horizontal walls. Then we just change the images to suite our need. We only need to enter three variables. The only thing with this design is that all the walls are 7x7, but they work.

The coordinates belong to the bottom one if it is the vertical wall and left side if it Horizontal side.

a) The x-coordinate.
b) The z-coordinate.
c) The number of times we need to draw 7x7 wall.

```
for(int i=0;i<1;i++){
 GLfloat start =56.0f;
 GLfloat wall   = -63.0f;
 glBindTexture(GL_TEXTURE_2D, texture[floor]);
 glBegin(GL_POLYGON);
  glTexCoord2f(0.0f,0.0f);glVertex3f(start+7.0f+(7.0f*i),0.0f,wall-7.0f);
  glTexCoord2f(1.0f,0.0f);glVertex3f(start+7.0f+(7.0f*i),0.0f,wall);
```

```
   glTexCoord2f(1.0f,1.0f);glVertex3f(start+(7.0f*i),0.0f,wall);
   glTexCoord2f(0.0f,1.0f);glVertex3f(start+(7.0f*i),0.0f,wall-7.0f);
  glEnd();
}

for(int i=0;i<8;i++){
  GLfloat start =28.0f;
  GLfloat wall = -35.0f;
  glBindTexture(GL_TEXTURE_2D, texture[floor]);
  glBegin(GL_POLYGON);
   glTexCoord2f(0.0f,0.0f); glVertex3f(start+7.0f,0.0f,wall-(7.0f*i)-7.0f);
   glTexCoord2f(1.0f,0.0f); glVertex3f(start+7.0f,0.0f,wall-(7.0f*i));
   glTexCoord2f(1.0f,1.0f); glVertex3f(start,0.0f,wall-(7.0f*i));
   glTexCoord2f(0.0f,1.0f); glVertex3f(start,0.0f,wall-(7.0f*i)-7.0f);
  glEnd();
}
```

In order to make it easier to change the texture for the rooms, and to be able to the change the texture for each side of the rooms, we made each room to have, N+1, parameters. Where N is the number of sides in the room and the extra one is for the floor. An example of this is:

```
void drawRoom_2(int side_1,int side_2,int side_3,int side_4,int side_5,int floor)
```

For the floor we just made it as a single polygon, again the texture ID was passed as a parameter.

<u>4.2 Advanced Features</u>

On top of the basic features that are available in our game, we added several advanced features to make the game look more realistic as well as improve the overall game play. We decided to add the following advanced features to our game; Transparent Objects, Animation, Collision Detection, Sound Effects, and Restrictions.

Transparent Objects

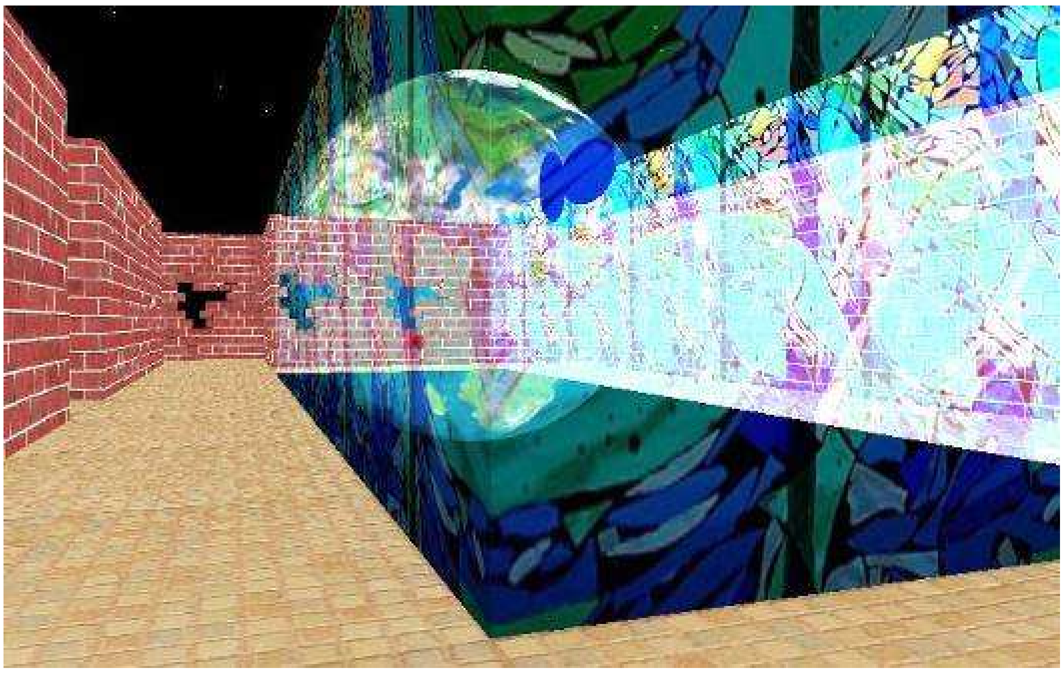

Figure 4.2.1 Fish Tank

To make any surface transparent, GL_BLEND must be enabled. Then glColor4f(1.0f, 1.0f, 1.0f, 1); has to be defined; where, the last parameter is the the alpha parameter that refelcts how trasparent the surface will be. Once the traparency has been drawn BLEND must be disabled.

The following transparency effects were achieved using TGA file format. The difference between TGA file format and bitmaps is that TGA files have a fourth, 8 bit value; used after the red, green and blue values that specifies the opacity of a pixel. These extra bits make opengl able to recognize alpha channels. TGA files can be easily edited and created using Adobe Photoshop 6+. Once the image has been created and the transparent fields, GL_BLEND and GL_ALPHA_TEST are enabled and the alpha factor that will not be displayed into the screen using the subroutine glAlphaFunc. The grass in the "grass room" was also made using the same technique.

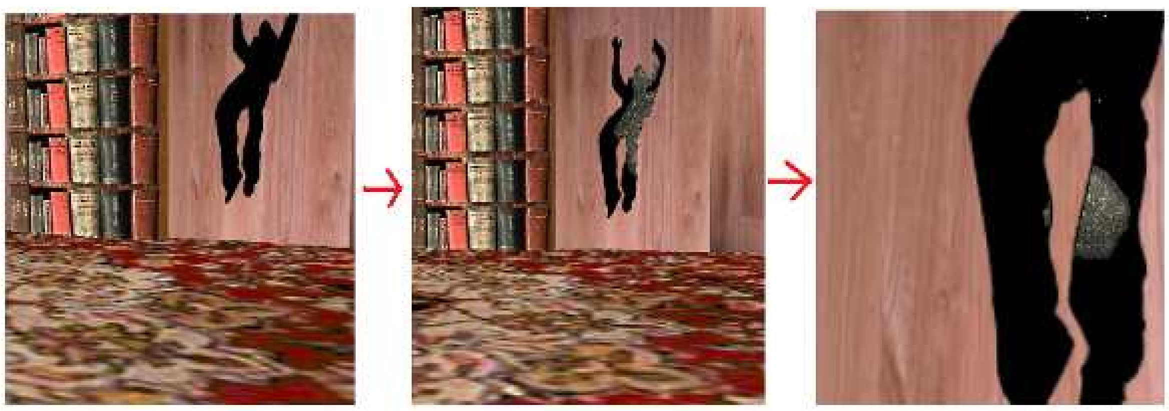

Figure 4.2.2 Transparency Example (a)

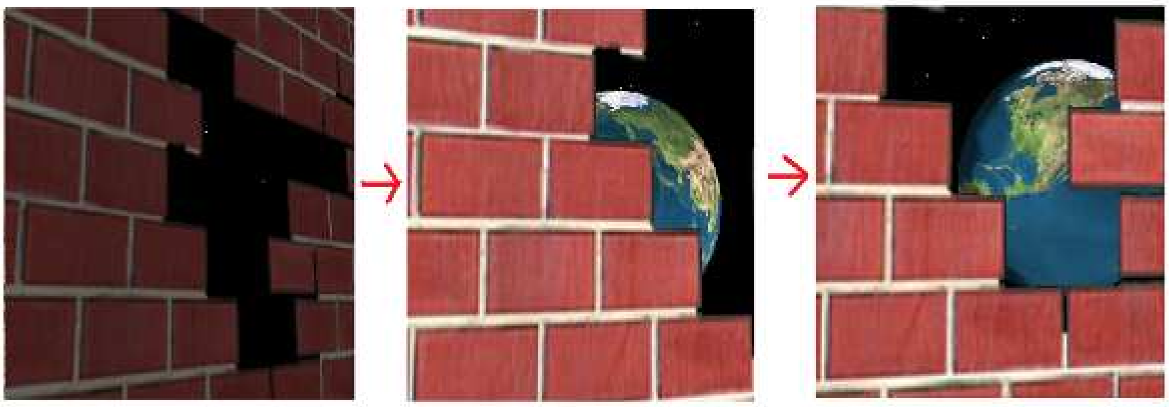

Figure 4.2.3 Transparency Example (b)

The above examples resumed in the following code.

```
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
glColor4f(1.0f,1.0f,1.0f,1.0f);
glAlphaFunc(GL_GREATER,0.1f);
```

We have used also transparency to produce text in the game. We loaded bitmap textures with text in them on polygons. The background of the bitmap is black. Then, we made the background of the bitmaps transparent to have only the text apparent. This has proved to be very effective and sufficient to include text writing in the game. Two examples are shown in the following figures of text achieved with this method.

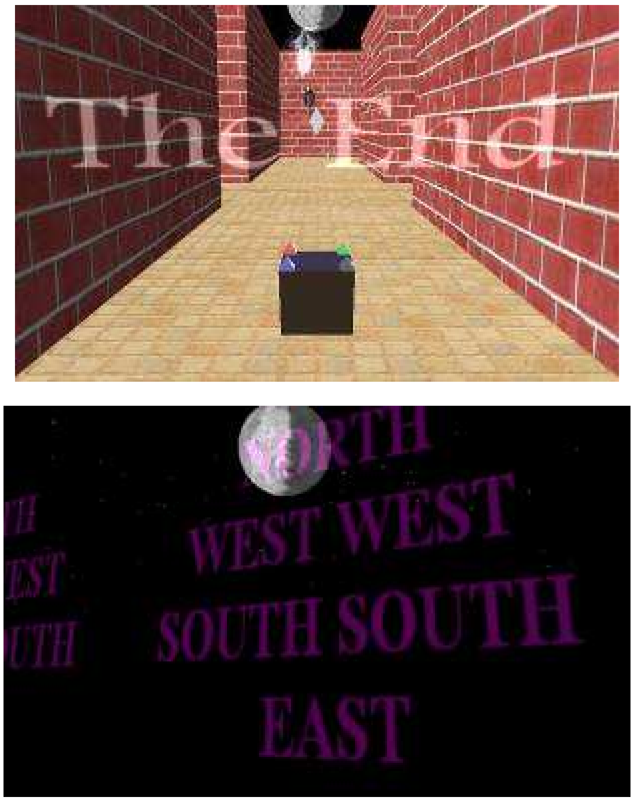

Figure 4.2.4 Transparency Example (c)

Text Achieved Using Transparency and Texture Mapping

Animation

Animation creates a more realistic interaction between the player and the game. By adding this feature, it made the game much more fluent then just translating the objects without any animation. Adding animation requires precise handling of vertices and their coordinates in our world plane.

Main character:

The head, arms and legs can rotate individually. While the character is moving forward/sideward, the function Walk() or Walk_side() is called, and the arms and legs will be rotating within a certain angle which is like an human walking motion. The character moves and his arms and legs move when the player presses the arrow keys.

Worm:

Each join can rotate depending on the adjacent part(s). Because of this structure, the worm can move as creeping. The worm is moving in a cycle direction; it moves one degree from the centre of the room whenever the DrawWorm() is called.

Book:

The book can be opened or closed by the player. This motion is made by using glRotatef().It will disappear after the player presses the ‘p’ key to pick it up.

Safety box:

If the player input a correct code into the box, then the top side will open. The open motion is like opening a laptop lid and this motion is made by using glRotatef().The safety box is open when the player presses the ‘o’ key with a correct code.

Door:

The door can be moved or rotated by the player. It is moved by using glTranslatef() and glRotatef().The door is moved when the player presses the ‘o’ key.

Key:

The key is composed of two torus, four cubes and one sphere. It will not physically rotate or translate to somewhere, but it will disappear after the player presses the ‘p’ key to pick it up.

Cylinder:

The cylinder is place under the floor in the music room in the beginning of the game; it will translate upward in a certain condition.

Fishes:

Each of the fishes is composed of an oval and a cone with a certain scale. They are moving in the aquarium. One is in an elliptical motion, and the other is in a rectangle motion. They move whenever DrawFish() and DrawFish2() are called. They are moving in a fixed path, not randomly.

Collision Detection

In order to make sure that the objects do not move through each other or to be more accurate, not to collide with on another. We uased a simple template we created. Each object schell was represented as a rectangle. This rectangle was represented by four equalities.

```
params.left>0-modelLength
&&
params.left<83
&&
params.forward>0
&&
params.forward<7+modelLength
```

The first two equalities are used to constraint in the x-axis and the next two equalities constraint in the z-axis. We used the same equalities for moving objects aswell. However, instead of the number like 0,83, and 7 that are written above, they were replaced with variables, that changed with the motion of the objects.

Sound Effects

To integrate sound into out game, we used OPENAL. Although OPENAL is hard to use, it is very a free, powerful and flexible API. OPENAL allows us to position the source of the sounds and listener using OPENGL x, y and z coordinates. Position the sources of sound gives the user a sense of reality in the game since for instance, if the user is facing the source of the sound the sound will be louder; however, if the user is not facing the source of the sound and the sound is emitted from the left side of the player, then the sound will be emitted only by the left speaker and the player will know that the sound is coming from the left.

For instance, to make the following sequence, the coordinates of the plane are updated together with the coordinates of the source of the sound, and as it moves away from the listener, the sound decreases giving an effect of reality.

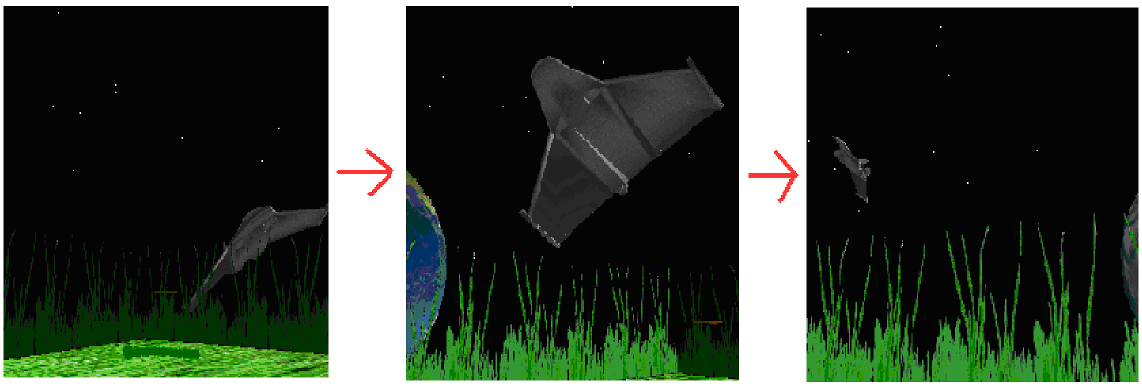

Figure : depics the sound effect done by the ship going away from the lisener as the sound becomes less louder

Figure 4.2.5 Spaceship Sound Effect

Sound effects like the footsteps were made using 4 different footstep files that are randomly chose as the main player advances. Other effects like the background music loop over and over during the game. Sound produced by models can not be heard until the players come close enough to the model and press “t”.

Restrictions

The restrictions made for the room and for the objects are found in the checkRoute. This is also done with a simple template we created. For simplicity of creating restrctions, all the restricted areas are made of rectangular areas.

```
params.left>0-modelLength
&&
params.left<83
&&
params.forward>0
&&
params.forward<7+modelLength
```

The first two equalities are used to constraint in the x-axis and the next two equalities constraint in the z-axis. This way if we wanted to change anything in the desing or if we wanted to translate something anywhere it will be easy for us to find it and change it.

In order to specifiy whether the user is allowed to pick up an object or not, we used restrcitions to decide wheter the player was within a certain proximity. However, we used the state variables to make sure the object has not already been picked up, or the user has collected other required objects.

All this control was made from a function called restriction that we emplemnted. It also returns an int to say what descion it has taken when it is called so other functions can work coresspondongly, such as the menu function. For example if the restriction function returns “8” the menu knows it has to add “blueCrystal” to the menu.

5.0 GAME DESCRIPTION

5.1 3D Maze

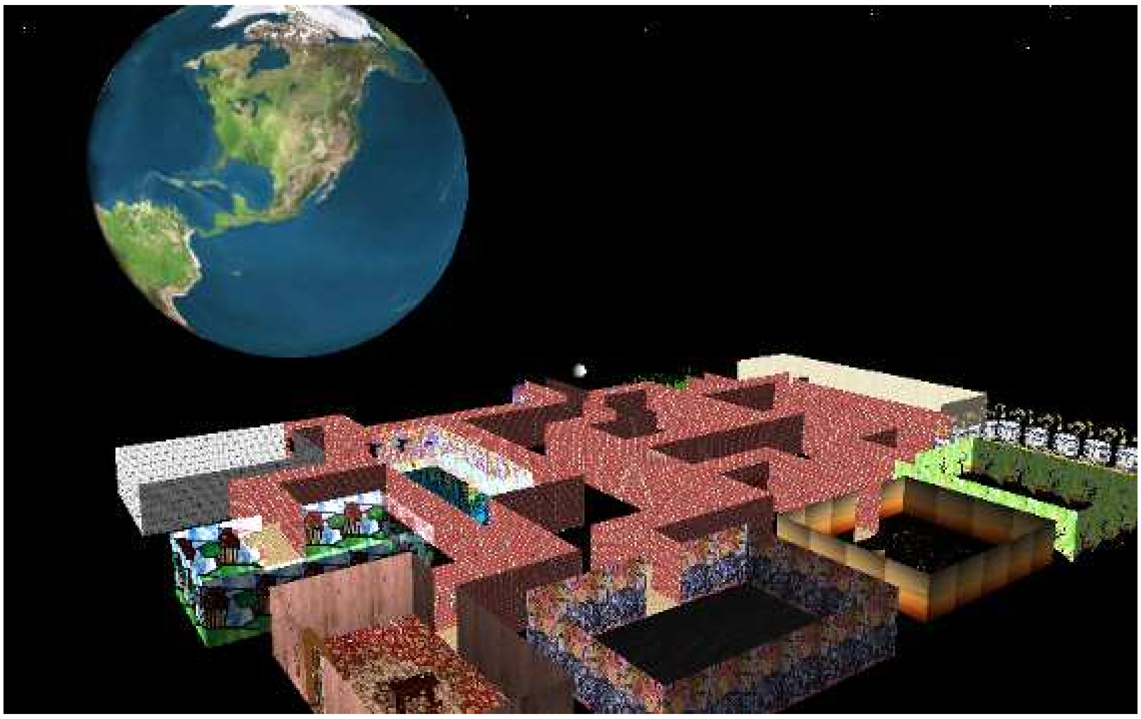

Figure 5.1.1 3D Maze Overview

The primary objective of the game is to find all four crystals, and to do so, the player must navigate through a complex maze and successfully solve or do all riddles and puzzles presented in each of our unique rooms. The maze is enclosed in brick walls and although none leads to a dead end, a wrong turn will force the player to track back to the beginning.

The start of the game places the player near the bottom center of our world plane and immediately, the player will notice that the game will involve finding his or her way through the maze. For the simplicity of our game, we did not make the maze too complicated but our maze layout can be easily modified for a much harder navigation. Our game is made so that the player may have to visit a room more than once and by doing so, the player should become familiar with the paths to each of the different rooms quite easily.

5.2 Rooms

There are a total of nine unique rooms in Crystal Maze game. Each room requires the player to perform a certain action in order to move onto the next step. The player must solve riddles or do certain actions in sequence for the next hint to be given in each of the rooms.

Room 1:

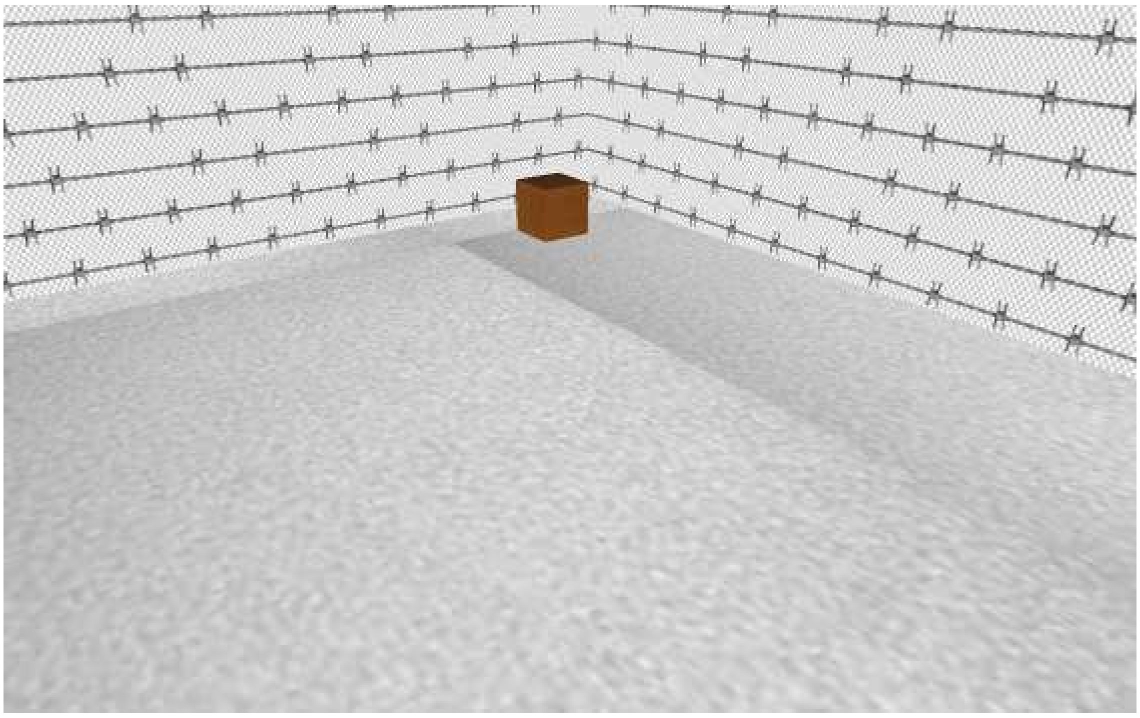

Figure 5.2.1 Room 1

The major theme of this room is bank. A safety box is located in the room where the player must enter a secret code to open the safety box. When the player correctly enters the secret code, one of four crystals, the blue crystal is given to the player.

Room 2:

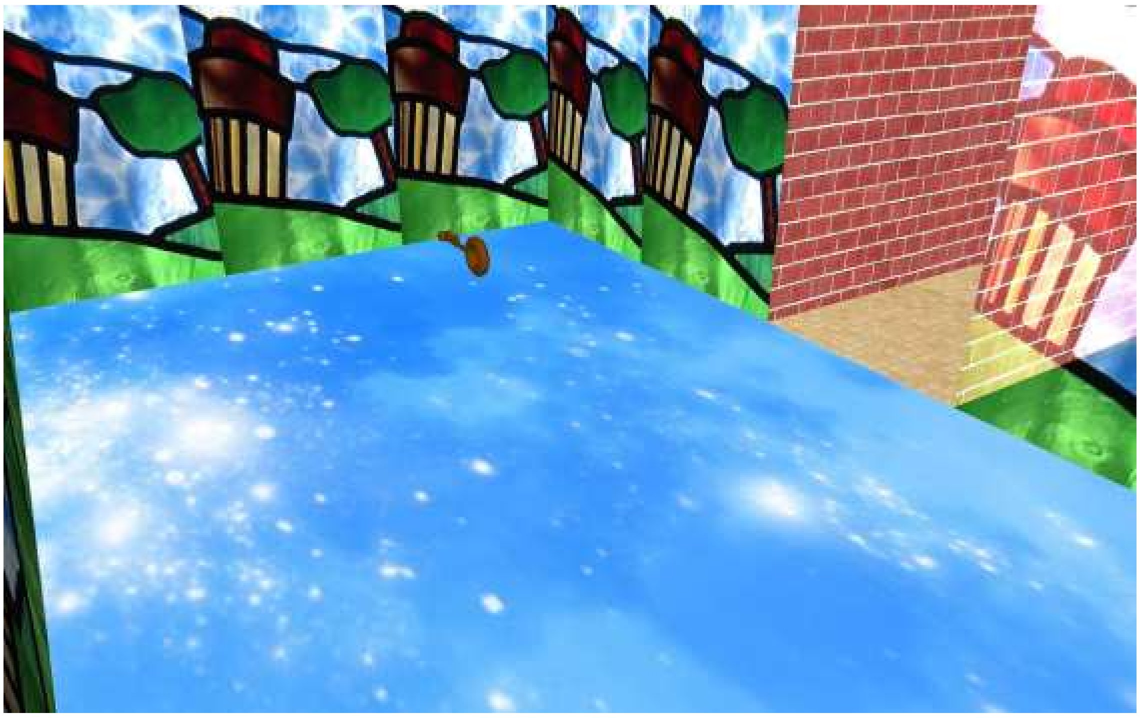

Figure 5.2.2 Room 2

This room is one of the colourful room in our game. In this room the player must locate the key and pick it up using the ‘P’ button on the keyboard. Once the key is picked up, it will disappear and will be available to use by the right-click button of the mouse.

Room 3:

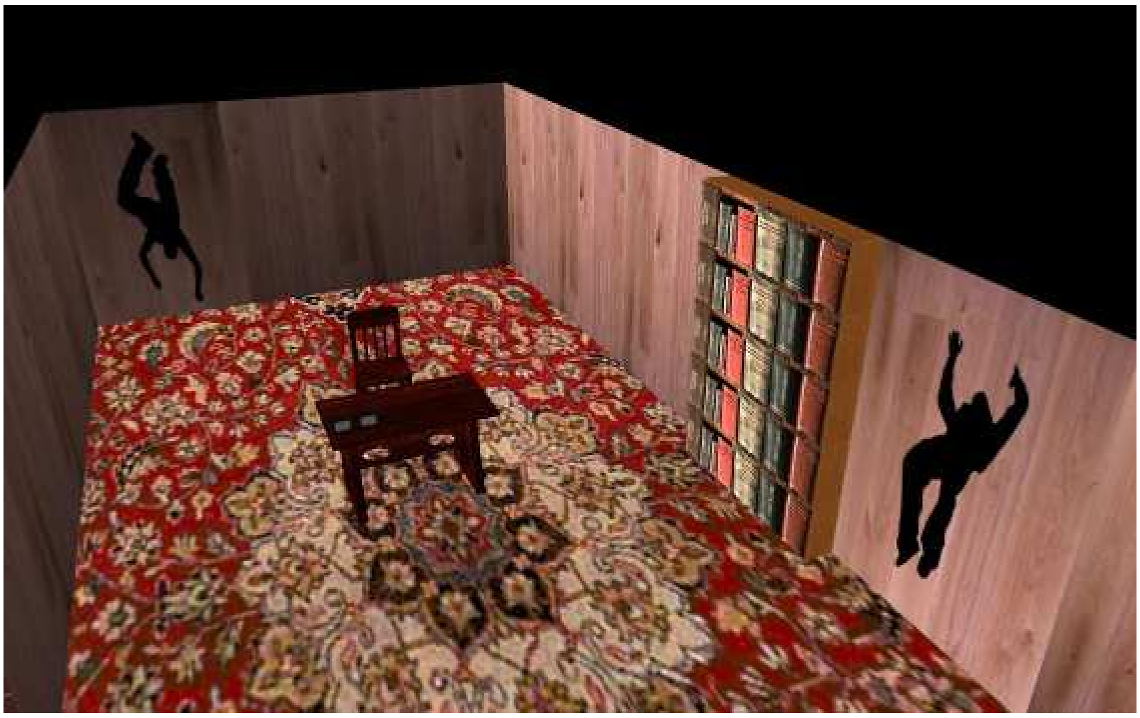

Figure 5.2.3 Room 3

This room contains wooden table and chair as well as books to make a library or a study room type theme. A reading glasses is placed on the table and the player must pick up the glasses using the ‘P’ key. Also, there is a bookcase thin the room which is a hidden door that opens. To open the hidden door, the player must possess the book he or she got from Room 7 and must press the ‘O’ key. Once the door is opened, the player will be given the yellow crystal.

Room 4:

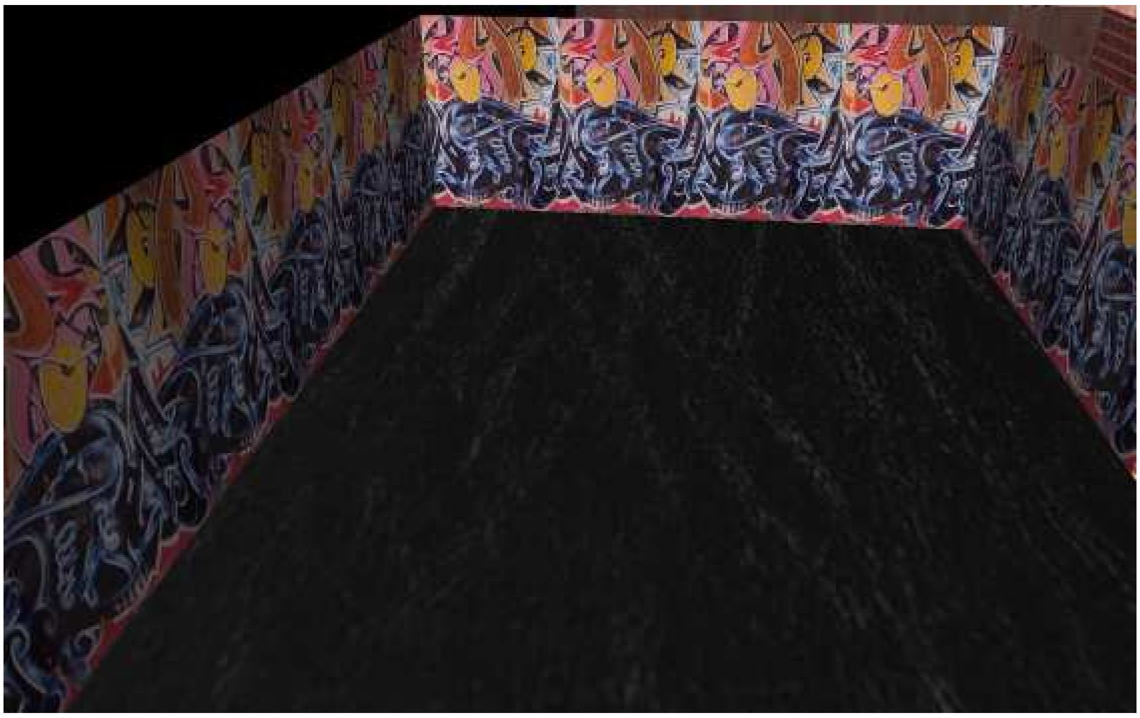

Figure 5.2.4 Room 4

This room has an underground street theme with paintings on the wall. There are no unique objects in this room, however, by using the glasses the player picked up from Room 3, the player will be able to read a graffiti on the wall that explains the next steps. To put on the glasses, the player must use the right-click button and select the glasses from the list of items he or she currently has. The steps given in this room refers to the steps he or she must perform in Room 6.

Room 5:

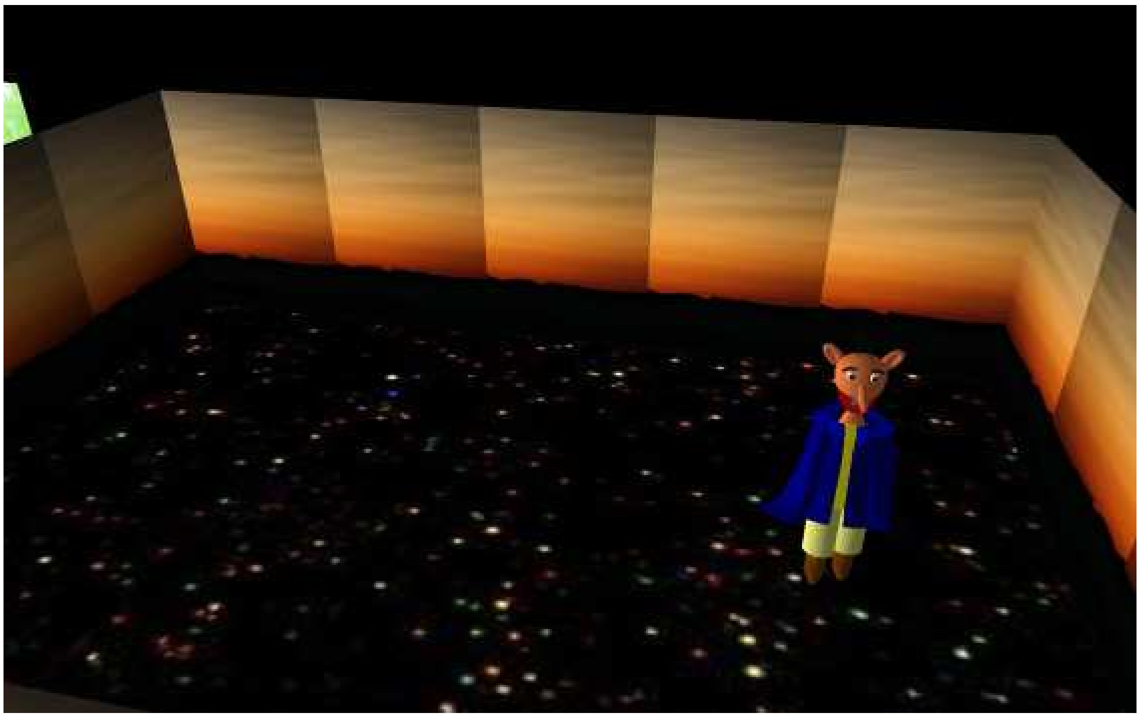

Figure 5.2.5 Room 5

The main theme of this room is to be a space like, mystical room. The Wizard is located in this room and will give the player a bell when the player talks to the Wizard. To talk to the Wizard, the player must press the ‘T’ key. The Wizard will give a bell as well as say a hidden riddle which is a hint to the next step.

Room 6:

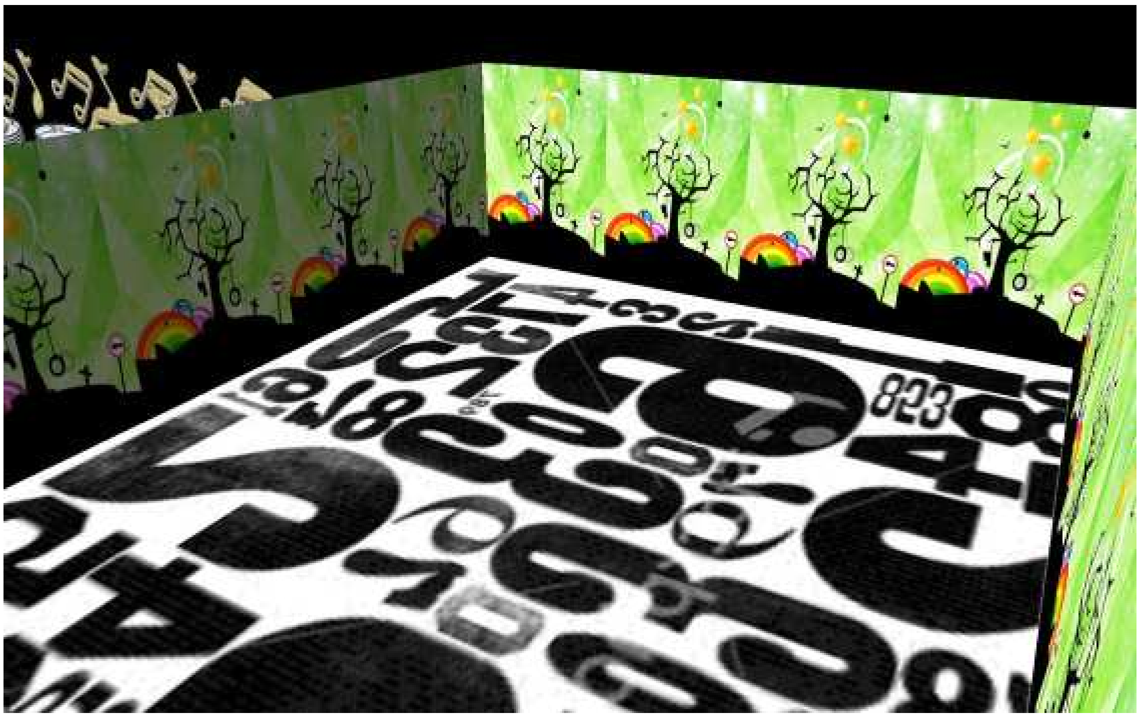

Figure 5.2.6 Room 6

This room has a checker board layout and the colors of the square box lights up when the player steps on it. The player must perform a sequence of steps he or she was given in Room 4 in order to move on in the game. The player must start at the middle of the room and follow the correct sequence of steps. When the player is making the correct steps, the color of the squares will appear different from randomly stepping on the floor. Once the player performs the steps successfully, a secret code will be given which is used to open the safety box in Room 1.

Room 7:

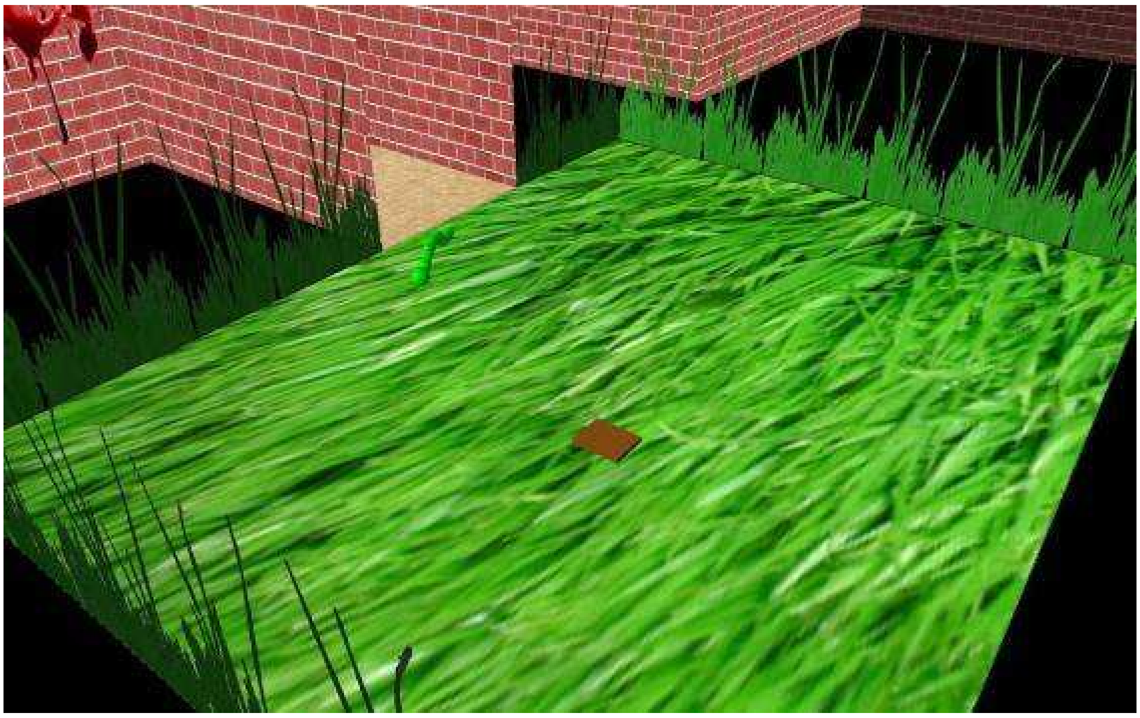

Figure 5.2.7 Room 7

The theme given to this room is a grass field. There is a worm crawling around the room as well as a book on the grass field. The player must pick up the book using the ‘P’ key and the book will give the next hint for the game.

Room 8:

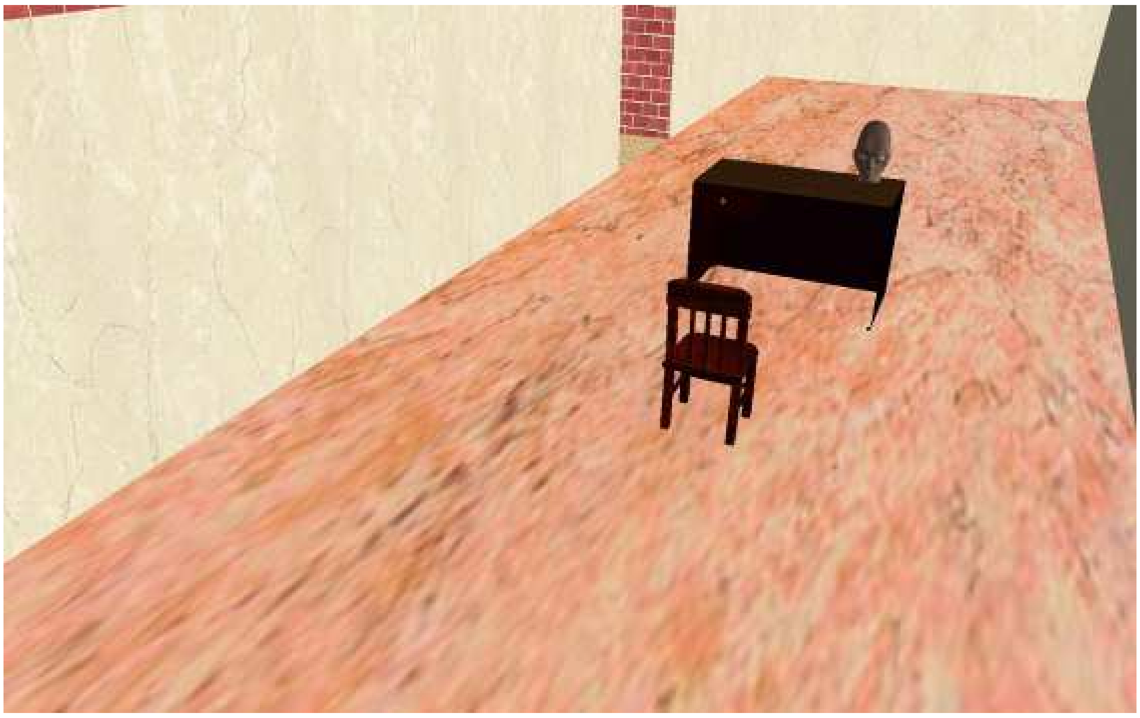

Figure 5.2.8 Room 8

This room has a simple office look to it. There is a table and chair in this room but the table has a drawer that can be opened. The player needs the key to open the drawer and when the player opens the drawer using the ‘O’ key, he or she will be a red crystal that lies within the drawers.

Room 9:

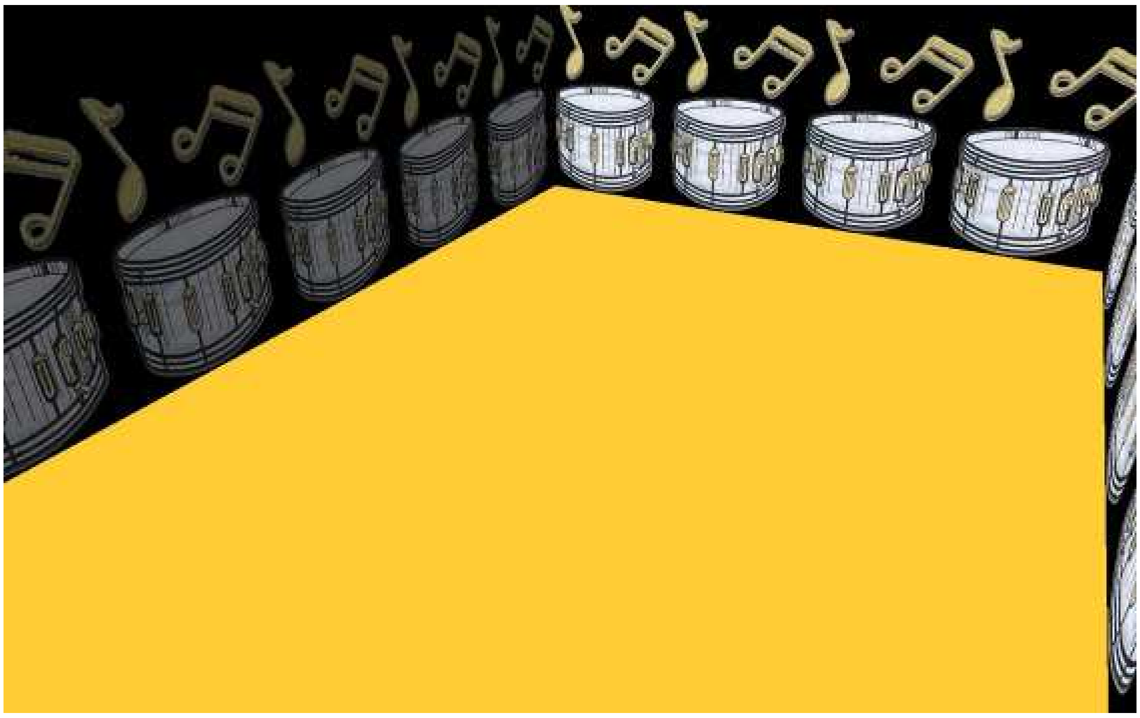

Figure 5.2.9 Room 9

This is an empty room but when the player rings the bell he or she got from the Wizard, a cylinder will come up from the ground. The cylinder will have the last and final crystal, the green crystal on top of it.

When the player collects all four crystals and places them on the crystal stand, the game ends. To place the crystal on the crystal stand, the player must right-click on the mouse button and selects the crystal he or she wishes to place on the stand. The player will be greeted with a message telling him or her that the game is completed.

6.0 DIFFICULTIES FACED

In the camera part, looking at different directions is achieved by rotating the world around the camera. This achieved the objective, but made it difficult to implement features that are related to the camera.

First, when we wanted to do the second person view, we faced some difficulty when placing the camera behind the main character since the character will rotate with the world when the player looks into different directions. Therefore, we had to draw the main character before doing the rotations for the whole world one time for the second person view, and drawing the main character again after the rotations for the free moving camera and first person view. This difference in the location of the draw function for the main character is controlled by a Boolean. If the camera was rotated to achieve the view rotations, we would not have needed to do that.

Second, when we have placed one light to imitate the sun effect, the position of the light looked like it was changing when we rotate the world. In fact, the light is fixed, but because the world rotates and the camera does not rotate with the world, the light looked like it was changing direction with respect to the world. To overcome this difficulty, we had to change the position of the light source according to how much the world has been rotated, so we have a moving light with the world now. However, moving the light to the right position every time the world is rotated has created the effect of a fixed light in one direction that we originally wanted.

When it came to designing the floor and the walls, we needed a quick way to generate them, with a code template. So the difficulty was in actually creating the loops and the draw functions that will suite our need.

Also for the restrictions and the collision detection, it was not easy to immediately think of a way to implement them. We had to brainstorm more than once to make sure that this will apply to all our cases. Not only that but how we are going to manage the functions.

This leads us to another difficulty we faced, which was actually how each part was going to interact with one another. We needed to make it modular, which was not very clear how that was going to happen at first glance. It required us to sit down see each group members’ strengths so that we could assign tasks and modules accordingly. We then developed a way of communication that was efficient.

As with any software development, difficulties will arise and it’s our job as software engineers to overcome these obstacles by doing researching and collaborating. Our group did a great job communicating with each other when we ran into problems and made sure that we were on track to finish the project.

7.0 FUTURE IMPROVEMENTS

There is always room for improvements in any software, especially a 3D computer game. The amount of features we can add to such game is nearly infinite and it’s just a matter of which feature would make the greatest impact on the game if implemented. If given the time and resource to do so, we would have improved the following in our game.

The biggest shortfall of our game is the fact that there is only one level to play. One of the major improvements we can make to the Crystal Maze game is to have layers of levels with increasing complexity and difficulty of the maze and the riddles. A more thoughtful storyline with the possibility of adding trailers between levels will definitely be a plus. Outside of the complexity and storyline details, some key features in our game could also be improved.

The second person view can be improved to have the camera change position when the player looks at different directions to avoid the cases where the camera becomes behind other objects. This can be achieved by comparing the position of the camera to the restrictions of the objects and making a decision at this point to fix the camera position.

The objects that the player has collected can be placed on one corner of the screen which will make it easier and faster for the player to know the objects collected instead of right clicking the mouse to see them.

Another important improvement could be the development of a texture loader that can load any size bmp files. This will make the texture mapping look better. We can also make the game become network enabled, where more than one user can play the same instance of the game from different computers.

On top of all the possible improvements we mentioned for the game, there will always be more room for improvements. Imagination is the only limiting factor when creating a 3D game and it will always be a good idea to have a suggestion forum of some sort when developing and game. This will allow the developers to get infinite amount of input and these inputs should be taken into consideration when updating the software.

8.0 CONCLUSION

It was a challenge to create a fun, working and visually interesting 3D game in the timeframe given to us. We were able to create a 3D maze game that also required thinking and problem solving. Our game required the use of several external resources and libraries as well as our own way of implementing basic and advanced features to the game. Some of the key advanced features implemented were transparency, animation, collision detection and sound effects. Adding such features improved the game play and showed what OpenGL was capable of doing when creating a 3D game. Our group learned a many interesting techniques involved in creating a 3D game and this project gave us an insight to what game creation process was like. Overall, our Crystal Maze game was a success that not only has a great story line and functions but many advanced features as well.

Reference

[1] Fallout Software, “The OpenGL Light Bible,” http://www.falloutsoftware.com/tutorials/gl/gl8.htm.

[2] OpenGL.org, “Avoiding 16 Common OpenGL Pitfalls,” 2008, http://www.opengl.org/resources/features/KilgardTechniques/oglpitfall/.

[3] Antonio Ramires Fernandes, “GLUT Tutorial,” http://www.lighthouse3d.com/opengl/glut/index.php?12.

[4] Gamedev.net “OpenGL Tutorials,” 2008, http://nehe.gamedev.net.

[5] Ultimate 3D Links, “Free Objects,” http://www.3dlinks.com/Free_3D_Objects.cfm

[6] E. Angel, *Interactive Computer Graphics: A Top-Down Approach Using OpenGL 4th ed.* Addison-Wesley, 2006.