

The God Core  
A Science Fiction Video Game Developed in C++

Author: Jeremy Greenburg  
Mentor: Dr. Alton Coalter  
Second Reader: Joshua Guerin

July 9, 2016

# Contents

<b>1</b>	<b>Preamble</b>	<b>3</b>
<b>2</b>	<b>Setting</b>	<b>3</b>
<b>3</b>	<b>Programming</b>	<b>3</b>
3.1	The Language . . . . .	3
3.2	OpenGL . . . . .	3
3.3	SOIL . . . . .	3
3.4	FMOD . . . . .	3
3.5	Classes . . . . .	3
3.6	Problems and Frustrations . . . . .	5
<b>4</b>	<b>Appendices</b>	<b>5</b>
4.1	Source Code . . . . .	5
4.2	Database . . . . .	91
4.3	Scripts . . . . .	91
4.4	Images . . . . .	91
4.5	Music . . . . .	91
4.6	Sounds . . . . .	91

# 1 Preamble

# 2 Setting

# 3 Programming

## 3.1 The Language

## 3.2 OpenGL

## 3.3 SOIL

## 3.4 FMOD

## 3.5 Classes

Of the two years of my project, I have constructed *CLASSNUM* classes, across *2CLASSNUM* files. I will describe the classes and there methods here in brief, but the complete files can be found in Appendix 4.1 for your viewing pleasure.

### 3.5.1 CameraControl

The CameraControl class is designed to control and manipulate the player's perspective as they navigate through the game. It contains two ordered triples of floating point numbers: The xyz location of the player, and the rotation along the x axis (looking left/right), the y axis (up/down), and the z axis (barrel roll). It also contains two additional floating point values the, movement speed and the turning speed.

The CameraControl class contains four methods for rotating the camera, lookUp and lookDown which affect the y rotation, as well as lookLeft and lookRight which affect the x rotation.

There are six movement methods, moveForward and moveBackward are designed to move the camera forward and backwards in respect to where the player is looking currently. This proved quite frustrating to find and I had to ask many math majors for their input until Robert Deyoso was able to help me pin down a formula. These functions adjust the the x and z coordinates as follows:

$$z := z \pm \text{moveSpeed} * \cos(\text{radian}(x\_angle))$$

$$x := x \mp \text{moveSpeed} * \sin(\text{radian}(x\_angle))$$

strafeLeft and strafeRight, again, move the camera directly left and directly right according to where the camera is looking. The formula is similar to the above, except the angle is increased or decreased by  $90^\circ$ .

moveUp and moveDown were the easiest functions to derive, as up and down are independent of the direction the camera is facing, so it is simply increasing and decreasing the y value respectively.

invertCam increases the z angle by  $180^\circ$  to flip the world upside down. resetCam resets the 3-tuples to their default values (0, 0, -1) for the position and (0, 0, 0) for the angles.

While those functions work to modify the values within the class, Display is the method that actually moves and rotates the camera within the world. It calls glRotate three times to rotate the camera along the respective axis, and then calls glTranslate to move the camera into the correct position.

### 3.5.2 HeadsUpDisplay

The HeadsUpDisplay class is a complex class that overlays a display to present information or display aesthetics such as the helmet's bounds to the player. The class contains four constant integers to designate the boundaries of the screen (bottom, top, left right), as well as two more integers dimTime and darkTime to control the length of the dim and dark functions. There are three strings held by the class, currentAlert dictates information that will be printed to the center of the screen, currentText is what the user is typing, and currentInput is what the user has entered as input into the developer console (for more information, see the next section). The developer console dev, as well as a TextEngine helmet, also reside in this class.

DisplayHUD is the activation method, it calls prepare2D, drawHUD, and prepare3D. This is called in the main function every frame unless the player is in a screen.

prepare2D prepares OpenGL for rendering 2D images (the HUD is the last item that is drawn to the screen every frame, therefore nothing can cover it) by disabling everything related to the depth buffer and projecting everything onto a matrix that is orthogonal to the screen bounds.

### 3.5.3 Rectangle

For collision purposes, when a Rectangle is created it calculates the Plane equation of the rectangle (Form  $ax + by + cz + d = 0$ ).

This equation is calculated using the first three corners of the rectangle (Calling them A, B, and C) as follows:

$$\vec{AB} = \begin{bmatrix} Bx - Ax \\ By - Ay \\ Bz - Az \end{bmatrix} \quad \vec{AC} = \begin{bmatrix} Cx - Ax \\ Cy - Ay \\ Cz - Az \end{bmatrix}$$

$$a = \vec{AB}_2 * \vec{AC}_3 - \vec{AB}_3 * \vec{AC}_2$$

$$b = \vec{AB}_3 * \vec{AC}_1 - \vec{AB}_1 * \vec{AC}_3$$

$$c = \vec{AB}_1 * \vec{AC}_2 - \vec{AB}_2 * \vec{AC}_1$$

$$d = aAx + bAy + cAz$$

We can also derive the norm of the plane using the equation  $\sqrt{a^2 + b^2 + c^2}$

### 3.5.4 CollisionEngine

This determines when the player has collided with an object in the world. There are two types of collisions: player-object collisions and player-wall collisions.

Player object collisions are simple to detect, as both the player and the object can be placed within imaginary "bounding spheres" that extend around the player and object. Collision can be detected with this formula:  $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} < r_2 + r_1$  If the distance between the two spheres is less than the sum of the radii of the two spheres, they must be colliding.

Player-wall collisions were much harder to reconcile. Because walls tend to be long and thin, you can't simply place one within a bounding sphere, the resulting sphere would simply be too massive.

To rectify that, the collision is split into two phases: broad and narrow.

In the broad phase, we use the plane equation  $ax + by + cz + d$  that is derived in the Rectangle section. We use the formula  $\frac{ax + by + cz + d}{\sqrt{a^2 + b^2 + c^2}}$ , where x, y, and z are the player's x, y, and z coordinates. If the resulting value is less than the radius of the player's bounding sphere, the player has hit that plane and we move onto the narrow phase.

In the narrow phase, each wall is aligned on an axis: x, y, or z. We simply take the largest and smallest values of the coordinates on that axis (for instance, if the wall is x aligned, we take the largest and smallest x value). If the sphere is in between the two values, the player has hit the wall. Otherwise, they hit the plane but not the wall.

### 3.5.5 Console

### 3.5.6 MusicManager

### 3.5.7 TextEngine

### 3.5.8 SaveManager

### 3.5.9 Keyboard

### 3.5.10 Rectangle and Triangle

## 3.6 Problems and Frustrations

# 4 Appendices

## 4.1 Source Code

### 4.1.1 main.cpp

```
1  /*****\
2  * main.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file creates an OpenGL window to display the game *
8  * and promptly passes control over to the GameManager object*
9  \*****/
10
11 // Because doth openGL demandeth
12 #include <cstdlib>
13 // OpenGL API
14 #include <GL\glut.h>
15
16 // The Game manger
17 #include "GameManager.h"
18 GameManager Overlord;
19 // Save manager
20 #include "SaveManager.h"
21 // Return codes
22 #include "Return.h"
23 // System log
24 #include "Logger.h"
25
26 // Normal key presses
27 void normal(unsigned char key, int x, int y);
28
29 // For key releases
30 void key_up(unsigned char key, int x, int y);
31
32 // For Special keys
33 void special(int key, int x, int y);
34
35 // Mouse clicks
36 void mouse(int button, int state, int x, int y);
37
38 // Mouse movement
```

```

39 void motionPassive(int x, int y);
40
41 // Changing Window size (Not exactly working as hoped...
42 void changeSize(int w, int h);
43
44 // Initializes GLUT callbacks and returns true if core.sav exists (false otherwise
45 )
46 bool initGame(int argc, char **argv);
47
48 // Manages the game's scenes
49 void manageScenes();
50
51 GLfloat light_diffuse[] = { 0.3f, 0.3f, 0.3f, 0.5f };
52 GLfloat light_position[] = { 0.0f, 1.0f, 0.0f, 0.0f };
53 GLfloat mat_specular[] = { 0.3f, 0.2f, 0.3f, 0.5f };
54 GLfloat mat_shininess[] = { 3.0f };
55 GLfloat lmodel_ambient[] = { 0.6f, 0.6f, 0.6f, 1.0f };
56
57 using namespace std;
58
59 //***** FUNCTION DEFINITIONS *****/
60
61 int main(int argc, char **argv)
62 {
63     SaveManager s;
64     s.saveLevel("Hello");
65
66     Overlord.canContinue = initGame(argc, argv);
67
68     // Begin the game
69     glutMainLoop();
70
71     // If we ever get here, something bad happened
72
73     Logger log;
74     log.logLine("ERROR: GlutMainLoop exited early");
75
76     return EXIT_EARLY;
77 }
78
79 bool initGame(int argc, char **argv)
80 {
81     // Obliterate log file
82     Logger log;
83     log.nuke();
84
85     // Initialize GLUT
86     glutInit(&argc, argv);
87
88     // Create window
89     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
90     glutInitWindowPosition(50, 50);
91     glutInitWindowSize(500, 500);
92     glutCreateWindow("The God Core");
93
94     // register callbacks

```

```

94     glutDisplayFunc(manageScenes);
95     glutReshapeFunc(changeSize);
96     glutIdleFunc(manageScenes);
97     glutPassiveMotionFunc(motionPassive);
98     glutMouseFunc(mouse);
99     glutKeyboardFunc(normal);
100    glutKeyboardUpFunc(key_up);
101    glutSpecialFunc(special);
102
103    // Prebuilt function that works transparency
104    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
105
106    // Enable transparency
107    glEnable(GL_BLEND);
108    // Enable depth buffer
109    glEnable(GL_DEPTH_TEST);
110    // Let there be light!
111    glEnable(GL_LIGHTING);
112    // First light source
113    glEnable(GL_LIGHT0);
114    // Light doesnt turn everything grey
115    glEnable(GL_COLOR_MATERIAL);
116    // Light properties
117    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
118    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
119    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
120    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
121    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
122    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
123
124    glutWarpPointer(300, 300);
125
126    // Start in Fullscreen
127    glutFullScreen();
128
129    SaveManager SaveSystem;
130    return SaveSystem.checkSave();
131 }
132
133 // Everything below here is just passed along to the overlord
134
135 void mouse(int button, int state, int x, int y)
136 {
137     Overlord.mouse(button, state, x, y);
138 }
139
140 void motionPassive(int x, int y)
141 {
142     Overlord.motionPassive(x, y);
143 }
144
145 void changeSize(int w, int h)
146 {
147     Overlord.changeSize(w, h);
148 }
149

```

```

150 void manageScenes()
151 {
152     Overlord.manageScenes();
153 }
154
155 void normal(unsigned char key, int x, int y)
156 {
157     Overlord.normal(key, x, y);
158 }
159
160 void key_up(unsigned char key, int x, int y)
161 {
162     Overlord.key_up(key, x, y);
163 }
164
165 void special(int key, int x, int y)
166 {
167     Overlord.special(key, x, y);
168 }

```

#### 4.1.2 CameraControl.h

```

1  /*****\
2  * CameraControl.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the CameraControl *
8  * Class, which stores: *
9  *     The x, y, z ordered triple of the player's location *
10 *     The degree to which the player is turned, along *
11 *         the x, y, and z axes *
12 * And contains methods to translate the player along *
13 * 3D space *
14 \*****/
15
16 #ifndef CAMERA_CONTROL_H
17 #define CAMERA_CONTROL_H
18
19 class CameraControl
20 {
21 private:
22     // Speeds for moving and rotating
23     double moveSpeed = 0.1f, turnSpeed = 0.5f;
24
25 public:
26     // Negatively adjusts angle and modifies lx
27     void lookLeft();
28     // Positively adjusts angle and modifies lx
29     void lookRight();
30     // Positively adjusts angle and modifies ly
31     void lookUp();
32     // Negatively adjusts angle and modifies ly
33     void lookDown();
34     // Translate the camera to the left
35     void strafeLeft();

```



```

36         // Translates the to the right
37         void strafeRight();
38         // Translates the camera forwards
39         void moveForward(int mod);
40         // Translate the camera backards
41         void moveBackward(int mod);
42         // Moves the camera positively along the Y axis
43         void moveUp();
44         // Moves the camera negatively along the Z axis
45         void moveDown();
46         // Flips the camera
47         void invertCam();
48         // If the player begins to run
49         void increaseSpeed();
50         // If the player begins to walk
51         void decreaseSpeed();
52         // Resets the camera to it's initial state
53         void resetCam();
54         // calls gluLookAt
55         void Display();
56
57         // Location of the camera
58         double x =0.0, y = 0.0, z = -1.0;
59         double prevx, prevz;
60         // Angles of rotation
61         double x_angle = 0.0, y_angle = 0.0, z_angle = -1.0;
62     };
63
64 #endif

```

#### 4.1.3 CameraControl.cpp

```

1  /*****\
2  * CameraControl.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the CameraControl *
8  * Class. For more information, see CameraControl.h *
9  \*****/
10
11 // Class definition
12 #include "CameraControl.h"
13
14 // For sin()
15 #include <cmath>
16
17 // glut is unhappy when cstdlib isn't here :/
18 #include <cstdlib>
19
20 // OpenGL API
21 #include <GL\glut.h>
22
23 // To display Suit Warnings
24 #include "TextEngine.h"
25

```

```

26 // To include Globals Variables
27 #include "Globals.h"
28
29 // For converting degrees to radians
30 const double PI = 3.14159;
31
32 // Takes in an angle, in degrees, and returns the angle in radians
33 double toRadian(double angle)
34 {
35     return angle * PI / 180;
36 }
37
38 void CameraControl::lookLeft()
39 {
40     if (!isPaused)
41     {
42         x_angle -= 3 * turnSpeed;
43
44         // To avoid potential underflow errors
45         if (x_angle < 0)
46         {
47             x_angle += 360;
48         }
49     }
50 }
51 void CameraControl::lookRight()
52 {
53     if (!isPaused)
54     {
55         x_angle += 3 * turnSpeed;
56
57         // To avoid potential overflow errors
58         if (x_angle > 360)
59         {
60             x_angle -= 360;
61         }
62     }
63 }
64
65 void CameraControl::lookUp()
66 {
67     if (!isPaused)
68     {
69         y_angle -= 2 * turnSpeed;
70
71         // To avoid potential underflow errors
72         if (y_angle < 0)
73         {
74             y_angle += 360;
75         }
76     }
77 }
78
79 void CameraControl::lookDown()
80 {
81     if (!isPaused)

```

```

82         {
83             y_angle += 2 * turnSpeed;
84
85             // To avoid potential overflow errors
86             if (y_angle > 360)
87             {
88                 y_angle -= 360;
89             }
90         }
91     }
92
93     void CameraControl::strafeLeft()
94     {
95         prevz = z;
96         prevx = x;
97         // Angles + 90 degrees for an angle that is perpendicular to x_angle
98         z = z + moveSpeed * cos(toRadian(x_angle + 90));
99         x = x - moveSpeed * sin(toRadian(x_angle + 90));
100    }
101
102    void CameraControl::strafeRight()
103    {
104        prevz = z;
105        prevx = x;
106        // Angles - 90 degrees for an angle that is perpendicular to x_angle
107        z = z + moveSpeed * cos(toRadian(x_angle - 90));
108        x = x - moveSpeed * sin(toRadian(x_angle - 90));
109    }
110
111    void CameraControl::moveForward(int mod)
112    {
113        prevz = z;
114        prevx = x;
115        z = z + moveSpeed * mod * cos(toRadian(x_angle));
116        x = x - moveSpeed * mod * sin(toRadian(x_angle));
117    }
118
119    void CameraControl::moveBackward(int mod)
120    {
121        prevz = z;
122        prevx = x;
123        z = z - moveSpeed * mod * cos(toRadian(x_angle));
124        x = x + moveSpeed * mod * sin(toRadian(x_angle));
125    }
126
127    void CameraControl::moveUp()
128    {
129        y -= moveSpeed;
130    }
131
132    void CameraControl::moveDown()
133    {
134        y += moveSpeed;
135    }
136
137    void CameraControl::invertCam()

```

```

138 {
139     z_angle += 180;
140 }
141
142 void CameraControl::resetCam()
143 {
144     x = 0.0;
145     y = 0.0;
146     z = -1.0;
147     x_angle = 0.0;
148     y_angle = 0.0;
149     z_angle = 0.0;
150
151 }
152
153 void CameraControl::increaseSpeed()
154 {
155     moveSpeed *= 2;
156 }
157
158 void CameraControl::decreaseSpeed()
159 {
160     moveSpeed /= 2;
161 }
162
163 void CameraControl::Display()
164 {
165     // To stop eternal movement
166     glLoadIdentity();
167
168     // Rotate along proper axes
169     glRotatef(y_angle, 1, 0, 0);
170     glRotatef(x_angle, 0, 1, 0);
171     glRotatef(z_angle, 0, 0, 1);
172
173     // Translate along the Plane
174     glTranslatef(x, y, z);
175 }

```

#### 4.1.4 CollisionEngine.h

```

1  /*****\
2  * CollisionEngine.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * * *
7  * This file creates the decleration of the CollisionEngine *
8  * class, which uses sweet sweet math to determine how the *
9  * player interacts with his environment *
10 \*****/
11
12 #ifndef COLLISION_ENGINE_H
13 #define COLLISION_ENGINE_H
14
15 class CollisionEngine
16 {

```

```

17 private:
18     // Determines if wall/door collision occurred
19     bool collideWalls();
20     // Determines if other collision occurred
21     bool collideObjects();
22     // Determines if an object can be interacted with
23     void checkInteract();
24 public:
25     // Master function that calls others
26     bool collide();
27
28 };
29
30 #endif

```

#### 4.1.5 CollisionEngine.cpp

```

1  /*****\
2  * CollisionEngine.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the CollisionEngine *
8  * class. For more information, see SaveManager.h *
9  \*****/
10
11 #include "CollisionEngine.h"
12
13 // For the Cam
14 #include "Globals.h"
15 // absolute value
16 #include <cmath>
17
18 // System Log
19 #include "Logger.h"
20
21 using namespace std;
22
23 const double PLAYER_RADIUS = .3;
24 const double OBJECT_RADIUS = 1; // Object interactivity radius
25
26 void CollisionEngine::checkInteract()
27 {
28     activeSwitch = NULL;
29     activeTerminal = NULL;
30     // Auto don't work in these parts
31     for (unsigned int i = 0; i < switches.size(); i++)
32     {
33         // Apprently. Somehow. Values are mirrored here.
34         // So you have to do addition instead of subtraction to get the
35         // proper distance
36         // As I noticed after observing the distance increase as I moved
37         // towards an object.
38         // And I am stumped as to why this happens.
39         double distance = pow((switches[i].getX() + Cam.x), 2) + pow((
40             switches[i].getY() + Cam.y), 2) + pow((switches[i].getZ() + Cam

```

```

        .z), 2);
38     distance = sqrt(distance);
39     double radii = (PLAYER_RADIUS + OBJECT_RADIUS);
40
41     if (distance < radii)
42     {
43         interactivity = true;
44         activeSwitch = &switches[i];
45         return;
46     }
47 }
48
49 for (unsigned int i = 0; i < terminals.size(); i++)
50 {
51     double distance = pow((terminals[i].getX() + Cam.x), 2) + pow((
        terminals[i].getY() + Cam.y), 2) + pow((terminals[i].getZ() +
        Cam.z), 2);
52     distance = sqrt(distance);
53     double radii = (PLAYER_RADIUS + OBJECT_RADIUS);
54
55     if (distance < radii && terminals[i].isOn)
56     {
57         interactivity = true;
58         activeTerminal = &terminals[i];
59         return;
60     }
61 }
62
63 interactivity = false;
64 }
65
66 bool CollisionEngine::collideObjects()
67 {
68     /*
69     // If bounding spheres intersect
70     double distance = pow((x + Cam.x), 2) + pow((y + Cam.y), 2) + pow((z + Cam
        .z), 2);
71     distance = sqrt(distance);
72     double radii = (PLAYER_RADIUS + .1);
73     if (distance < radii) // Figure out a standard radius for player,
        dynamically take radius from object
74     {
75         return true;
76     }
77     */
78     return false;
79 }
80
81 bool CollisionEngine::collideWalls()
82 {
83     if (collision == false)
84     {
85         return false;
86     }
87
88     // Gotta check doors first

```

```

89         // And if you hit an open door
90         // You just ignore collision
91         // Because otherwise you can't fit
92         for (auto i : doors)
93         {
94             double distance = fabs(Cam.x * i.a + Cam.y * i.b + Cam.z * i.c + i
                                     .d); // Distance from door
95
96             if ((distance / i.getNorm() < PLAYER_RADIUS) && i.isInBounds())
97             {
98                 if (i.isOpen) return false;
99                 else return true;
100             }
101         }
102
103         for (auto i : walls)
104         {
105             double distance = fabs(Cam.x * i.a + Cam.y * i.b + Cam.z * i.c + i
                                     .d); // Distance from wall
106
107             if ((distance / i.getNorm() < PLAYER_RADIUS) && i.isInBounds())
108                 return true;
109         }
110         return false;
111     }
112
113     bool CollisionEngine::collide()
114     {
115         checkInteract();
116         return (collideWalls() || collideObjects());
117     }

```

#### 4.1.6 Console.h

```

1  #ifndef CONSOLE_H
2  #define CONSOLE_H
3
4  /*****\
5  * Connsole.h *
6  * This file was created by Jeremy Greenburg *
7  * As part of The God Core game for the University of *
8  * Tennessee at Martin's University Scholars Organization *
9  * *
10 * This file contains the declaration of the Console Class, *
11 * As well as the Trip struct for holding three integers *
12 * The Developer Console takes input from the user and *
13 * Activates various effects based upon what the user has *
14 * Typed in. *
15 \*****/
16
17
18 // To act as a circular buffer for console history
19 #include <deque>
20 // Stores actual console input
21 #include <vector>
22 // std::string

```

```

23 #include <string>
24 // For processing text
25 #include "TextEngine.h"
26
27 // Windows API
28 #include <shlobj.h>
29
30
31 // To make rgb values easier to store
32 #include "Triple.h"
33
34 class Console
35 {
36 private:
37     /***** Variables for the console itself *****/
38
39     // Triples for good color, bad color, and neutral colors
40     Triple VALID_COLOR, INVALID_COLOR, NEUTRAL_COLOR;
41     // What the console "says" (aka what appears on screen)
42     std::deque<std::string> console_log;
43     // The colors of said strings
44     std::deque<Triple> console_color;
45     // Contains the actual player input
46     std::vector<std::string> console_input;
47     // The current (finished) input being processed
48     std::string currentInput;
49     // The current (unfinished) input being type
50     std::string currentText;
51     // Console History
52     TextEngine log;
53
54     // Path to core.sav
55     char CHAR_PATH[MAX_PATH];
56     std::string SAVE_PATH;
57
58     bool isActive;
59
60     // The bottom of the console
61     const int SCREENBOTTOM = 500;
62
63     // Prints the current input and console_history
64     void printInput();
65     // Processes completed input
66     void processInput();
67
68     // Command functions
69
70     // Toggles collision on and off
71     void toggleCollision();
72
73     // Toggles godMode on and off
74     void toggleGod();
75
76     // Decrpyts the entry in core.sav
77     void decrpytSave();
78

```



```

79         // Shutdowns program
80         void halt();
81
82         // Clears the console log
83         void clear();
84
85         // Writes input to core.sav
86         void writeToSave(std::string input);
87
88         // Reads a bit from the file
89         void readFromFile(std::string input);
90
91         // Changes the currently played track
92         void playSong(std::string input);
93
94     public:
95         // Initializes VALID_COLOR, INVALID_COLOR, NEUTRAL_COLOR, and SAVE_PATH
96         Console();
97         // Manages console functions if input has been provided
98         void activate(std::string input, std::string text);
99         // Manages console function if input is still being provided
100        void activate(std::string text);
101        // Returns the console_input[count]
102        std::string getHist(int count);
103        // Returns console_input.size()
104        int getHistNum();
105
106 };
107
108 #endif

```

#### 4.1.7 Console.cpp

```

1  /*****\
2  * Console.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the Console class *
8  * For more information, see Console.cpp *
9  \*****/
10
11 // File I/O
12 #include <fstream>
13
14 // Class declaration
15 #include "Console.h"
16
17 // For saving and loading
18 #include "SaveManager.h"
19
20 // System log
21 #include "Logger.h"
22
23 // Contains global environment variables
24 #include "Globals.h"

```

```

25
26 // Return codes
27 #include "Return.h"
28
29 using namespace std;
30
31 Console::Console()
32 {
33     // Green!
34     VALID_COLOR = makeTrip(0, 1, 0);
35     // Red!
36     INVALID_COLOR = makeTrip(1, 0, 0);
37     // Gray!
38     NEUTRAL_COLOR = makeTrip(1, 1, 1);
39
40     // Get path to documents
41     HRESULT ret = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL,
42     SHGFP_TYPE_CURRENT, CHAR_PATH);
43     // Assign to SAVE_PATH
44     SAVE_PATH = CHAR_PATH;
45     // Concatenate save file
46     SAVE_PATH += "\\The God Core\\core.sav";
47 }
48 void Console::activate(string input, string text)
49 {
50     currentInput = input;
51     // This should be empty. But just incase.
52     currentText = text;
53
54     processInput();
55     printInput();
56 }
57
58 void Console::activate(string text)
59 {
60     currentText = text;
61
62     printInput();
63 }
64
65 void Console::printInput()
66 {
67     deque<string>::iterator it = console_log.begin();
68     deque<Triple>::iterator jt = console_color.begin();
69     // Iterates through the console's current log and prints it to the screen
70     for (it; it != console_log.end(); it++, jt++)
71     {
72         //                                     Index of it
73         log.printString(0, 10 + 10 * (it - console_log.begin()),
74         jt->a, jt->b, jt->c, *it);
75     }
76
77     // Prints whatever the user is typing
78     log.printString(0, SCREENBOTTOM / 2.4, 1, 1, 1, currentText);
79 }

```

```

80
81 void Console::processInput()
82 {
83     // TODO: Break this behemoth up into little, managable functions
84
85     if (currentInput == "TogClip")
86         toggleCollision();
87
88     else if (currentInput == "TogGod")
89         toggleGod();
90
91     else if (currentInput.substr(0, 5) == "Save ")
92         writeToSave(currentInput.substr(5)); // Save everything after "
93         Save "
94
95     else if (currentInput == "Decrypt")
96         decryptSave();
97
98     else if (currentInput.substr(0, 5) == "Read ")
99         readFromFile(currentInput.substr(5)); // Read everything after "
100         Read "
101
102     else if (currentInput == "Halt")
103         halt();
104
105     else if (currentInput == "Clear")
106         clear();
107
108     else if (currentInput.substr(0, 5) == "Play ")
109         playSong(currentInput.substr(5)); // Process everything after "
110         Play "
111
112     else if (currentInput == "Goto Main")
113     {
114         isInMain = true;
115         isInConsole = false;
116         HUD.toggleConsole();
117     }
118
119     // Invalid command
120     else
121     {
122         console_log.push_back("ERROR: Do not recognize \"" + currentInput
123             + "\"");
124         console_color.push_back(INVALID_COLOR);
125     }
126
127     // Clears the top of the console if too much history is added
128     if (console_log.size() > 9)
129     {
130         console_log.pop_front();
131         console_color.pop_front();
132     }
133
134     // Store the current input
135     console_input.push_back(currentInput);

```

```

132 }
133
134 void Console::writeToSave(string input)
135 {
136     // Writes whatever is in input to the save file.
137     // Probably not going to be good for loading purposes
138
139     SaveManager Jesus;
140
141     Jesus.saveLevel(input);
142
143     console_log.push_back("Saved: " + input);
144     console_color.push_back(VALID_COLOR);
145 }
146
147 void Console::readFromFile(string input)
148 {
149     // Syntax = Read core.sav
150     if (input == "core.sav")
151     {
152         ifstream infile(SAVE_PATH);
153
154         string text;
155
156         // For now, core.sav only has one line. Hopefully I'll update this
157         // when I change that
158         infile >> text;
159
160         console_log.push_back(text);
161         console_color.push_back(VALID_COLOR);
162     }
163
164     // Syntax = Read TAG FILE
165     else
166     {
167         // There should be a space seperating the file and the tag. We
168         // find that space
169         size_t pos = input.find(' ');
170
171         // If there ain't no space
172         if (pos == string::npos)
173         {
174             console_log.push_back("ERROR: No tag detected");
175             console_color.push_back(INVALID_COLOR);
176         }
177
178         // Hooray! There's a space
179         else
180         {
181             string tag = input.substr(0, pos);
182             string file = input.substr(pos + 1); // +1 to avoid the
183             // space
184
185             const char* TEXT_PATH = "Resources\\Text\\";
186             string fullPath = TEXT_PATH + file;

```

```

185         // Simply to test for the file's existence
186         ifstream infile(fullPath);
187
188         string text;
189         getline(infile, text);
190
191         // If there ain't no file
192         if (!infile)
193         {
194             console_log.push_back("ERROR: File \"" + file +
195                                   "\" not found");
196             console_color.push_back(INVALID_COLOR);
197         }
198
199         // Hooray! There's a file
200         else
201         {
202             console_log.push_back("Reading \"" + file + "\"
203                                   with tag \"" + tag + '\"');
204             console_color.push_back(VALID_COLOR);
205
206             vector<string> readText = log.getText(file, tag);
207
208             vector<string>::iterator it;
209
210             for (it = readText.begin(); it != readText.end();
211                  it++)
212             {
213                 // Push everything we found into the log
214                 console_log.push_back(*it);
215                 console_color.push_back(NEUTRAL_COLOR);
216
217                 // So we don't grow too much, keep bounds
218                 // checking
219                 if (console_log.size() > 9)
220                 {
221                     console_log.pop_front();
222                     console_color.pop_front();
223                 }
224             }
225
226             infile.close();
227         }
228     }
229
230 void Console::toggleCollision()
231 {
232     console_log.push_back("Noclip toggled.");
233     console_color.push_back(VALID_COLOR);
234
235     collision = !collision;
236 }
237
238 void Console::toggleGod()

```

```

237 {
238     console_log.push_back("God Mode toggled.");
239     console_color.push_back(VALID_COLOR);
240
241     godMode = !godMode;
242 }
243
244 void Console::decrpytSave()
245 {
246     SaveManager Jesus;
247     string sData = Jesus.loadGame();
248
249     console_log.push_back(sData);
250     console_color.push_back(VALID_COLOR);
251 }
252
253 void Console::halt()
254 {
255     Logger log;
256     log.logLine("Exiting via console");
257     exit(EXIT_OK);
258 }
259
260 void Console::clear()
261 {
262     console_log.clear();
263     console_color.clear();
264     console_input.clear();
265 }
266
267 void Console::playSong(string input)
268 {
269     int sNum = getSongNum(input);
270
271     if (sNum == -1) // Invalid input
272     {
273         console_log.push_back("ERROR: " + input + " not a valid song file
274                                ");
275         console_color.push_back(INVALID_COLOR);
276     }
277
278     else // Valid input
279     {
280         songNum = sNum;
281         changeSong = true;
282         string song = getSongName(sNum);
283         console_log.push_back("Now playing " + song);
284         console_color.push_back(VALID_COLOR);
285     }
286 }
287
288 string Console::getHist(int count)
289 {
290     int size = console_input.size();
291     if (console_input.empty())
292     {

```

```

292         return "";
293     }
294
295     // If, somehow, a fool manages to get a variable that is out of bounds
296
297     else if (count >= size)
298     {
299         return console_input.back();
300     }
301
302     else if (count < 0)
303     {
304         return console_input.front();
305     }
306
307     else
308     {
309         return console_input[size - count - 1];
310     }
311 }
312
313 int Console::getHistNum()
314 {
315     return console_input.size();
316 }

```

#### 4.1.8 Door.h

```

1  /*****\
2  * Door.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the Door class *
8  * It's mostly a fancy wrapper for a Plane with a bit *
9  * Of added functionality *
10 \*****/
11
12 #ifndef DOOR_H
13 #define DOOR_H
14
15 // Class declaration
16 #include "Plane.h"
17 // std::string
18 #include <string>
19
20 // Figure out a way to bind a controller to the door to activate it.
21 class Door
22 {
23 private:
24     // Name, so a switch can find it
25     std::string id;
26     // The physical door
27     Plane rect;
28 public:
29     // Is the door open?

```

```

30     bool isOpen;
31     // Plane's a, b, c, and d.
32     // For easier access
33     double a, b, c, d;
34
35     // Takes in the initial Plane and name
36     Door(Plane _rect, std::string _id);
37     // Calls rect.Display()
38     void Display();
39     // Returns rect.getNorm()
40     double getNorm();
41     // Returns id
42     std::string getID();
43     // Returns rect.isInBounds()
44     bool isInBounds();
45 };
46
47 #endif

```

#### 4.1.9 Door.cpp

```

1  /*****\
2  * Door.cpp                                     *
3  * This file was created by Jeremy Greenburg    *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  *                                               *
7  * This file contains the definition of the Door class. *
8  * for more information, see Door.h             *
9  \*****/
10
11 // Class declaration
12 #include "Door.h"
13
14 using namespace std;
15
16 Door::Door(Plane _rect, std::string _id) : rect(_rect), id(_id)
17 {
18     isOpen = false;
19     a = rect.a;
20     b = rect.b;
21     c = rect.c;
22     d = rect.d;
23 };
24
25 void Door::Display()
26 {
27     if (!isOpen) rect.Display();
28 }
29
30 double Door::getNorm()
31 {
32     return rect.getNorm();
33 }
34
35 string Door::getID()
36 {

```



```

37         return id;
38     }
39
40     bool Door::isInBounds()
41     {
42         return rect.isInBounds();
43     }

```

#### 4.1.10 GameManager.h

```

1  /*****\
2  * GameManager.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the GameManger class*
8  * Which oversees and manages the flow of the game *
9  \*****/
10
11 #ifndef GAMEMANAGER_H
12 #define GAMEMANAGER_H
13
14 //***** LIBRARIES AND CLASSES *****/
15
16 // For the keyboard functionality
17 #include "Keyboard.h"
18
19 // glut really wants cstdlib here
20 #include <cstdlib>
21
22 // For arrays of strings
23 #include <string>
24 #include <vector>
25
26 // OpenGL API
27 #include <GL\glut.h>
28
29 // Standard I/O for debugging
30 #include <iostream>
31
32 // To manage background music
33 #include "MusicManager.h"
34
35 // To manage saving and loading
36 #include "SaveManager.h"
37
38 class GameManager
39 {
40 private:
41     // Variables
42
43     // Objects
44     MusicManager SoundSystem;
45     Keyboard board;
46
47     // Because the main menu is dumb, we have to know when to get a click

```

```

48         bool processClick = false;
49
50         // When in the main menu, mouse coords of a click
51         int mouse_x, mouse_y;
52
53         // Functions
54
55     public:
56
57         // Captures mouse clicks
58         void mouse(int button, int state, int x, int y);
59         // Captures mouse motion
60         void motionPassive(int x, int y);
61         // CHanges window size
62         void changeSize(int w, int h);
63         // Manages scene display
64         void manageScenes();
65         // Sample drawing function
66         void draw();
67         // Normal key presses
68         void normal(unsigned char key, int x, int y);
69         // Key releases
70         void key_up(unsigned char key, int x, int y);
71         // Special keys
72         void special(int key, int x, int y);
73         // To manage playing and releasing music
74         void manageMusic();
75
76         // Wether or not core.sav exists
77         bool canContinue;
78
79     };
80
81 #endif

```

#### 4.1.11 GameManager.cpp

```

1  /*****\
2  * GameManager.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the defintion of the GameManager class.*
8  * for more information, see GameManager.h *
9  \*****/
10
11 // Class declaration
12 #include "GameManager.h"
13 // Globals
14 #include "Globals.h"
15 // Level
16 #include "Level.h"
17 // Main Menu
18 #include "MainMenu.h"
19
20 #include "Logger.h"

```

```

21
22 using namespace std;
23
24 void GameManager::mouse(int button, int state, int x, int y)
25 {
26     if (button == GLUT_RIGHT_BUTTON)
27     {
28         if (state == GLUT_DOWN)
29         {
30
31         }
32
33         else
34         {
35
36         }
37     }
38
39     else if (button == GLUT_LEFT_BUTTON)
40     {
41         if (state == GLUT_DOWN)
42         {
43             if (isPaused)
44             {
45                 isPaused = pause.getClick(x, y);
46                 bool yes = false;
47             }
48
49             else if (isInMain)
50             {
51                 mouse_x = x;
52                 mouse_y = y;
53                 processClick = true;
54             }
55
56             Logger log;
57             vector<string> output = { "X: ", to_string(x), " ", "Y:",
58                                     to_string(y) };
59             log.logLine(output);
60
61         }
62     }
63
64     else
65     {
66     }
67 }
68
69 void GameManager::motionPassive(int x, int y)
70 {
71     static int _x = 0, _y = 0;
72
73     // If nothing else is happening basically
74     if (!isPaused && !isInConsole && !isInTerminal && !isInMain)
75     {
76         if (x > _x)

```

```

76         {
77             Cam.lookRight();
78             _x = x;
79         }
80
81     else if (x < _x)
82     {
83         Cam.lookLeft();
84         _x = x;
85     }
86
87     if (y < _y)
88     {
89         Cam.lookUp();
90         _y = y;
91     }
92
93     else if (y > _y)
94     {
95         Cam.lookDown();
96         _y = y;
97     }
98
99     // Loop around to the other side of the screen
100
101     bool updateMouse = false;
102     int newY = y, newX = x;
103     if (y == 0 || y > 700)
104     {
105         updateMouse = true;
106         newY = 300;
107         _y = 300;
108     }
109
110     if (x == 0 || x > 700)
111     {
112         updateMouse = true;
113         newX = 300;
114         _x = 300;
115     }
116
117     if (updateMouse)
118     {
119         glutWarpPointer(newX, newY);
120     }
121 }
122 }
123
124 void GameManager::changeSize(int w, int h)
125 {
126     // Don't want to divide by zero
127     if (h == 0)
128         h = 1;
129
130     double ratio = w * 1.0 / h;
131

```

```

132         // Use the Projection Matrix
133         glMatrixMode(GL_PROJECTION);
134
135         // Reset Matrix
136         glLoadIdentity();
137
138         // Set the viewport to be the entire window
139         glViewport(0, 0, w, h);
140
141         // Set the correct perspective.
142         gluPerspective(45, ratio, 1, 100);
143
144         // Get Back to the Modelview
145         glMatrixMode(GL_MODELVIEW);
146     }
147
148     void GameManager::draw()
149     {
150         if (loading)
151         {
152             lvl.loadLevel(curr_level);
153
154             loading = false;
155         }
156
157         else
158         {
159             lvl.displayLevel();
160         }
161     }
162
163     void GameManager::manageScenes()
164     {
165         // If we need to change the song, we can do it here
166         if (changeSong)
167         {
168             manageMusic();
169         }
170
171         // Clears the previous drawing
172         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
173
174         if (isPaused)
175         {
176             glutSetCursor(GLUT_CURSOR_LEFT_ARROW);
177             pause.display();
178         }
179
180         else if (isInTerminal)
181         {
182             activeTerminal->DisplayScreen();
183         }
184
185         else if (isInMain)
186         {
187             // Enable using textures (pictures)

```

```

188         glutSetCursor(GLUT_CURSOR_LEFT_ARROW);
189         static MainMenu MM;
190
191         // For some reason, MM breaks horribly when it's a global or class
            member
192         // So we'll just handle mouse clicks in the display function
193         // Rather than the mouse click function
194         // Because I'm a competent programmer
195         if (processClick)
196         {
197             MM.getClick(mouse_x, mouse_y);
198             processClick = false;
199         }
200
201         MM.display();
202     }
203
204     // glutSetCursor(GLUT_CURSOR_LEFT_ARROW); Keypads maybe?
205
206     else
207     {
208         // Enable using textures (pictures)
209         glutSetCursor(GLUT_CURSOR_NONE);
210         draw();
211
212         // Moves the camera to the correct position
213         Cam.Display();
214         if (goDim)
215         {
216             HUD.goDim(30);
217             goDim = false;
218         }
219
220         else if (goDark)
221         {
222             HUD.goDark(30);
223             goDark = false;
224         }
225
226         // Prompt the user to interact if we should
227         if (interactivity) HUD.displayWarning("INTERACT");
228         else HUD.displayWarning("");
229
230         // Prints the HUD
231         HUD.DisplayHUD();
232     }
233
234     // Displays the current drawing
235     glutSwapBuffers();
236 }
237
238 void GameManager::manageMusic()
239 {
240     // All variables need to persist between frames
241     static SoundClass background;
242

```

```

243     SoundSystem.releaseSound(background);
244     changeSong = false;
245
246     // Because you can never have too much bounds checking
247     if (songNum >= 0 && songNum <= 9)
248     {
249         std::string song = getSongName(songNum);
250         SoundSystem.makeSound(&background, song.c_str());
251         SoundSystem.playSound(background);
252     }
253 }
254
255 // Normal key presses
256 void GameManager::normal(unsigned char key, int x, int y)
257 {
258     board.normal(key, x, y);
259 }
260
261 // Key releases
262 void GameManager::key_up(unsigned char key, int x, int y)
263 {
264     board.key_up(key, x, y);
265 }
266
267 // Special keys
268 void GameManager::special(int key, int x, int y)
269 {
270     board.special(key, x, y);
271 }

```

#### 4.1.12 GCTypes.h

```

1  #ifndef GC_TYPES_H
2  #define GC_TYPES_H
3
4  // Object Types
5  #define T_DOOR 0
6  #define T_TERMINAL 1
7  #define T_LEVEL_END 2
8
9  #endif

```

#### 4.1.13 Globals.h

```

1  /*****\
2  * Globals.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the Globals *
8  * All of them. *
9  * Thers a lot of them *
10 \*****/
11
12 #ifndef GLOBALS_H
13 #define GLOBALS_H

```

```

14
15 // ALLLLLLL the classes
16 #include "HeadsUpDisplay.h"
17 #include "CameraControl.h"
18 #include "PauseScreen.h"
19 #include "Level.h"
20 #include "Terminal.h"
21 #include "Door.h"
22 #include "Switch.h"
23 #include "Plane.h"
24
25 // Remember that if you're doing anything else, globals are bad.
26 // But we're in the hellscape that is graphics
27 // There are no rules here
28 // Only madness dwells here
29
30 // Typedefs make life easy
31 typedef std::vector<Plane> vr;
32 typedef std::vector<Door> vd;
33 typedef std::vector<Switch> vs;
34 typedef std::vector<Terminal> vt;
35
36 // Pointers to various interactive objects
37 extern Switch *activeSwitch;
38 extern Terminal *activeTerminal;
39
40 // Vectors containing all of the level's assets
41 extern vr walls;
42 extern vd doors;
43 extern vs switches;
44 extern vt terminals;
45
46 extern bool
47     // Are we colliding / Can we die?
48     collision, godMode,
49     // Go dim or go dark?
50     goDim, goDark,
51     // Dunno if I actually need this one
52     loading,
53     // Is in various different stages of non-normal play
54     isInConsole, isPaused, isInTerminal, isInMain,
55     // Should we change the song?
56     changeSong,
57     // Is something in interaction range?
58     interactivity;
59
60 // Number of song to change to
61 extern int songNum;
62
63
64 extern int levelNum;
65 extern std::string curr_level;
66
67 // Constant strings of the song names
68 extern const char *SONG0, *SONG1, *SONG2, *SONG3, *SONG4, *SONG5,
69                 *SONG6, *SONG7, *SONG8, *SONG9;

```



```

70
71 // Lots of global objects
72 extern HeadsUpDisplay HUD;
73 extern CameraControl Cam;
74 extern PauseScreen pause;
75 extern Level lvl;
76
77 // Converts a songname to an integer
78 int getSongNum(std::string input);
79 // Converts an integer to a songname
80 std::string getSongName(int input);
81 // Converts level_num to a string in curr_level
82 std::string getLevelString(int input);
83
84 #endif

```

#### 4.1.14 Globals.cpp

```

1  /*****\
2  * Globals.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file instantiates the global variables *
8  \*****/
9
10 #include "Globals.h"
11
12 vr walls;
13 vd doors;
14 vs switches;
15 vt terminals;
16
17 Switch *activeSwitch = NULL;
18 Terminal *activeTerminal = NULL;
19
20 bool collision = true;
21 bool godMode = false;
22 bool goDim = false;
23 bool goDark = false;
24 bool loading = true;
25 bool isInConsole = false;
26 bool isPaused = false;
27 bool isInTerminal = false;
28 bool isInMain = true;
29 bool changeSong = true;
30 bool interactivity = false;
31
32 int songNum = 0;
33
34 int levelNum = 0;
35 std::string curr_level = "LEVELZERO";
36
37 const char* SONG0 = "Dark Fog.mp3";
38 const char* SONG1 = "Mismer.mp3";
39 const char* SONG2 = "Cold Hope.mp3";

```

```

40 const char* SONG3 = "One Sly Move.mp3";
41 const char* SONG4 = "Hypnothis.mp3";
42 const char* SONG5 = "Lightless Dawn.mp3";
43 const char* SONG6 = "Spacial Harvest.mp3";
44 const char* SONG7 = "Zombie Flood.mp3";
45 const char* SONG8 = "Get on my Level.mp3";
46 const char* SONG9 = "Story of Life.mp3";
47
48 HeadsUpDisplay HUD;
49 CameraControl Cam;
50 PauseScreen pause;
51 Level lvl;
52
53 int getSongNum(std::string input)
54 {
55     if (input == SONG0 || input == "0")
56         return 0;
57     if (input == SONG1 || input == "1")
58         return 1;
59     if (input == SONG2 || input == "2")
60         return 2;
61     if (input == SONG3 || input == "3")
62         return 3;
63     if (input == SONG4 || input == "4")
64         return 4;
65     if (input == SONG5 || input == "5")
66         return 5;
67     if (input == SONG6 || input == "6")
68         return 6;
69     if (input == SONG7 || input == "7")
70         return 7;
71     if (input == SONG8 || input == "8")
72         return 8;
73     if (input == SONG9 || input == "9")
74         return 9;
75     return -1; // Invalid song
76 }
77
78 std::string getSongName(int input)
79 {
80     std::string ret;
81     switch (input)
82     {
83     case 0: ret = SONG0;
84         break;
85     case 1: ret = SONG1;
86         break;
87     case 2: ret = SONG2;
88         break;
89     case 3: ret = SONG3;
90         break;
91     case 4: ret = SONG4;
92         break;
93     case 5: ret = SONG5;
94         break;
95     case 6: ret = SONG6;

```

```

96         break;
97     case 7: ret = SONG7;
98         break;
99     case 8: ret = SONG8;
100        break;
101     case 9: ret = SONG9;
102        break;
103     default: ret = "\0";;
104        break;
105 }
106
107     return ret;
108 }
109
110 std::string getLevelString(int input)
111 {
112     std::string ret;
113     switch (input)
114     {
115     case 0: ret = "LEVELZERO";
116         break;
117     case 1: ret = "LEVELONE";
118         break;
119     case 2: ret = "LEVELTWO";
120         break;
121     case 3: ret = "LEVELTHREE";
122         break;
123     case 4: ret = "LEVELFOUR";
124         break;
125     case 5: ret = "LEVELFIVE";
126         break;
127     case 6: ret = "LEVELSIX";
128         break;
129     case 7: ret = "LEVELSEVEN";
130         break;
131     case 8: ret = "LEVELEIGHT";
132         break;
133     case 9: ret = "LEVELNINE";
134         break;
135     default: ret = "ERROR";;
136         break;
137     }
138
139     return ret;
140 }

```

#### 4.1.15 HeadsUpDisplay.h

```

1  /*****\
2  * HeadsUpDisplay.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the HeadsUpDisplay *
8  * Class, which created an Orthoganl Matrix infront of the *
9  * Screen which allows for a 2D Heads Up Display to be *

```

```

10  * Printed before the user at any time                                     *
11  * It also passes input to the developer console                         *
12  \*****/
13
14  #ifndef HEADSUPDISPLAY
15  #define HEADSUPDISPLAY
16
17  // Base class for 2D operations
18  #include "TwoD.h"
19
20  // For displaying text in the HUD
21  #include "TextEngine.h"
22  // The Developer Console
23  #include "Console.h"
24
25  class HeadsUpDisplay : public TwoD
26  {
27  private:
28      // Duration of time to dim screen (Goes from black to clear as time
        progresses)
29      int dimTime = 0;
30      // Duration of time to go dark (completely black)
31      int darkTime = 0;
32      // Wether or not to dim
33      bool dimNow = false;
34      // Wether or not to darken
35      bool darkNow = false;
36      // Wether or not we are in developer console
37      bool devConsole = false;
38
39      // Tag to current alert
40      std::string currentAlert;
41      // Text to print to the screen
42      std::string currentText;
43      // What the user is typing
44      std::string currentInput;
45
46      // To Display text
47      TextEngine helmet;
48      // Dev Console
49      Console dev;
50
51      // Draws an info bar at the top of the screen
52      void drawHelmetBounds();
53      // Displays suit alerts
54      void DisplayAlerts();
55      // Draws the Heads Up Display
56      void drawHUD();
57      // Manages the dimming of the screen
58      void dim();
59      // Manages the darkening of the screen
60      void dark();
61      // Draws the box which stores the info text
62      void drawInfoBox();
63      // Draws the developer console window
64      void drawConsole();

```

```

65         // Displays standard info in the top left corner
66         void displayInfo(char* tag);
67
68
69     public:
70         // Manages the HUD
71         void DisplayHUD();
72
73         /****          ALTERATION FUNCTIONS          *****/
74         \**** Should always be called before DisplayHud *****/
75
76         // Tells the HUD how long to dim
77         void goDim(int time);
78
79         //Tells the HUD how long to go dark
80         void goDark(int time);
81
82         // Flips dev_console
83         void toggleConsole();
84
85         // Takes in a tag to print to screen
86         void displayWarning(std::string warning);
87
88         // Takes in a string to print to screen
89         void printToConsole(std::string text);
90
91         // Signifies a completed input to the console
92         void inputString(std::string text);
93
94         // Returns an item of the console's log
95         std::string getHist(int count);
96
97         // Returns the number of items in the console's log
98         int getHistNum();
99     };
100
101 #endif

```

#### 4.1.16 HeadsUpDisplay.cpp

```

1  /*****\
2  * HeadsUpDisplay.cpp                                *
3  * This file was created by Jeremy Greenburg          *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  *                                                    *
7  * This file contains the definition of the HeadsUpDisplay *
8  * Class. For more information, see HeadsUpDisplay.h    *
9  \*****/
10
11 // Class Declaration
12 #include "HeadsUpDisplay.h"
13
14 // OpenGL API
15 #include <gl\glut.h>
16
17 // For counting seconds

```

```

18 #include <ctime>
19
20 // For displaying Planes
21 #include "Plane.h"
22
23 // For displaying triangles
24 #include "Triangle.h"
25
26 using namespace std;
27
28 void HeadsUpDisplay::drawHelmetBounds()
29 {
30     // Helmet bounds are black
31     double colors[4] = { 0, 0, 0, 1 };
32
33     // The top of the helmet
34     double top_vertices[9] =
35     {
36         SCREENRIGHT, SCREENTOP, -1,
37         SCREENLEFT, SCREENTOP, -1,
38         SCREENRIGHT / 2.0, SCREENBOTTOM / 20.0, -1
39     };
40
41     // The left of the hemlet
42     double left_vertices[9] =
43     {
44         SCREENLEFT, SCREENBOTTOM, -1,
45         SCREENLEFT, SCREENTOP, -1,
46         SCREENRIGHT / 20.0, 3 * SCREENBOTTOM / 5.0, -1
47     };
48
49     // The back of the helmet
50     double right_vertices[9] =
51     {
52         SCREENRIGHT, SCREENBOTTOM, -1,
53         SCREENRIGHT, SCREENTOP, -1,
54         19 * SCREENRIGHT / 20.0, 3 * SCREENBOTTOM / 5.0, -1
55     };
56
57     Triangle top_helm{ top_vertices, colors };
58     Triangle left_helm{ left_vertices, colors };
59     Triangle right_helm{ right_vertices, colors };
60
61     top_helm.Display2D();
62     left_helm.Display2D();
63     right_helm.Display2D();
64 }
65
66 void HeadsUpDisplay::DisplayAlerts()
67 {
68     helmet.openFile(.45 * SCREENRIGHT, .5 * SCREENBOTTOM,
69         1, 1, 1,
70         "suitAlerts.log", currentAlert);
71 }
72
73 void HeadsUpDisplay::dim()

```

```

74 {
75     static int startTime;
76     static bool timeSet = false;
77     if (dimNow)
78     {
79         if (!timeSet)
80         {
81             startTime = time(NULL);
82             timeSet = true;
83         }
84
85         int currentTime = time(NULL);
86         int timeElapsed = currentTime - startTime;
87         if (timeElapsed < dimTime)
88         {
89             // A black square that grows more transparent as time
90             // passes
91             double colors[4] = { 0, 0, 0, (double)(dimTime -
92             timeElapsed) / dimTime };
93             double dimVert[12] =
94             {
95                 SCREENLEFT, SCREENTOP, -1,
96                 SCREENLEFT, SCREENBOTTOM, -1,
97                 SCREENRIGHT, SCREENBOTTOM, -1,
98                 SCREENRIGHT, SCREENTOP, -1
99             };
100             Plane black{ dimVert, colors };
101             black.Display2D();
102         }
103         else
104         {
105             dimNow = false;
106             timeSet = false;
107         }
108     }
109 }
110
111 void HeadsUpDisplay::dark()
112 {
113     static int startTime;
114     static bool timeSet = false;
115     if (darkNow)
116     {
117         if (!timeSet)
118         {
119             startTime = time(NULL);
120             timeSet = true;
121         }
122
123         int currentTime = time(NULL);
124         int timeElapsed = currentTime - startTime;
125         if (timeElapsed < darkTime)
126         {
127             // A black square that obscures vision

```

```

128         double colors[4] = { 0, 0, 0, 1 };
129         double dimVert[12] =
130         {
131             SCREENLEFT, SCREENTOP, -1,
132             SCREENLEFT, SCREENBOTTOM, -1,
133             SCREENRIGHT, SCREENBOTTOM, -1,
134             SCREENRIGHT, SCREENTOP, -1
135         };
136
137         Plane black{ dimVert, colors };
138         black.Display2D();
139     }
140
141     else
142     {
143         darkNow = false;
144         timeSet = false;
145     }
146 }
147 }
148
149 void HeadsUpDisplay::drawConsole()
150 {
151     double colors[4] = { .1, .1, .1, .9 };
152     double vertices[12] =
153     {
154         SCREENLEFT, SCREENTOP, -1,
155         SCREENLEFT, SCREENBOTTOM / 5, -1,
156         SCREENRIGHT, SCREENBOTTOM / 5, -1,
157         SCREENRIGHT, SCREENTOP, -1
158     };
159
160     Plane console_tab{ vertices, colors };
161     console_tab.Display2D();
162
163     if (currentInput != "")
164     {
165         dev.activate(currentInput, currentText);
166         currentInput.clear();
167     }
168
169     else
170     {
171         dev.activate(currentText);
172     }
173 }
174
175 void HeadsUpDisplay::drawInfoBox()
176 {
177     double colors[4] = { 0, 1, 1, .5 };
178     double vertices[12] =
179     {
180         SCREENLEFT, SCREENTOP, -1,
181         SCREENLEFT, SCREENBOTTOM / 10, -1,
182         SCREENRIGHT / 10, SCREENBOTTOM / 10, -1,
183         SCREENRIGHT / 10, SCREENTOP, -1

```



```

184         };
185
186         Plane info{ vertices, colors };
187         info.Display2D();
188     }
189
190     void HeadsUpDisplay::displayInfo(char* tag)
191     {
192         helmet.openFile(SCREENLEFT, SCREENTOP + 20, 1, 1, 1,
193             "suitAlerts.log", "INFO-WELL");
194     }
195
196     void HeadsUpDisplay::goDim(int time)
197     {
198         dimTime = time;
199         dimNow = true;
200     }
201
202     void HeadsUpDisplay::goDark(int time)
203     {
204         darkTime = time;
205         darkNow = true;
206     }
207
208     void HeadsUpDisplay::displayWarning(std::string warning)
209     {
210         currentAlert = warning;
211     }
212
213     void HeadsUpDisplay::printToConsole(std::string text)
214     {
215         currentText = text;
216     }
217
218     void HeadsUpDisplay::inputString(std::string text)
219     {
220         currentInput = text;
221     }
222
223     void HeadsUpDisplay::toggleConsole()
224     {
225         devConsole = !devConsole;
226     }
227
228     void HeadsUpDisplay::drawHUD()
229     {
230         drawHelmetBounds();
231
232         if (dimNow)
233         {
234             dim();
235         }
236
237         else if (darkNow)
238         {
239             dark();

```

```

240     }
241
242     drawInfoBox();
243     displayInfo("SUIT-WELL");
244
245     if (devConsole)
246     {
247         drawConsole();
248     }
249
250     if (currentAlert != "")
251     {
252         DisplayAlerts();
253     }
254 }
255
256 string HeadsUpDisplay::getHist(int count)
257 {
258     return dev.getHist(count);
259 }
260
261 int HeadsUpDisplay::getHistNum()
262 {
263     return dev.getHistNum();
264 }
265
266 void HeadsUpDisplay::DisplayHUD()
267 {
268     prepare2D();
269
270     drawHUD();
271
272     prepare3D();
273 }

```

#### 4.1.17 Keyboard.h

```

1  /*****\
2  * Keyboard.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the Keyboard class, *
8  * which logs keypresses from the user and determines, *
9  * depending on the context, what action to take such. *
10 \*****/
11
12 #ifndef KEYBOARD_H
13 #define KEYBOARD_H
14
15 // std::string
16 #include <string>
17
18 class Keyboard
19 {
20 private:

```

```

21         // Signals to recieve a part of the console's history
22         bool getPrev, getNext;
23
24     public:
25         // Normal keys
26         void normal(unsigned char key, int x, int y);
27         // To read console input
28         void inputConsole(unsigned char key, int x, int y);
29         // To read terminal input
30         void inputTerminal(unsigned char key, int x, int y);
31         // To interact with the world
32         void interact(unsigned char key, int x, int y);
33         // If a key is released
34         void key_up(unsigned char key, int x, int y);
35         // Special keys (functions, arrows, ect.)
36         void special(int key, int x, int y);
37         // Manages interactivity
38         void interact();
39     };
40
41 #endif

```

#### 4.1.18 Keyboard.cpp

```

1  /*****\
2  * Keyboard.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the defintion of the Keyboard class. *
8  * for more information, see Keyboard.h *
9  \*****/
10
11 // Class decleration
12 #include "Keyboard.h"
13
14 // std::string
15 #include <string>
16
17 // glut really wants cstdlib here
18 #include <cstdlib>
19
20 // OpenGL API
21 #include <GL\glut.h>
22
23 // To recieve and manage global variables
24 #include "Globals.h"
25 // Collision detection
26 #include "CollisionEngine.h"
27
28 // Return codes
29 #include "Return.h"
30 // System log
31 #include "Logger.h"
32
33 using namespace std;

```

```

34
35 void Keyboard::normal(unsigned char key, int x, int y)
36 {
37     // If we are currently capturing input
38     if (isInConsole)
39     {
40         inputConsole(key, x, y);
41     }
42
43     // If we're in a computer
44     else if (isInTerminal)
45     {
46         inputTerminal(key, x, y);
47     }
48
49     // Otherwise (as long we aren't in a menu)
50     else if (!isPaused && !isInMain)
51     {
52         interact(key, x, y);
53     }
54
55     else
56     {
57         switch (key)
58         {
59             // Escape
60             case 27:
61                 isPaused = false;
62                 //pause.reset();
63                 break;
64         }
65     }
66
67 }
68
69 void Keyboard::inputConsole(unsigned char key, int x, int y)
70 {
71     // User string input
72     static string input;
73     // Number in console history
74     static int count = 0;
75
76     // Up arrow, recieves the next older entry in the console's history
77     if (getPrev)
78     {
79         input = HUD.getHist(count);
80
81         if (count < HUD.getHistNum() - 1)
82         {
83             count++;
84         }
85
86         getPrev = false;
87     }
88
89     // Down arrow, recieves the next newer entry in the console's history

```

```

90     else if (getNext)
91     {
92         input = HUD.getHist(count);
93
94         if (count > 0)
95         {
96             count--;
97         }
98
99         getNext = false;
100     }
101
102     // Enter key, process and clear input
103     else if (key == 13)
104     {
105         HUD.inputString(input);
106         input.clear();
107         count = 0;
108     }
109
110     // Tilda, close the console
111     else if (key == '~' || isInConsole == false)
112     {
113         input.clear();
114         isInConsole = false;
115         HUD.toggleConsole();
116         count = 0;
117     }
118
119     // Backspace. Self explanatory
120     else if (key == 8 && !input.empty())
121     {
122         input.pop_back();
123     }
124
125     // Otherwise, type normally
126     else
127     {
128         input += key;
129     }
130
131     // Print what's been typed so far
132     HUD.printToConsole(input);
133 }
134
135 // Pretty much a copy pasta of inputConsole because I'm a terrible programmer
136 // I'll try to combine em in the future, I swear
137 // Just adjust all of these to do terminally stuff I guess
138 void Keyboard::inputTerminal(unsigned char key, int x, int y)
139 {
140     // TODO: Fix terminal input with active Terminal hijibis
141
142     // User string input
143     static string input;
144     // Number in console history
145     static int count = 0;

```

```

146
147 // Up arrow, recieves the next older entry in the console's history
148 if (getPrev)
149 {
150     input = activeTerminal->getHist(count);
151
152     if (count < activeTerminal->getHistNum() - 1)
153     {
154         count++;
155     }
156
157     getPrev = false;
158 }
159
160 // Down arrow, recieves the next newer entry in the console's history
161 else if (getNext)
162 {
163     input = activeTerminal->getHist(count);
164
165     if (count > 0)
166     {
167         count--;
168     }
169
170     getNext = false;
171 }
172
173 // Enter key, process and clear input
174 else if (key == 13)
175 {
176     activeTerminal->getInput(input);
177     input.clear();
178     count = 0;
179 }
180
181 // Backspace. Self explanatory
182 else if (key == 8 && !input.empty())
183 {
184     input.pop_back();
185 }
186
187 // Otherwise, type normally
188 else
189 {
190     input += key;
191 }
192
193 // Print what's been typed so far
194 activeTerminal->getText(input); // Drawing handled elsewhere?
195 }
196
197 void Keyboard::interact(unsigned char key, int x, int y)
198 {
199     CollisionEngine col;
200     // Speed at which the player moves
201     int speedMod = 1;

```

```

202
203     int modKey = glutGetModifiers();
204
205     if (modKey == GLUT_ACTIVE_SHIFT)
206     {
207         speedMod = 2;
208     }
209
210     else
211     {
212         speedMod = 1;
213     }
214
215     switch (key)
216     {
217     case 'w':
218     case 'W':
219         Cam.moveForward(speedMod);
220         if (col.collide())
221         {
222             Cam.moveBackward(speedMod);
223         }
224         break;
225     case 'a':
226     case 'A':
227         Cam.strafeRight();
228         if (col.collide())
229         {
230             Cam.strafeLeft();
231         }
232         break;
233     case 's':
234     case 'S':
235         Cam.moveBackward(speedMod);
236         if (col.collide())
237         {
238             Cam.moveForward(speedMod);
239         }
240         break;
241     case 'd':
242     case 'D':
243         Cam.strafeLeft();
244         if (col.collide())
245         {
246             Cam.strafeRight();
247         }
248         break;
249     case 'e':
250     case 'E':
251         interact();
252         break;
253     case '~':
254         isInConsole = true;
255         HUD.toggleConsole();
256         break;
257

```

```

258         // Enter
259     case 13:
260         //goDim = true;
261         break;
262
263         // Escape
264     case 27:
265         isPaused = true;
266         break;
267     }
268 }
269
270 void Keyboard::key_up(unsigned char key, int x, int y)
271 {
272     // I'm sure I'll do something smart here
273 }
274
275 void Keyboard::special(int key, int x, int y)
276 {
277     Logger log;
278     // We start in fullscreen
279     static bool fullScreen = true;
280     switch (key)
281     {
282     case GLUT_KEY_F1:
283         fullScreen = !fullScreen;
284         break;
285
286     case GLUT_KEY_F2:
287         // Only way to exit main loop.
288         log.logLine("Exiting via F2");
289         exit(EXIT_OK);
290         break;
291
292     case GLUT_KEY_F3:
293         isInTerminal = !isInTerminal;
294         break;
295
296     case GLUT_KEY_F4:
297         isInMain = !isInMain;
298         break;
299
300     case GLUT_KEY_F5:
301         log.logCamCoords();
302         break;
303
304     case GLUT_KEY_UP:
305         if (isInConsole || isInTerminal)
306         {
307             getPrev = true;
308             getNext = false;
309
310             // To ensure that the input is updated BEFORE next key
311             // press
312             normal(0, 0, 0);
313         }
314     }
315 }

```



```

313         break;
314
315     case GLUT_KEY_DOWN:
316         if (isInConsole || isInTerminal)
317         {
318             getNext = true;
319             getPrev = false;
320
321             // To ensure that the input is updated BEFORE next key
322             // press
323             normal(0, 0, 0);
324         }
325         break;
326     }
327     if (fullScreen)
328     {
329         glutFullScreen();
330     }
331     else
332     {
333         glutReshapeWindow(1367, 767);
334         glutPositionWindow(50, 50);
335     }
336 }
337 }
338
339 void Keyboard::interact()
340 {
341     // Only do things if we actually can
342     if (interactivity)
343     {
344         if (activeSwitch != NULL) activeSwitch->toggle();
345         else if (activeTerminal != NULL)
346         {
347             isInTerminal = true;
348         }
349     }
350 }

```

#### 4.1.19 Level.h

```

1  /*****\
2  * Level.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * * *
7  * This file contains the declaration of the Level class *
8  * Which loads all level assets from a sqlite database *
9  * (data.db) *
10 \*****/
11
12 #ifndef LEVEL_H
13 #define LEVEL_H
14
15 // std::string

```

```

16 #include <string>
17 // std::vector
18 #include <vector>
19 // Planes for walls/doors/such else
20 #include "Plane.h"
21
22 // SQLite API
23 #include "sqlite3.h"
24
25 // Glut API
26 #include <GL\glut.h>
27
28 class Level
29 {
30 private:
31     // Used to load cylinders
32     GLUquadricObj *quadratic;
33     // The current level being loaded
34     std::string currLevel;
35
36     // Look, the names are self-explanatory
37     void loadWalls(sqlite3 *db);
38     void loadDoors(sqlite3 *db);
39     void loadSwitches(sqlite3 *db);
40     void loadTerminals(sqlite3 *db);
41 public:
42     // Manages the loading of the level
43     void loadLevel(std::string levelName);
44     // Draws the level
45     void displayLevel();
46 };
47
48 #endif

```

#### 4.1.20 Level.cpp

```

1  /*****\
2  * Level.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * * *
7  * This file contains the definition of the Level class. *
8  * for more information, see Keyboard.h *
9  \*****/
10
11 // Class declaration
12 #include "Level.h"
13 // To use Planes
14 #include "Plane.h"
15 // Vectors to plop stuff in
16 #include "Globals.h"
17 // Return codes
18 #include "Return.h"
19 // System log
20 #include "Logger.h"
21 // Object Types

```

```

22 #include "GCTypes.h"
23
24 using namespace std;
25
26 void Level::loadWalls(sqlite3 *db)
27 {
28     walls.clear();
29     // Prepared Statement
30     sqlite3_stmt *stm;
31     // SQL command
32     string cmd;
33     // Connection Error Test
34     int err;
35     cmd = "SELECT * FROM walls WHERE LEVEL = \"" + currLevel + "\"";
36
37     err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
38
39     if (err != SQLITE_OK)
40     {
41         Logger log;
42         vector<string> output = { "FATAL ERROR: failed to load walls from
43             ", currLevel };
44         log.logLine(output);
45         exit(STATEMENT_ERROR);
46     }
47
48     // While we still get rows of output
49     while (sqlite3_step(stm) == SQLITE_ROW)
50     {
51         double x1, x2, x3, x4,
52             y1, y2, y3, y4,
53             z1, z2, z3, z4,
54             r, g, b, a;
55         string axis;
56
57         x1 = sqlite3_column_double(stm, 2);
58         x2 = sqlite3_column_double(stm, 3);
59         x3 = sqlite3_column_double(stm, 4);
60         x4 = sqlite3_column_double(stm, 5);
61
62         y1 = sqlite3_column_double(stm, 6);
63         y2 = sqlite3_column_double(stm, 7);
64         y3 = sqlite3_column_double(stm, 8);
65         y4 = sqlite3_column_double(stm, 9);
66
67         z1 = sqlite3_column_double(stm, 10);
68         z2 = sqlite3_column_double(stm, 11);
69         z3 = sqlite3_column_double(stm, 12);
70         z4 = sqlite3_column_double(stm, 13);
71
72         r = sqlite3_column_double(stm, 14);
73         g = sqlite3_column_double(stm, 15);
74         b = sqlite3_column_double(stm, 16);
75         a = sqlite3_column_double(stm, 17);
76
77         axis = reinterpret_cast<const char*>(sqlite3_column_text(stm, 18))

```

```

77         ;
78         char ax;
79         if (axis == "x") ax = 'x';
80         else if (axis == "y") ax = 'y';
81         else if (axis == "z") ax = 'z';
82         else ax = 0;
83
84         double verts[12] =
85         {
86             x1, y1, z1,
87             x2, y2, z2,
88             x3, y3, z3,
89             x4, y4, z4
90         };
91         double colors[4] = { r, g, b, a };
92
93         Plane rect(verts, colors, ax);
94
95         walls.push_back(rect);
96     }
97
98     /*
99     Logger log;
100     vector<string> output = { "Loaded walls on", currLevel };
101     log.logLine(output);
102     */
103
104     // Deconstructs the statement
105     sqlite3_finalize(stm);
106 }
107
108 void Level::loadDoors(sqlite3 *db)
109 {
110     doors.clear();
111     // Prepared Statement
112     sqlite3_stmt *stm;
113     // SQL command
114     string cmd;
115     // Connection Error Test
116     int err;
117     cmd = "SELECT * FROM doors WHERE LEVEL = \"" + currLevel + "\"";
118
119     err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
120
121     if (err != SQLITE_OK)
122     {
123         Logger log;
124         vector<string> output = { "FATAL ERROR: Can't load doors while
125             loading", currLevel };
126         log.logLine(output);
127
128         exit(STATEMENT_ERROR);
129     }
130
131     // While we still get rows of output

```

```

131 while (sqlite3_step(stm) == SQLITE_ROW)
132 {
133     double x1, x2, x3, x4,
134           y1, y2, y3, y4,
135           z1, z2, z3, z4,
136           r, g, b, a;
137     string id;
138     string axis;
139
140     id = reinterpret_cast<const char*>(sqlite3_column_text(stm, 0));
141     x1 = sqlite3_column_double(stm, 2);
142     x2 = sqlite3_column_double(stm, 3);
143     x3 = sqlite3_column_double(stm, 4);
144     x4 = sqlite3_column_double(stm, 5);
145
146     y1 = sqlite3_column_double(stm, 6);
147     y2 = sqlite3_column_double(stm, 7);
148     y3 = sqlite3_column_double(stm, 8);
149     y4 = sqlite3_column_double(stm, 9);
150
151     z1 = sqlite3_column_double(stm, 10);
152     z2 = sqlite3_column_double(stm, 11);
153     z3 = sqlite3_column_double(stm, 12);
154     z4 = sqlite3_column_double(stm, 13);
155
156     r = sqlite3_column_double(stm, 14);
157     g = sqlite3_column_double(stm, 15);
158     b = sqlite3_column_double(stm, 16);
159     a = sqlite3_column_double(stm, 17);
160
161     a = sqlite3_column_double(stm, 17);
162
163     axis = reinterpret_cast<const char*>(sqlite3_column_text(stm, 18))
164         ;
165
166     char ax;
167     if (axis == "x") ax = 'x';
168     else if (axis == "y") ax = 'y';
169     else if (axis == "z") ax = 'z';
170     else ax = 0;
171
172     double verts[12] =
173     {
174         x1, y1, z1,
175         x2, y2, z2,
176         x3, y3, z3,
177         x4, y4, z4
178     };
179     double colors[4] = { r, g, b, a };
180
181     Plane rect(verts, colors, ax);
182     doors.push_back(Door(rect, id));
183 }
184
185 Logger log;

```

```

186         vector<string> output = { "Loaded doors on", currLevel };
187         log.logLine(output);
188
189         // Deconstructs the statement
190         sqlite3_finalize(stm);
191     }
192
193     void Level::loadSwitches(sqlite3 *db)
194     {
195         switches.clear();
196         // Prepared Statement
197         sqlite3_stmt *stm;
198         // SQL command
199         string cmd;
200         // Connection Error Test
201         int err;
202         cmd = "SELECT * FROM switches WHERE LEVEL = \"" + currLevel + "\"";
203
204         err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
205
206         if (err != SQLITE_OK)
207         {
208             Logger log;
209             vector<string> output = { "FATAL ERROR: Can't load switches while
                loading", currLevel };
210             log.logLine(output);
211
212             exit(STATEMENT_ERROR);
213         }
214
215         // While we still get rows of output
216         while (sqlite3_step(stm) == SQLITE_ROW)
217         {
218             double xt, yt, zt,
219                 xr, yr, zr;
220             string target, s_type, id;
221             int i_type;
222
223             id = reinterpret_cast<const char*>(sqlite3_column_int(stm, 0));
224             target = reinterpret_cast<const char*>(sqlite3_column_text(stm, 2)
                );
225             xt = sqlite3_column_double(stm, 3);
226             yt = sqlite3_column_double(stm, 4);
227             zt = sqlite3_column_double(stm, 5);
228
229             xr = sqlite3_column_double(stm, 6);
230             yr = sqlite3_column_double(stm, 7);
231             zr = sqlite3_column_double(stm, 8);
232
233             s_type = reinterpret_cast<const char*>(sqlite3_column_text(stm, 9)
                );
234
235             double translate[3] = { xt, yt, zt };
236             double rotate[3] = { xr, yr, zr };
237
238             if (s_type == "DOOR")

```

```

239         i_type = T_DOOR;
240     else if (s_type == "TERMINAL")
241         i_type = T_TERMINAL;
242     else if (s_type == "LEVEL_END")
243         i_type = T_LEVEL_END;
244     else
245     {
246         Logger log;
247         vector<string> output = { "Failed to evaluate string type
                entry: ", s_type, "for switch ", id };
248         log.logLine(output);
249
250         exit(DATA_ENTRY_ERROR);
251     }
252
253     switches.push_back(Switch(translate, rotate, i_type, id));
254
255     bool assigned = false;
256
257     if (s_type == "LEVEL_END")
258     {
259         assigned = true;
260
261         Logger log;
262         vector<string> output = { "Switch ", id, " bound to end
                level" };
263         log.logLine(output);
264     }
265
266     else if (s_type == "DOOR")
267     {
268         for (unsigned int i = 0; i < doors.size(); i++)
269         {
270             if (doors[i].getID() == target)
271             {
272                 Logger log;
273                 vector<string> output = { "Binding switch
                        ", id, " to door", target };
274                 log.logLine(output);
275
276                 switches[switches.size() - 1].assign(doors
                        [i]);
277
278                 assigned = true;
279             }
280         }
281     }
282
283     else if (s_type == "TERMINAL")
284     {
285         for (unsigned int i = 0; i < terminals.size(); i++)
286         {
287             if (terminals[i].getID() == target)
288             {
289                 Logger log;
290                 vector<string> output = { "Binding switch

```

```

291         ", id, " to terminal", target };
292         log.logLine(output);
293
294         switches[switches.size() - 1].assign(
295             terminals[i]);
296
297         assigned = true;
298     }
299 }
300
301 if (!assigned)
302 {
303     Logger log;
304     vector<string> output = { "Failed to bind switch ", id, "
305         to a ", s_type };
306     log.logLine(output);
307     exit(BINDING_ERROR);
308 }
309
310 Logger log;
311 vector<string> output = { "Loaded switches on", currLevel };
312 log.logLine(output);
313
314 // Deconstructs the statement
315 sqlite3_finalize(stm);
316 }
317
318 void Level::loadTerminals(sqlite3 *db)
319 {
320     terminals.clear();
321     // Prepared Statement
322     sqlite3_stmt *stm;
323     // SQL command
324     string cmd;
325     // Connection Error Test
326     int err;
327     cmd = "SELECT * FROM terminals WHERE LEVEL = \"" + currLevel + "\"";
328
329     err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
330
331     if (err != SQLITE_OK)
332     {
333         Logger log;
334         vector<string> output = { "FATAL ERROR: Can't load terminals while
335             loading", currLevel };
336         log.logLine(output);
337         exit(STATEMENT_ERROR);
338     }
339
340     // While we still get rows of output
341     while (sqlite3_step(stm) == SQLITE_ROW)
342     {

```



```

343         double xt, yt, zt,
344             xr, yr, zr;
345         string file, id;
346         id = reinterpret_cast<const char*>(sqlite3_column_text(stm, 0));
347         file = reinterpret_cast<const char*>(sqlite3_column_text(stm, 2));
348         xt = sqlite3_column_double(stm, 3);
349         yt = sqlite3_column_double(stm, 4);
350         zt = sqlite3_column_double(stm, 5);
351
352         xr = sqlite3_column_double(stm, 6);
353         yr = sqlite3_column_double(stm, 7);
354         zr = sqlite3_column_double(stm, 8);
355
356         double translate[3] = { xt, yt, zt };
357         double rotate[3] = { xr, yr, zr };
358
359         Logger log;
360         log.logLine(id);
361
362         terminals.push_back(Terminal(translate, rotate, file, id));
363     }
364
365     Logger log;
366     vector<string> output = { "Loaded terminals on", currLevel };
367     log.logLine(output);
368
369     // Deconstructs the statement
370     sqlite3_finalize(stm);
371 }
372
373 void Level::loadLevel(std::string levelName)
374 {
375     Logger log;
376     vector<string> output = { "Starting to load", levelName };
377     log.logLine(output);
378
379     if (quadratic == NULL)
380     {
381         quadratic = gluNewQuadric();
382     }
383
384     currLevel = levelName;
385
386     // Connection to SQL database
387     sqlite3 *db;
388     // 1 if error with DB
389     int connectErr = sqlite3_open("Data.db", &db);
390
391     if (connectErr != SQLITE_OK)
392     {
393         Logger log;
394         log.logLine("FATAL ERROR: Can't access database");
395
396         exit(DATABASE_ERROR);
397     }
398 }

```

```

399
400     loadWalls(db);
401     loadDoors(db);
402     loadTerminals(db);
403
404     // Loading switches must be last to properly bind targets
405     loadSwitches(db);
406
407     // Closes the database
408     sqlite3_close(db);
409
410     output[0] = "Finished loading";
411     log.logLine(output);
412
413     // Get out of wall
414     for (unsigned int i = 0; i < 3; i++)
415     {
416         Cam.moveForward(1);
417     }
418 }
419
420 void Level::displayLevel()
421 {
422     for (auto i : walls)
423     {
424         i.Display();
425     }
426
427     for (auto i : doors)
428     {
429         i.Display();
430     }
431
432     for (auto i : switches)
433     {
434         i.Display();
435     }
436
437     for (auto i : terminals)
438     {
439         i.Display();
440     }
441 }

```

#### 4.1.21 Logger.h

```

1  /*****\
2  * Logger.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * * *
7  * This file contains the declaration of the Logger class *
8  * Which writes messages to output.log because it's more *
9  * Reliable than stdout *
10 \*****/
11

```

```

12 #ifndef LOGGER_H
13 #define LOGGER_H
14
15 #include <shlobj.h>
16
17 // std::vector
18 #include <vector>
19 // std::string
20 #include <string>
21
22 class Logger
23 {
24 private:
25     // Path to the log file
26     char CHAR_PATH[MAX_PATH];
27     std::string LOG_PATH;
28
29 public:
30     Logger();
31     // Erases the log file, called at the beggining of the program
32     void nuke();
33     // Writes to the log, either multiple lines or one line
34     void logLine(std::vector<std::string> input);
35     void logLine(std::string input);
36     // Writes the Camera Coordinates to the log file
37     void logCamCoords();
38
39 };
40
41 #endif

```

#### 4.1.22 Logger.cpp

```

1  /*****\
2  * Logger.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the defintion of the Logger class. *
8  * for more information, see Logger.h *
9  \*****/
10
11 // Class declaration
12 #include "Logger.h"
13 // For Cam coords
14 #include "Globals.h"
15 // File I/O
16 #include <fstream>
17
18 #include <iostream>
19
20 using namespace std;
21
22 Logger::Logger()
23 {

```

```

24         HRESULT ret = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL,
25             SHGFP_TYPE_CURRENT, CHAR_PATH);
26         LOG_PATH = CHAR_PATH;
27         LOG_PATH += "\\The God Core\\output.log";
28     }
29     void Logger::nuke()
30     {
31         ofstream outfile(LOG_PATH); // Nukes everything within
32     }
33
34     void Logger::logLine(vector<string> input)
35     {
36         ofstream outfile(LOG_PATH, ios::app);
37
38         string output;
39
40         for (auto i : input)
41         {
42             output += i;
43             output += " ";
44         }
45         outfile << output << std::endl;
46     }
47
48     void Logger::logLine(string input)
49     {
50         ofstream outfile(LOG_PATH, ios::app);
51
52         outfile << input << std::endl;
53     }
54
55     void Logger::logCamCoords()
56     {
57         ofstream outfile(LOG_PATH, ios::app);
58
59         outfile << "Player Coordinates:\n";
60         outfile << "X: " << -Cam.x << endl;
61         outfile << "Y: " << -Cam.y << endl;
62         outfile << "Z: " << -Cam.z << endl;
63     }

```

#### 4.1.23 MainMenu.h

```

1  /*****\
2  * MainMenu.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the MainMenu class *
8  * Which uses the Simple OpenGL Interface Library to load a *
9  * png picture of the main menu, as well as provide button *
10 * Interactivity *
11 \*****/
12
13 #ifndef MAIN_MENU_H

```

```

14 #define MAIN_MENU_H
15
16 // For loading pictures
17 #include <SOIL.h>
18 // Inherit 2D functionality
19 #include "TwoD.h"
20
21 // Make OpenGL happy
22 #include <cstdlib>
23 // OpenGL API
24 #include <GL\glut.h>
25
26 class MainMenu : public TwoD
27 {
28 public:
29     // Loads the picture up in memory
30     MainMenu();
31     // Handles drawing to the screen
32     void display();
33     // Handles and processes mouse clicks
34     void getClick(double x, double y);
35
36 private:
37     // Draws the main picture
38     void drawMainPic();
39     // DEBUG: draws boxes around all buttons
40     void drawClickBoxes();
41     // What the picture is bound to
42     GLint texture;
43 };
44
45 #endif

```

#### 4.1.24 MainMenu.cpp

```

1  /*****\
2  * MainMenu.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the MainMenu class. *
8  * for more information, see MainMenu.h *
9  \*****/
10
11 // Class declaration
12 #include "MainMenu.h"
13 // isInMain
14 #include "Globals.h"
15 // Return codes
16 #include "Return.h"
17 // System log
18 #include "Logger.h"
19
20 using namespace std;
21
22 MainMenu::MainMenu()

```

```

23 {
24     texture = SOIL_load_OGL_texture
25         (
26             "Resources\\Images\\Main.png",    // Image to load
27             SOIL_LOAD_AUTO,                    //
28             ???
29             SOIL_CREATE_NEW_ID,
30             SOIL_FLAG_MIPMAPS | SOIL_FLAG_NTSC_SAFE_RGB |
31             SOIL_FLAG_COMPRESS_TO_DXT // !!!!!!!
32         );
33
34     if (texture == 0)
35     {
36         Logger log;
37         vector<string> output = {"FATAL ERROR: SOIL cannot load image",
38             SOIL_last_result()};
39         log.logLine(output);
40         exit(SOIL_ERROR);
41     }
42 }
43
44 void MainMenu::drawMainPic()
45 {
46     glEnable(GL_TEXTURE_2D);
47
48     glBindTexture(GL_TEXTURE_2D, texture); // Prepares the texture for usage
49
50     glColor3d(1, 1, 1);
51     glBegin(GL_QUADS);
52     glTexCoord2d(0, 0);    glVertex2d(SCREENLEFT, SCREENTOP);
53     glTexCoord2d(0, 1);    glVertex2d(SCREENLEFT, SCREENBOTTOM);
54     glTexCoord2d(1, 1);    glVertex2d(SCREENRIGHT, SCREENBOTTOM);
55     glTexCoord2d(1, 0);    glVertex2d(SCREENRIGHT, SCREENTOP);
56
57     glEnd();
58
59     glDisable(GL_TEXTURE_2D);
60 }
61
62 void MainMenu::drawClickBoxes()
63 {
64     glColor3d(1, 0, 0);
65
66     // Start a new game
67     glBegin(GL_LINE_LOOP);
68     glVertex2d(SCREENRIGHT / 20.0, SCREENBOTTOM / 2.2);
69     glVertex2d(SCREENRIGHT / 20.0, SCREENBOTTOM / 1.9);
70     glVertex2d(SCREENRIGHT / 3.0, SCREENBOTTOM / 1.9);
71     glVertex2d(SCREENRIGHT / 3.0, SCREENBOTTOM / 2.2);
72     glEnd();
73
74     // Load game
75     glBegin(GL_LINE_LOOP);
76     glVertex2d(SCREENRIGHT / 10.0, SCREENBOTTOM / 1.57);
77     glVertex2d(SCREENRIGHT / 10.0, SCREENBOTTOM / 1.75);

```

```

76     glVertex2d(SCREENRIGHT / 3.5, SCREENBOTTOM / 1.75);
77     glVertex2d(SCREENRIGHT / 3.5, SCREENBOTTOM / 1.57);
78     glEnd();
79
80     // Options
81     glBegin(GL_LINE_LOOP);
82     glVertex2d(SCREENRIGHT / 8.5, SCREENBOTTOM / 1.35);
83     glVertex2d(SCREENRIGHT / 8.5, SCREENBOTTOM / 1.45);
84     glVertex2d(SCREENRIGHT / 3.9, SCREENBOTTOM / 1.45);
85     glVertex2d(SCREENRIGHT / 3.9, SCREENBOTTOM / 1.35);
86     glEnd();
87
88     // Options
89     glBegin(GL_LINE_LOOP);
90     glVertex2d(SCREENRIGHT / 8.5, SCREENBOTTOM / 1.35);
91     glVertex2d(SCREENRIGHT / 8.5, SCREENBOTTOM / 1.45);
92     glVertex2d(SCREENRIGHT / 3.9, SCREENBOTTOM / 1.45);
93     glVertex2d(SCREENRIGHT / 3.9, SCREENBOTTOM / 1.35);
94     glEnd();
95 }
96
97 void MainMenu::getClick(double x, double y)
98 {
99     // Start new game
100    if (x >= SCREENRIGHT / 20.0 && x <= SCREENRIGHT / 3.0)
101    {
102        if (y >= SCREENBOTTOM / 2.2 && y <= SCREENBOTTOM / 1.9)
103        {
104            isInMain = false;
105        }
106    }
107
108    // Load Game
109    if (x >= SCREENRIGHT / 10.0 && x <= SCREENRIGHT / 3.5)
110    {
111        if (y >= SCREENBOTTOM / 1.75 && y <= SCREENBOTTOM / 1.57)
112        {
113            isInMain = false;
114        }
115    }
116
117    // Options
118    if (x >= SCREENRIGHT / 8.5 && x <= SCREENRIGHT / 3.9)
119    {
120        if (y >= SCREENBOTTOM / 1.45 && y <= SCREENBOTTOM / 1.35)
121        {
122            //
123        }
124    }
125
126    // Exit
127    if (x >= SCREENRIGHT / 20.0 && x <= SCREENRIGHT / 3.0)
128    {
129        if (y >= SCREENBOTTOM / 2.2 && y <= SCREENBOTTOM / 1.9)
130        {
131            exit(0);

```

```

132         }
133     }
134 }
135
136 void MainMenu::display()
137 {
138     prepare2D();
139
140     drawMainPic();
141
142     // Disable once finished
143     drawClickBoxes();
144
145     glEnd();
146
147     prepare3D();
148 }

```

#### 4.1.25 MusicManager.h

```

1  /*****\
2  * MusicManager.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the MusicManager *
8  * Class, which uses the FMOD API to load .mp3 files into *
9  * Memory, play them when called, and release the memory *
10 * When the song is no longer needed. *
11 \*****/
12
13 #ifndef MUSICMANAGER_H
14 #define MUSICMANAGER_H
15
16 // FMOD API
17 #include <fmod.hpp>
18
19 // Creates new type for ease of use
20 typedef FMOD::Sound* SoundClass;
21
22 class MusicManager
23 {
24 private:
25     // Pointer to dynamic system memory to load music
26     FMOD::System *m_pSystem;
27
28     // The path to the music folder
29     static const char* MUSIC_PATH;
30
31 public:
32     // Loads the song in memory
33     void makeSound(SoundClass *psound, const char *song);
34     // Plays the song (Always loops)
35     void playSound(SoundClass pSound, bool bLoop = true);
36     // Releases the song
37     void releaseSound(SoundClass psound);

```



```

38         // Initializes FMOD
39         MusicManager();
40     };
41
42 #endif

```

#### 4.1.26 MusicManager.cpp

```

1  /*****\
2  * FILENAME *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the MusicManager *
8  * Class. For more information, see MusicManager.h *
9  \*****/
10
11 // Class definition
12 #include "MusicManager.h"
13
14 // Because concatenating char*'s are really hard
15 #include <string>
16
17 // Return codes
18 #include "Return.h"
19
20 // System log
21 #include "Logger.h"
22
23 using namespace std;
24
25 // Initialize the constant member of the class
26 const char* MusicManager::MUSIC_PATH = "Resources\\Music\\";
27
28 MusicManager::MusicManager()
29 {
30     Logger log;
31     if (FMOD::System_Create(&m_pSystem) != FMOD_OK)
32     {
33         log.logLine("FATAL ERROR: FMOD unable to create system");
34         exit(FMOD_ERROR);
35     }
36
37     int driverCount = 0;
38     m_pSystem->getNumDrivers(&driverCount);
39
40     // If you have no driver, you have bigger problems to worry about
41     if (driverCount == 0)
42     {
43         // Report Error
44         log.logLine("ERROR: FMOD unable to detect drivers");
45         exit(FMOD_ERROR);
46     }
47
48     log.logLine("FMOD succesfully initialized");
49     // Initialize our Instance with 36 Channels

```

```

50         m_pSystem->init(36, FMOD_INIT_NORMAL, NULL);
51     }
52
53     void MusicManager::makeSound(SoundClass *psound, const char *song)
54     {
55         // MUSIC_PATH is placed in a nice string. Good string. Strings are friends
56         string fullPath = MUSIC_PATH;
57         // Now there is a full path to the song
58         fullPath += song;
59
60         m_pSystem->createSound(fullPath.c_str(), FMOD_DEFAULT, 0, psound);
61     }
62
63     void MusicManager::playSound(SoundClass pSound, bool bLoop)
64     {
65         if (!bLoop)
66             pSound->setMode(FMOD_LOOP_OFF);
67         else
68         {
69             pSound->setMode(FMOD_LOOP_NORMAL);
70             pSound->setLoopCount(-1);
71         }
72
73         m_pSystem->playSound(pSound, NULL, false, 0);
74     }
75
76     void MusicManager::releaseSound(SoundClass pSound)
77     {
78         pSound->release();
79     }

```

#### 4.1.27 PauseScreen.h

```

1  /*****\
2  * PauseScreen.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the PauseScreen *
8  * class, which contains the rules for drawing the Pause *
9  * Screen, as well as mechanics for detecting button clicks *
10 * and rules for when each button is clicked. *
11 * *
12 * The PauseScreen class is inherited from the Screen class *
13 * to take advantage of it's native drawing functions as well *
14 * as its native variables, but redefines the getClick *
15 * function to allow for PauseScreen's differing mechanics *
16 \*****/
17
18 #ifndef PAUSESCREEN_H
19 #define PAUSESCREEN_H
20
21 // 2D functionallity
22 #include "TwoD.h"
23 // std::string
24 #include <string>

```

```

25 // std::vector
26 #include <vector>
27
28 class PauseScreen : public TwoD
29 {
30 private:
31     int num_of_buttons, activeButton;
32     std::vector <std::string> buttonNames;
33
34
35 public:
36     // Initializes variables
37     PauseScreen();
38
39     // Displays the pause screen
40     void display();
41     /*
42      * Detects where the player clicks on the screen and responds accordingly.
43      * Returns false if the player clicks the exit button (indicating that the
44      * screen should close)
45      * Returns true otherwise (indicating that the screen should remain open
46      */
47     bool getClick(int x, int y);
48
49     // Performs an action depending on which button has been clicked
50     void doStuff();
51 };
52 #endif

```

#### 4.1.28 PauseScreen.cpp

```

1  /*****\
2  * PauseScreen.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the PauseScreen class*
8  * For more information, see PauseScreen.h *
9  \*****/
10
11 // Class declaration
12 #include "PauseScreen.h"
13
14 // SaveManager class
15 #include "SaveManager.h"
16
17 // Global variables
18 #include "Globals.h"
19
20 // Return codes
21 #include "Return.h"
22
23 PauseScreen::PauseScreen()
24 {
25     num_of_buttons = 4;

```

```

26         activeButton = -1;
27
28         buttonNames.push_back("Inventory");
29         buttonNames.push_back("Save");
30         buttonNames.push_back("Load");
31         buttonNames.push_back("Quit");
32     }
33
34
35     bool PauseScreen::getClick(int x, int y)
36     {
37         // The left and right bounds of a button
38         if (x > SCREENLEFT + 20 &&
39             x < SCREENRIGHT / 10)
40         {
41             for (int i = 0; i < num_of_buttons; i++)
42             {
43                 // If y is in the particular bounds of a button
44                 if (y > SCREENBOTTOM / num_of_buttons * (i + .1)
45                     &&
46                     y < SCREENBOTTOM / num_of_buttons * (i + 1))
47                 {
48                     if (activeButton == i)
49                         activeButton = -1;
50                     else
51                         activeButton = i;
52                 }
53             }
54         }
55
56         else if (
57             // The bounds of the exit button
58             x > 19 * SCREENRIGHT / 20 && y < SCREENBOTTOM / 20
59             )
60         {
61             // Exit button, close window
62             return false;
63         }
64
65         // Not exit button, keep window
66         return true;
67     }
68
69     void PauseScreen::doStuff()
70     {
71         // Inventory
72         if (activeButton == 0)
73         {
74             // Inventory here
75         }
76
77         // Save
78         else if (activeButton == 1)
79         {
80             //SaveManager Jesus; // Jesus saves
81             //Jesus.saveLevel(curr_level);

```

```

82     }
83
84     // Load
85     else if (activeButton == 2)
86     {
87         //SaveManager Jesus; // Jesus... loads?
88         loading = true;
89
90         //curr_level = Jesus.loadGame();
91     }
92
93     // Quit
94     else if (activeButton == 3)
95     {
96         exit(EXIT_OK);
97     }
98 }
99
100 void PauseScreen::display()
101 {
102     prepare2D();
103
104     // We're gonna have specialized actions for this main menu
105     //drawExit();
106     //drawSideBar();
107     //drawButtons();
108     doStuff();
109
110     prepare3D();
111 }

```

#### 4.1.29 Plane.h

```

1  /*****\
2  * Plane.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the Plane class *
8  * Which is used to hold the details of a 2D Plane and *
9  * draw it to the screen *
10 \*****/
11
12 #ifndef Plane_H
13 #define Plane_H
14
15 class Plane
16 {
17 private:
18     // Arrays containing the color and vertices of the Plane
19     double color[4];
20     // What axis is it aligned on (x y z)
21     char axis;
22     // The vertices of the corners
23     double vertices[12];
24 public:

```

```

25
26     // Paramaterized constructor, as there cannot be a Plane without vertices
27     // Can take an axis or can ignore axis
28     Plane(const double(&new_vertices)[12], const double(&new_color)[4], char
        _axis);
29     Plane(const double(&new_vertices)[12], const double(&new_color)[4]);
30
31     // Part of the plane equation, calculated in constructor
32     double a, b, c, d;
33
34     // Determines if the player is in the bounds of the Plane (based on axis)
35     bool isInBounds();
36
37     // Returns the plane norm (Perpendicular line)
38     double getNorm();
39
40     // Print a Plane in 3D
41     void Display();
42     // Print a Plane in 2D
43     void Display2D();
44 };
45
46 #endif

```

#### 4.1.30 Plane.cpp

```

1  /*****\
2  * Plane.cpp                                     *
3  * This file was created by Jeremy Greenburg      *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  *                                                 *
7  * This file contains the definition of the Plane class *
8  * For more information, see Plane.h               *
9  \*****/
10
11 #include "Plane.h"
12
13 // For std::copy
14 #include <iterator>
15 #include <utility>
16
17 // max and min
18 #include <algorithm>
19
20 // OpenGL API
21 #include <GL\glut.h>
22
23 // For Cam coords
24 #include "Globals.h"
25
26 using namespace std;
27
28 Plane::Plane(const double (&new_vertices)[12], const double (&new_color)[4], char
    _axis)
29 {
30     // Copies the color

```

```

31     copy(begin(new_color), end(new_color), color);
32
33     // Copies the vertices
34     copy(begin(new_vertices), end(new_vertices), vertices);
35
36
37     // Somedays I wonder what I'm even doing \\
38     // When I forget what all this means: http://keisan.casio.com/exec/system
39         /1223596129 \\
40
41     // Calculate vector equation  $ax + by + cz + d = 0$ 
42     // Get two vectors from three of the corners
43     double AB[] = { vertices[3] - vertices[0], vertices[4] - vertices[1],
44                     vertices[5] - vertices[2] };
45     double AC[] = { vertices[6] - vertices[0], vertices[7] - vertices[1],
46                     vertices[8] - vertices[2] };
47     // Cross Product of AB and AC
48     a = (AB[1] * AC[2]) - (AB[2] * AC[1]);
49     b = (AB[2] * AC[0]) - (AB[0] * AC[2]);
50     c = (AB[0] * AC[1]) - (AB[1] * AC[0]);
51     d = (a * vertices[0] + b * vertices[1] + c * vertices[2]);
52
53     axis = _axis;
54 }
55
56 Plane::Plane(const double(&new_vertices)[12], const double(&new_color)[4])
57 {
58     // Copies the color
59     copy(begin(new_color), end(new_color), color);
60
61     // Copies the vertices
62     copy(begin(new_vertices), end(new_vertices), vertices);
63
64
65     // Somedays I wonder what I'm even doing \\
66     // When I forget what all this means: http://keisan.casio.com/exec
67         /system/1223596129 \\
68
69     // Calculate vector equation  $ax + by + cz + d = 0$ 
70     // Get two vectors from three of the corners
71     double AB[] = { vertices[3] - vertices[0], vertices[4] - vertices[1],
72                     vertices[5] - vertices[2] };
73     double AC[] = { vertices[6] - vertices[0], vertices[7] - vertices[1],
74                     vertices[8] - vertices[2] };
75     // Cross Product of AB and AC
76     a = (AB[1] * AC[2]) - (AB[2] * AC[1]);
77     b = (AB[2] * AC[0]) - (AB[0] * AC[2]);
78     c = (AB[0] * AC[1]) - (AB[1] * AC[0]);
79     d = (a * vertices[0] + b * vertices[1] + c * vertices[2]);
80
81     axis = 0;
82 }
83
84 void Plane::Display()
85 {
86     // Set's OpenGL's color to the color of the Plane

```

```

81         glColor4f(color[0], color[1], color[2], color[3]);
82
83         glBegin(GL_QUADS);
84         glVertex3d(vertices[0], vertices[1], vertices[2]);
85         glVertex3d(vertices[3], vertices[4], vertices[5]);
86         glVertex3d(vertices[6], vertices[7], vertices[8]);
87         glVertex3d(vertices[9], vertices[10], vertices[11]);
88         glEnd();
89     }
90
91     void Plane::Display2D()
92     {
93         glColor4f(color[0], color[1], color[2], color[3]);
94
95         glBegin(GL_QUADS);
96         glVertex2d(vertices[0], vertices[1]);
97         glVertex2d(vertices[3], vertices[4]);
98         glVertex2d(vertices[6], vertices[7]);
99         glVertex2d(vertices[9], vertices[10]);
100        glEnd();
101    }
102
103     bool Plane::isInBounds()
104     {
105         if (axis == 'x')
106         {
107             vector<double> X = { vertices[0], vertices[3], vertices[6],
108                                 vertices[9] };
109             double maxX = *max_element(X.begin(), X.end());
110             double minX = *min_element(X.begin(), X.end());
111
112             return (-Cam.x <= maxX && -Cam.x >= minX);
113         }
114
115         else if (axis == 'y')
116         {
117             vector<double> Y = { vertices[1], vertices[4], vertices[7],
118                                 vertices[10] };
119             double maxY = *max_element(Y.begin(), Y.end());
120             double minY = *min_element(Y.begin(), Y.end());
121
122             return (-Cam.y <= maxY && -Cam.x >= minY);
123         }
124
125         else if (axis == 'z')
126         {
127             vector<double> Z = { vertices[2], vertices[5], vertices[8],
128                                 vertices[11] };
129             double maxZ = *max_element(Z.begin(), Z.end());
130             double minZ = *min_element(Z.begin(), Z.end());
131
132             return (-Cam.z <= maxZ && -Cam.z >= minZ);
133         }
134         else return false;
135     }

```



```

134
135 double Plane::getNorm()
136 {
137     return sqrt(a * a + b * b + c * c);
138 }

```

#### 4.1.31 Return.h

```

1  /*****\
2  * Return.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains various return codes for when things *
8  * Go horribly wrong (and they do) *
9  * (just hopefully not during my senior defense) *
10 \*****/
11
12 #ifndef RETURN_H
13 #define RETURN_H
14
15 #define EXIT_OK 0
16 #define EXIT_EARLY 1 // If we exit OpenGL main loop early
17 #define FMOD_ERROR 2 // Fmod can't load sound
18 #define DATABASE_ERROR 3 // sqlite can't load database
19 #define STATEMENT_ERROR 4 // sqlite statement fails to execute
20 #define SOIL_ERROR 5 // SOIL fails to load image
21 #define DATA_ENTRY_ERROR 6
22 #define BINDING_ERROR 7
23
24 #endif

```

#### 4.1.32 Resource.h

```

1  /*****\
2  * Return.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains various return codes for when things *
8  * Go horribly wrong (and they do) *
9  * (just hopefully not during my senior defense) *
10 \*****/
11
12 #ifndef RETURN_H
13 #define RETURN_H
14
15 #define EXIT_OK 0
16 #define EXIT_EARLY 1 // If we exit OpenGL main loop early
17 #define FMOD_ERROR 2 // Fmod can't load sound
18 #define DATABASE_ERROR 3 // sqlite can't load database
19 #define STATEMENT_ERROR 4 // sqlite statement fails to execute
20 #define SOIL_ERROR 5 // SOIL fails to load image
21 #define DATA_ENTRY_ERROR 6
22 #define BINDING_ERROR 7

```

```

23
24 #endif

```

#### 4.1.33 SaveManager.h

```

1  /*****\
2  * SaveManager.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the SaveManager *
8  * Class, which saves data by encrypting an array of strings *
9  * And writing them to core.sav, or by reading in an array of *
10 * Strings from core.sav and decrypting them *
11 \*****/
12
13 #ifndef SAVEMANAGER_H
14 #define SAVEMANAGER_H
15
16 // Windows API
17 #include <shlobj.h>
18
19 // Because concatenating char*'s is really hard
20 #include <string>
21
22 class SaveManager
23 {
24 private:
25     // The path to core.sav
26     char CHAR_PATH[MAX_PATH];
27     std::string SAVE_PATH;
28
29     // Takes an unencrypted string and returns an encrypted string
30     std::string encryptData(std::string data);
31     // Takes an encrypted string and returns a decrypted string
32     std::string decryptData(std::string data);
33 public:
34     SaveManager();
35     // Writes the array of encrypted strings to core.sav
36     void saveLevel(std::string);
37     // Reads in an array of encrypted strings from core.sav and decrypts them
38     std::string loadGame();
39     // Returns true if core.save exists
40     bool checkSave();
41 };
42
43 #endif

```

#### 4.1.34 SaveManager.cpp

```

1  /*****\
2  * SaveManager.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *

```

```

7  * This file contains the definition of the SaveManager class*
8  * For more information, see SaveManager.h *
9  \*****/
10
11 // Class definition
12 #include "SaveManager.h"
13
14 // File I/O
15 #include <fstream>
16
17 using namespace std;
18
19 SaveManager::SaveManager()
20 {
21     HRESULT ret = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL,
22     SHGFP_TYPE_CURRENT, CHAR_PATH);
23     SAVE_PATH = CHAR_PATH;
24     SAVE_PATH += "\\The God Core\\core.sav";
25 }
26
27 string SaveManager::encryptData(string data)
28 {
29     string ret_str;
30     for (unsigned int i = 0; i < data.length()*3; i+=3)
31     {
32         ret_str += data[i/3] + 48;
33         ret_str += data[i/3] - 48;
34         ret_str += data[i/3] + 53;
35     }
36     return ret_str;
37 }
38
39 string SaveManager::decryptData(string data)
40 {
41     string ret_str;
42     for (unsigned int i = 0; i < data.length(); i+=3)
43     {
44         ret_str += data[i] - 48;
45     }
46     return ret_str;
47 }
48
49 void SaveManager::saveLevel(string curr_level)
50 {
51     ofstream save(SAVE_PATH);
52
53     string encr_str = encryptData(curr_level);
54
55     save << encr_str;
56
57     save.close();
58 }
59
60
61 string SaveManager::loadGame()

```

```

62 {
63     ifstream save(SAVE_PATH);
64
65     string test;
66     string dcr_str;
67     save >> test;
68     dcr_str = decryptData(test);
69
70     save.close();
71
72     return dcr_str;
73 }
74
75 bool SaveManager::checkSave()
76 {
77     ifstream save(SAVE_PATH);
78
79     if (save)
80     {
81         return true;
82     }
83
84     else
85     {
86         return false;
87     }
88 }

```

#### 4.1.35 Switch.h

```

1  /*****\
2  * Switch.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the Switch class *
8  * Which is bound to a Door via pointer and can open and *
9  * Close the door at will *
10 \*****/
11
12 #ifndef SWITCH_H
13 #define SWITCH_H
14
15 // Door class
16 #include "Door.h"
17 // Terminal Class
18 #include "Terminal.h"
19
20 // Types
21 #include "GCTypes.h"
22
23 class Switch
24 {
25 private:
26     void* target; // The door that this switch activates
27     // Translation and rotation coordinates

```

```

28         double translate[3], rotate[3];
29
30         // One of the predefined types
31         int targetType;
32
33         std::string id;
34
35     public:
36         // Initializes the translation and rotation matrices
37         Switch(const double(&_translate)[3], const double(&_rotate)[3], int _type,
38             std::string _id);
39         // Binds the target pointer to a door
40         void assign(Door &_target);
41         void assign(Terminal &_target);
42         // Opens/Closes the door
43         void toggle();
44         // Actually draws the switch
45         void Display();
46
47         // Gets the translation coordinates
48         double getX();
49         double getY();
50         double getZ();
51     };
52 #endif

```

#### 4.1.36 Switch.cpp

```

1  /*****\
2  * Switch.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the Switch class *
8  * For more information, see CameraControl.h *
9  \*****/
10
11 // Class declaration
12 #include "Switch.h"
13
14 // Allows copying arrays
15 #include <iterator>
16 #include <utility>
17 #include <algorithm>
18
19 #include "Globals.h"
20
21 // OpenGL API
22 #include <GL\glut.h>
23
24 using namespace std;
25
26 Switch::Switch(const double(&_translate)[3], const double(&_rotate)[3], int _type,
27     string _id)

```

```

28         // Copies the color
29         copy(begin(_translate), end(_translate), translate);
30
31         // Copies the vertices
32         copy(begin(_rotate), end(_rotate), rotate);
33
34         targetType = _type;
35
36         target = NULL;
37
38         _id = id;
39     }
40
41     void Switch::assign(Door &_target)
42     {
43         target = &_target;
44     }
45
46     void Switch::assign(Terminal &_target)
47     {
48         target = &_target;
49     }
50
51     void Switch::toggle()
52     {
53         switch (targetType)
54         {
55             case T_DOOR:
56             {
57                 Door* t = (Door*)target;
58                 t->isOpen = !t->isOpen;
59                 break;
60             }
61             case T_TERMINAL:
62             {
63                 Terminal* t = (Terminal*)target;
64                 t->isOn = !t->isOn;
65                 break;
66             }
67             case T_LEVEL_END:
68             {
69                 levelNum++;
70                 curr_level = getLevelString(levelNum);
71                 loading = true;
72
73                 // TEMP
74                 songNum++;
75                 changeSong = true;
76             }
77         }
78     }
79
80     void Switch::Display()
81     {
82         // THOUGHT: Have a different switch display for power?
83

```

```

84     glPushMatrix();
85     glTranslated(translate[0], translate[1], translate[2]);
86     glRotated(rotate[0], 1, 0, 0);
87     glRotated(rotate[1], 0, 1, 0);
88     glRotated(rotate[2], 0, 0, 1);
89
90     glColor3d(0.9, 0.9, 0.9);
91     glutSolidCube(.1);
92
93     switch (targetType)
94     {
95     case T_DOOR:
96         glColor3d(0, 1, 0);
97         break;
98     case T_TERMINAL:
99         glColor3d(1, 0, 0);
100        break;
101    default:
102        glColor3d(0, 0, 1);
103    }
104
105    glScaled(.5, .5, 1.5);
106    glutSolidCube(.1);
107
108    glPopMatrix();
109 }
110
111 double Switch::getX()
112 {
113     return translate[0];
114 }
115
116 double Switch::getY()
117 {
118     return translate[1];
119 }
120
121 double Switch::getZ()
122 {
123     return translate[2];
124 }

```

#### 4.1.37 Terminal.h

```

1  /*****\
2  * Terminal.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the Terminal class *
8  * Which draws and manages ingame computer terminals *
9  * And has nothing to do with terminal illness I swear *
10 \*****/
11
12 #ifndef TERMINAL_H
13 #define TERMINAL_H

```

```

14
15 #include "TwoD.h" // To inherit 2D class
16
17 #include <cstdlib>
18
19 // For loading pictures
20 #include <SOIL.h>
21
22 #include "TextEngine.h" // To display text to screen
23
24 #include <string>
25
26 #include <GL\glut.h>
27
28 class Terminal : public TwoD // Inherit 2D functionality
29 {
30 private:
31     // text = what the user is typing, input = completed input
32     std::string currentInput, currentText, error, file;
33     std::vector<std::string> history, prompts, content;
34     std::string id;
35     // Where to print each item
36     const double INPUT_LINE = SCREENBOTTOM / 7.0;
37     const double ERROR_LINE = INPUT_LINE - 30;
38     const double PROMPT_START = INPUT_LINE + 30;
39     const double CONTENT_START = PROMPT_START + 100;
40
41     GLint bTexture;
42
43     int num;
44     // Print our text
45     TextEngine text;
46
47     // Translation and rotation matrices
48     double translate[3], rotate[3];
49
50     // Draws the actual terminal
51     void draw();
52
53     void processInput();
54
55     void parseFile();
56
57     static const char* TERM_PATH;
58
59 public:
60     // Draws the 3D object in the world
61     void Display();
62     // Draws the 2D Terminal screen
63     void DisplayScreen();
64     // Shows the currently typed string
65     void getText(std::string text);
66     // Signifies a completed string to process
67     void getInput(std::string text);
68     // Returns an item in the terminal's log
69     std::string getHist(int count);

```



```

70         // Returns the number of items in the terminal's log
71         int getHistNum();
72
73         // Gets the translation coordinates
74         double getX();
75         double getY();
76         double getZ();
77
78         std::string getID();
79
80         bool isOn = true;
81
82         Terminal(const double(&_translate)[3], const double(&_rotate)[3], std::
            string _file, std::string _id);
83
84     };
85
86 #endif

```

#### 4.1.38 Terminal.cpp

```

1  /*****\
2  * Terminal.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the Terminal class *
8  * For more information, see CameraControl.h *
9  \*****/
10
11 //
12 // Class declaration
13 #include "Terminal.h"
14
15 // Planes
16 #include "Plane.h"
17
18 // For system logging
19 #include "Logger.h"
20
21 // Return codes
22 #include "Return.h"
23
24 // Global variables
25 #include "Globals.h"
26
27 // File I/O
28 #include <fstream>
29
30 using namespace std;
31
32 const char* Terminal::TERM_PATH = "Resources\\Text\\";
33
34 void Terminal::getText(std::string text)
35 {
36     currentText = text;

```

```

37 }
38
39 void Terminal::getInput(std::string text)
40 {
41     currentInput = text;
42 }
43
44 string Terminal::getHist(int count)
45 {
46     int size = history.size();
47     if (history.empty())
48     {
49         return "";
50     }
51
52     // If, somehow, a fool manages to get a variable that is out of bounds
53
54     else if (count >= size)
55     {
56         return history.back();
57     }
58
59     else if (count < 0)
60     {
61         return history.front();
62     }
63
64     else
65     {
66         return history[size - count - 1];
67     }
68 }
69
70 int Terminal::getHistNum()
71 {
72     return history.size();
73 }
74
75 void Terminal::draw()
76 {
77     // Completely black background
78     double colors[4] = { 0, 0, 0, 1 };
79     double vertices[12] =
80     {
81         SCREENLEFT, SCREENTOP, -1,
82         SCREENLEFT, SCREENBOTTOM, -1,
83         SCREENRIGHT, SCREENBOTTOM, -1,
84         SCREENRIGHT, SCREENTOP, -1
85     };
86
87     Plane background{ vertices, colors };
88     background.Display2D();
89
90
91     // Gotta do the banner manually
92     glEnable(GL_TEXTURE_2D);

```

```

93
94     glBindTexture(GL_TEXTURE_2D, bTexture); // Prepares the texture for usage
95
96     glColor3d(1, 1, 1);
97     glBegin(GL_QUADS);
98     glTexCoord2d(0, 0);      glVertex2d(SCREENLEFT, SCREENTOP);
99     glTexCoord2d(0, 1); glVertex2d(SCREENLEFT, SCREENBOTTOM / 9.0);
100    glTexCoord2d(1, 1); glVertex2d(SCREENRIGHT, SCREENBOTTOM / 9.0);
101    glTexCoord2d(1, 0);      glVertex2d(SCREENRIGHT, SCREENTOP);
102
103    glEnd();
104
105    glDisable(GL_TEXTURE_2D);
106 }
107
108 void Terminal::DisplayScreen()
109 {
110     prepare2D();
111
112     draw();
113
114     // If we need to proces a command
115     if (currentInput != "")
116     {
117         processInput();
118
119         history.push_back(currentInput);
120
121         currentInput.clear();
122     }
123
124     else
125     {
126         // Print all prompts
127         for (unsigned int i = 0; i < prompts.size(); i++)
128         {
129             text.printString(SCREENLEFT, PROMPT_START + 15 * i, 0, 1,
130                             0, prompts[i]);
131         }
132
133         // Print an error
134         text.printString(SCREENLEFT, ERROR_LINE, 1, 0, 0, error);
135         // Echo user text
136         text.printString(SCREENLEFT, INPUT_LINE, 0, 1, 0, ":> " +
137                         currentText);
138
139         // If needed, print content
140         if (num != -1 && num < (signed int)content.size())
141         {
142             text.openFile(SCREENLEFT, CONTENT_START, 0, 1, 0, file,
143                           content[num]);
144         }
145     }
146
147     prepare3D();
148 }

```

```

146
147 void Terminal::processInput()
148 {
149     error = "";
150     if (currentInput == "exit" || currentInput == "Exit")
151     {
152         isInTerminal = false;
153         history.clear();
154     }
155
156     else if (currentInput == "clear" || currentInput == "Clear")
157     {
158         num = -1;
159     }
160
161     else if (currentInput == "help" || currentInput == "Help")
162     {
163         num = 0;
164     }
165
166     else
167     {
168         string first, last;
169         size_t pos = currentInput.find(" ");
170
171         first = currentInput.substr(0, pos); // First half of string
172         last = currentInput.substr(pos + 1); // Second half of string
173
174         if (first == "read" || first == "Read")
175         {
176             num = atoi(last.c_str());
177             if (num <= 0 || num >= (signed int)prompts.size())
178             {
179                 error = "ERROR: Invalid file number";
180                 num = -1;
181             }
182         }
183
184         else
185         {
186             error = "ERROR: Invalid Command: " + currentInput;
187             num = -1;
188         }
189     }
190 }
191
192 void Terminal::Display()
193 {
194     if (isOn)
195         glColor3d(1, 0, 0);
196     else
197         glColor3d(0, 0, 1);
198
199     glPushMatrix();
200
201     glTranslated(translate[0], translate[1], translate[2]);

```

```

202         glRotated(rotate[0], 1, 0, 0);
203         glRotated(rotate[1], 0, 1, 0);
204         glRotated(rotate[2], 0, 0, 1);
205
206         // Do stuff here
207         glutSolidCube(.1);
208
209         glPopMatrix();
210     }
211
212     double Terminal::getX()
213     {
214         return translate[0];
215     }
216
217     double Terminal::getY()
218     {
219         return translate[1];
220     }
221
222     double Terminal::getZ()
223     {
224         return translate[2];
225     }
226
227     void Terminal::parseFile()
228     {
229         ifstream infile{ TERM_PATH + file };
230         string buff;
231
232         content.push_back("HELP"); // Help text is always the 0th tag in the
            terminals
233
234         getline(infile, buff);
235         prompts.push_back(buff); // Push back the file tag
236         getline(infile, buff);
237
238         while (buff != "<TAGS>")
239         {
240             size_t pos = buff.find("--");
241             if (pos != string::npos)
242             {
243                 prompts.push_back(buff.substr(0, pos));
244                 content.push_back(buff.substr(pos + 3));
245             }
246             getline(infile, buff);
247         }
248
249     }
250
251     string Terminal::getID()
252     {
253         return id;
254     }
255
256     Terminal::Terminal(const double(&_translate)[3], const double(&_rotate)[3], string

```

```

        _file, string _id)
257 {
258     // Copies the color
259     copy(begin(_translate), end(_translate), translate);
260
261     // Copies the vertices
262     copy(begin(_rotate), end(_rotate), rotate);
263
264     bTexture = SOIL_load_OGL_texture
265     (
266         "Resources\\Images\\banner.png",    // Image to load
267         SOIL_LOAD_AUTO,                      // ???
268         SOIL_CREATE_NEW_ID,
269         SOIL_FLAG_MIPMAPS | SOIL_FLAG_COMPRESS_TO_DXT // !?!?!?!
270     );
271
272     if (bTexture == 0)
273     {
274         Logger log;
275         vector<string> output = { "FATAL ERROR: SOIL cannot load terminal
276             banner", SOIL_last_result() };
277         log.logLine(output);
278         exit(SOIL_ERROR);
279     }
280
281     file = _file;
282
283     id = _id;
284
285     num = 0;
286
287     parseFile();
288 }

```

#### 4.1.39 TextEngine.h

```

1  /*****
2  * TextEngine.h
3  * This file was created by Jeremy Greenburg
4  * As part of The God Core game for the University of
5  * Tennessee at Martin's University Scholars Organization
6  *
7  * This file contains the declaration of the TextEngine class*
8  * Which uses glutBitmapCharacter to print strings into the *
9  * OpenGL window.
10 \*****/
11
12 #ifndef TEXTENGINE_H
13 #define TEXTENGINE_H
14
15 // For string lengths in displaying text
16 #include <string>
17
18 // For multiple lines of text
19 #include <vector>
20
21 class TextEngine

```

```

22 {
23 private:
24     // The path to the game's text files (.log's)
25     static const char* TEXT_PATH;
26     // The offset between lines of characters
27     static const double LINE_OFFSET;
28
29     void displayText(
30         // 2d start location of the text
31         double x, double y,
32         // rgb color of text
33         double r, double g, double b,
34         // glut font and text to be displayed
35         void* font,
36         std::vector<std::string> text);
37
38     // Searches a text file for text related to the tag, and returns all text
39     // within the tag
40     std::vector<std::string> findText(std::string fileName, std::string tag);
41 public:
42     // Takes the location to display the text, color of the text,
43     // The file to read from, and a tag to search for
44     void openFile(double x, double y, double r, double g, double b,
45         std::string fileName, std::string tag);
46
47     // Takes in a string to display
48     void printString(double x, double y, double r, double g, double b,
49         std::string text);
50
51     // Returns text from fileName specified by tag
52     std::vector<std::string> getText(std::string fileName, std::string tag);
53 };
54
55 #endif

```

#### 4.1.40 TextEngine.cpp

```

1  /*****\
2  * TextEngine.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the TextEngine class *
8  * For more information, see TextEngine.h *
9  \*****/
10
11 // TextEngine declaration and std::string
12 #include "TextEngine.h"
13
14 // std::ifstream
15 #include <fstream>
16
17 // Standard I/O for debugging
18 #include <iostream>
19

```

```

20 // OpenGL API
21 #include <gl\glut.h>
22
23 using namespace std;
24
25 // Initializing the constants
26 const char* TextEngine::TEXT_PATH = "Resources\\Text\\";
27 const double TextEngine::LINE_OFFSET = 10;
28
29 void TextEngine::displayText(double x, double y,
30     double r, double g, double b,
31     void* font, vector<string> text)
32 {
33     vector<string>::iterator it;
34
35     // Iterates throuh the text vector and prints it to the screen
36     for (it = text.begin(); it != text.end(); it++)
37     {
38         glColor3d(r, g, b);
39         glRasterPos2d(x, y);
40
41         for (unsigned int i = 0; i < it->length(); i++)
42         {
43             glutBitmapCharacter(font, (*it)[i]);
44         }
45
46         // Because glut does not print newlines
47         y += LINE_OFFSET;
48     }
49 }
50
51 vector<string> TextEngine::findText(string fileName, string tag)
52 {
53     // The tags are listed between dollar signs
54     string fullTag = '$' + tag + '$';
55
56     string fullPath = TEXT_PATH + fileName;
57
58     ifstream infile(fullPath);
59
60     // Buffer to read in data
61     string buff;
62     // Array to store strings
63     vector<string> data;
64
65     // Find the string(s) to read in
66     getline(infile, buff);
67     while (infile && buff != fullTag)
68     {
69         getline(infile, buff);
70     }
71
72     // Store the string(s)
73     getline(infile, buff);
74     while (infile && buff != "$END$")
75     {

```



```

76         data.push_back(buff);
77         getline(infile, buff);
78     }
79
80     infile.close();
81
82     return data;
83 }
84
85 void TextEngine::openFile(double x, double y,
86     double r, double g, double b,
87     string fileName, string tag)
88 {
89     vector<string> input = findText(fileName, tag);
90
91     displayText(x, y, r, g, b,
92         GLUT_BITMAP_HELVETICA_12,
93         input);
94 }
95
96 vector<string> TextEngine::getText(string fileName, string tag)
97 {
98     vector<string> input = findText(fileName, tag);
99
100     return input;
101 }
102
103 void TextEngine::printString(double x, double y, double r, double g, double b,
104     string text)
105 {
106     glColor3d(r, g, b);
107     glRasterPos2d(x, y);
108
109     for (unsigned int i = 0; i < text.length(); i++)
110     {
111         glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, text[i]);
112     }
113
114     // Vertical spacing
115     y += LINE_OFFSET;
116 }

```

#### 4.1.41 Triangle.h

```

1  /*****\
2  * Triangle.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the Triangle class *
8  * Which is used to hold the details of a 2D Triangle and *
9  * draw it to the screen *
10 \*****/
11
12 #ifndef TRIANGLE_H
13 #define TRIANGLE_H

```

```

14
15 class Triangle
16 {
17 private:
18     // Arrays containing the colors and the xyz vertices of the triangles
19     double color[4], vertices[9];
20 public:
21     // Takes in the vertices and color of the triangle
22     Triangle(const double(&new_vertices)[9], const double(&new_color)[4]);
23     // Print the triangle in 3D
24     void Display();
25     // Print the triangle in 2D
26     void Display2D();
27 };
28
29 #endif

```

#### 4.1.42 Triangle.cpp

```

1  /*****\
2  * Triangle.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the triangle class *
8  * For more information, see Triangle.h *
9  \*****/
10
11 // Class declaration
12 #include "Triangle.h"
13
14 // For std::copy
15 #include <iterator>
16 #include <utility>
17
18 // OpenGL API
19 #include <GL\glut.h>
20
21 using namespace std;
22
23
24 Triangle::Triangle(const double(&new_vertices)[9], const double(&new_color)[4])
25 {
26     // Copies the color entry
27     copy(begin(new_color), end(new_color), color);
28
29     // Copies the vertices
30     copy(begin(new_vertices), end(new_vertices), vertices);
31 }
32
33 void Triangle::Display()
34 {
35     // Sets OpenGL's color to the triangle's color
36     glColor4f(color[0], color[1], color[2], color[3]);
37
38     // Draws the triangle

```

```

39         glBegin(GL_TRIANGLES);
40         glVertex3d(vertices[0], vertices[1], vertices[2]);
41         glVertex3d(vertices[3], vertices[4], vertices[5]);
42         glVertex3d(vertices[6], vertices[7], vertices[8]);
43         glEnd();
44     }
45
46     void Triangle::Display2D()
47     {
48         // Set's OpenGL's color to the triangle's color
49         glColor4f(color[0], color[1], color[2], color[3]);
50
51         // Draw's the triangle without the Z vertices
52         glBegin(GL_TRIANGLES);
53         glVertex2d(vertices[0], vertices[1]);
54         glVertex2d(vertices[3], vertices[4]);
55         glVertex2d(vertices[6], vertices[7]);
56         glEnd();
57     }

```

#### 4.1.43 Trigger.h

```

1  #ifndef TRIGGER_H
2  #define TRIGGER_H
3
4  #include "GCTypes.h"
5
6  class Trigger
7  {
8  private:
9      void* trigger; // The object that activates the target
10     void* target;  // The object that is activated by the target
11
12     int triggerType; // The type (defined from GCTypes.h) of the trigger
13     int targetType;  // The type (defined from GCTypes.h) of the target
14
15 public:
16
17     int getTriggerType();
18     Trigger();
19
20 };
21
22 #endif

```

#### 4.1.44 Trigger.cpp

```

1  #include "Trigger.h"
2
3  int Trigger::getTriggerType()
4  {
5      return triggerType;
6  }
7
8  Trigger::Trigger()
9  {
10 }

```

#### 4.1.45 Triple.h

```
1  /*****\
2  * Triple.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the declaration of the Triple class *
8  * Which is just a simple 3-tuple really *
9  \*****/
10
11 #ifndef TRIPLE_H
12 #define TRIPLE_H
13
14 class Triple
15 {
16 public:
17     double a, b, c;
18 };
19
20 // For converting to a triple
21 Triple makeTrip(double _a, double _b, double _c);
22
23 #endif
```

#### 4.1.46 Triple.cpp

```
1  /*****\
2  * Triple.cpp *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
6  * *
7  * This file contains the definition of the TwoD class *
8  * For more information, see CameraControl.h *
9  \*****/
10
11 #include "Triple.h"
12
13 Triple makeTrip(double _a, double _b, double _c)
14 {
15     Triple ret;
16     ret.a = _a;
17     ret.b = _b;
18     ret.c = _c;
19
20     return ret;
21 }
```

#### 4.1.47 TwoD.h

```
1  /*****\
2  * TwoD.h *
3  * This file was created by Jeremy Greenburg *
4  * As part of The God Core game for the University of *
5  * Tennessee at Martin's University Scholars Organization *
```

```

6  *
7  * This file contains the declaration of the TwoD class
8  * Which is used to hold the data and functionality for
9  * Drawing in 2D with OpenGL
10 \*****/
11
12 #ifndef TWOD
13 #define TWOD
14
15 // I realize that four classes shared identical functions and data members.
16 // I remembered 222 and the inheritance lecture
17 // Are you proud of me yet, Dr. Guerin?
18
19 class TwoD
20 {
21 protected:
22     // The pixel boundaries of the screen
23     // My warning to you: comment as you code.
24     // Because a year later I have no idea where 767 and 1367 came from
25     // And I'm to scared to change them now.
26     const double SCREENTOP = 0, SCREENBOTTOM = 1080,
27             SCREENLEFT = 0, SCREENRIGHT = 1920;
28
29     // Prepares OpenGL draw in 2D
30     void prepare2D();
31
32     // "Resets" OpenGL to draw in 3D
33     void prepare3D();
34
35 };
36
37 #endif

```

#### 4.1.48 TwoD.cpp

```

1  /*****\
2  * TwoD.cpp
3  * This file was created by Jeremy Greenburg
4  * As part of The God Core game for the University of
5  * Tennessee at Martin's University Scholars Organization
6  *
7  * This file contains the definition of the TwoD class
8  * For more information, see TwoD.h
9  \*****/
10
11 #include "TwoD.h"
12
13 // OpenGL API
14 #include <gl\glut.h>
15
16 void TwoD::prepare2D()
17 {
18     // Disable writing to the z buffer
19     glDisable(GL_DEPTH_TEST);
20     glDepthMask(GL_FALSE);
21     // Disables lighting
22     glDisable(GL_LIGHTING);

```

```

23
24     // Create an orthogonal matrix to write on
25     glMatrixMode(GL_PROJECTION);
26     glPushMatrix();
27     glLoadIdentity();
28     glOrtho(SCREENLEFT, SCREENRIGHT, SCREENBOTTOM, SCREENTOP, -1, 1);
29     glMatrixMode(GL_MODELVIEW);
30     glPushMatrix();
31     glLoadIdentity();
32 }
33
34 void TwoD::prepare3D()
35 {
36     // Discards the orthogonal matrices
37     glMatrixMode(GL_PROJECTION);
38     glPopMatrix();
39     glMatrixMode(GL_MODELVIEW);
40     glPopMatrix();
41
42     // Enables writing to the z buffer
43     glEnable(GL_DEPTH_TEST);
44     glDepthMask(GL_TRUE);
45     // Renable lighting
46     glEnable(GL_LIGHTING);
47 }

```

## 4.2 Database

### 4.2.1 Walls

#	ID	LEVEL	X1	X2	X3	X4	Y1	Y2	Y3	Y4	Z1	Z2	Z3	Z4	R	G	B	A	Axis
1	lvceiling	LEVELZERO	-5	-5	8	8	1	1	1	1	-4	1	1	-4	0.7	0.7	0.7	1	0
2	lvfloor	LEVELZERO	-5	-5	8	8	-1	-1	-1	-1	-4	1	1	-4	0.7	0.7	0.7	1	0
3	room0lftwall	LEVELZERO	-5	-5	5	5	-1	1	1	-1	-4	-4	-4	-4	0.3	0.3	0.3	1	x
4	room0frntlftwall	LEVELZERO	5	5	5	5	-1	1	1	-1	-4	-4	-2.5	-2.5	0.3	0.3	0.3	1	z
5	room0frntrghtwall	LEVELZERO	5	5	5	5	-1	1	1	-1	-0.5	-0.5	1	1	0.3	0.3	0.3	1	z
6	room0backwall	LEVELZERO	-5	-5	-5	-5	-1	1	1	-1	-4	-4	1	1	0.3	0.3	0.3	1	z
7	room0rghwall	LEVELZERO	-5	-5	5	5	-1	1	1	-1	1	1	1	1	0.3	0.3	0.3	1	x
8	room0frnttopwall	LEVELZERO	5	5	5	5	0.5	1	1	0.5	-2.5	-2.5	-0.5	-0.5	0.3	0.3	0.3	1	z
9	room1lftwall	LEVELZERO	5	5	8	8	-1	1	1	-1	-4	-4	-4	-4	0.3	0.3	0.3	1	x

Document generated with SQLiteStudio v3.0.7

### 4.2.2 Doors

### 4.2.3 Switches

## 4.3 Scripts

## 4.4 Images

## 4.5 Music

## 4.6 Sounds