

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

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

January 25, 2017

Contents

1	Preamble	3
2	Programming	3
2.1	The Language	3
2.2	APIs	3
2.3	Game Engine	3
3	Appendices	5
3.1	Source Code	5
3.2	Database	106
3.3	Images	106
3.4	Music	107

1 Preamble

2 Programming

2.1 The Language

2.2 APIs

2.2.1 OpenGL

OpenGL, or the Open Graphics Library, is one of the most popular graphics libraries out there. It gives access to linear algebra functions for matrix manipulation (which is important, as 3D graphics relies heavily upon matrix transformations), keyboard and mouse input, and windowing. I chose to use OpenGL over a different graphics library, such as Microsoft's DirectX, because it is open source and cross platform, which would make porting my game to a different Operating System a much easier task if I ever decide to in the future.

2.2.2 SOIL

SOIL, or the Simple OpenGL Interface Library, is a small extension to OpenGL that I picked up along the way. It is a texture library that can load .jpg and .png images and bind them to an OpenGL texture, making it very simple to incorporate such images into my game.

2.2.3 FMOD

I chose FMOD as the base for my game's audio as it is a simple, lightweight, and free to use sound API, and most other audio APIs that I looked at lacked support for MP3 files.

2.2.4 SQLite

I decided to use SQLite for my database because it is a lightweight simplified version of a SQL database, allowing the game data to be stored and embedded in the application without taking much room or take a great deal of time to perform a query.

2.3 Game Engine

I crafted the engine of my game in C++ over two years starting in my second semester sophomore year and ending the first semester of my senior year.

2.3.1 Camera Control

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 player can move forwards and backwards, as well as strafe left and right. To correctly formulate the player's movement, I had to envision a circle centered on the player with a radius of the player's movement speed. Based on the angle from the x and z rotation, the next place that the player move is simply a spot on the circumference of the circle based on the rotation angle, and moving forward can be derived from this formula:

$$\begin{aligned} z &:= z \pm \text{moveSpeed} * \cos(\text{radian}(x_angle)) \\ x &:= x \mp \text{moveSpeed} * \sin(\text{radian}(x_angle)) \end{aligned}$$

Following that formula, it's simple to implement movement to the left, right by adding or subtracting 90°, and backwards movement by adding 180°.

Whenever OpenGL renders a new frame, the 'camera' is always returned to the origin of the map, so after drawing the level and before flushing the buffer, the Camera Control calls `glTranslate` to move the camera to the correct location, and then calls `glRotate` 3 times, once for each axis, to orient the camera in the correct direction.

2.3.2 Heads Up Display

The Heads Up Display is drawn after the level is draw, so that it overlays information to the player. It primarily is used to add a bit of flavor to the game by drawing the helmet for the player, but it also serves to display the developer console when activated.

The display also delivers a prompt to the user whenever they are in range of an object that can be interacted with.

2.3.3 Rectangles and Triangles

Rectangles and triangles are the two fundamental polygons that build up my game. Rectangles in particular make up the walls, floors, ceilings, doors, terminals, and most of the HUD and menu. They started as simply two arrays- one that holds all 9 (for triangles) or 12 (for rectangles) values describing the coordinates in the game that they inhabit, as well as a 4 value vector containing the objects RGBA values.

For collision purposes, when a rectangle class is expanded with the ability to calculate and store its norm and Plane equation (Form $ax + by + cz + d = 0$).

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

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

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

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

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

$$d = aAx + bAy + cAz$$

The norm of the plane can then be derived using the equation $\sqrt{a^2 + b^2 + c^2}$

2.3.4 2D

2.3.5 Powered Objects

2.3.6 Collision Engine

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, the 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.

2.3.7 MusicManager

2.3.8 TextEngine

2.3.9 SaveManager

2.3.10 Keyboard

2.3.11 Level Loading

2.3.12 Console and Logging

2.3.13 In Game Terminals

In game terminals are bound to a *terminal file*, a unique file that contains the contents of its respective terminal.

The terminal file is divided into two sections: the file names and the file contents. This is an example terminal file:

```
1 <FILES>
2 [01] Name1 -- TAG
3 [02] Name2 -- TAG2
4 [03] Name3 -- TAG3
5
6 <TAGS>
7 $HELP$
8 Type Read <num> to read the corresponding file
9 Type Clear to clear a file from the screen
10 Type Exit to exit the terminal
11 Type Help to see this message again
12 $END$
13
14 $TAG$
15 Content 1
16 $END$
17
18 $TAG2$
19 Content 2
20 $END$
21
22 $TAG3$
23 Content 3
24 $END$
```

3 Appendices

3.1 Source Code

3.1.1 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, 0.3f };
52 GLfloat light_position[] = { 0.0f, 0.0f, 0.0f, 0.0f }; // Currently nonexistent
53   until I can figure out how lighting works
54 GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
55 GLfloat mat_shininess[] = { 75 };
56 GLfloat lmodel_ambient[] = { 0.6f, 0.6f, 0.6f, 1.0f };
57
58 using namespace std;
59
60 //***** FUNCTION DEFINITIONS *****/

```

```

59
60 int main(int argc, char **argv)
61 {
62     Overlord.canContinue = initGame(argc, argv);
63
64     // Begin the game
65     glutMainLoop();
66
67     // If we ever get here, something bad happened
68
69     Logger log;
70     log.logLine("ERROR: GlutMainLoop exited early");
71
72     return EXIT_EARLY;
73 }
74
75 bool initGame(int argc, char **argv)
76 {
77     // Obliterate log file
78     Logger log;
79     log.nuke();
80
81     // Initialize GLUT
82     glutInit(&argc, argv);
83
84     // Create window
85     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
86     glutInitWindowPosition(50, 50);
87     glutInitWindowSize(500, 500);
88     glutCreateWindow("The God Core");
89
90     // register callbacks
91     glutDisplayFunc(manageScenes);
92     glutReshapeFunc(changeSize);
93     glutIdleFunc(manageScenes);
94     glutPassiveMotionFunc(motionPassive);
95     glutMouseFunc(mouse);
96     glutKeyboardFunc(normal);
97     glutKeyboardUpFunc(key_up);
98     glutSpecialFunc(special);
99
100    // Prebuilt function that works transparency
101    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
102
103    // Enable transparency
104    glEnable(GL_BLEND);
105    // Enable depth buffer
106    glEnable(GL_DEPTH_TEST);
107    // Let there be light!
108    glEnable(GL_LIGHTING);
109    // First light source
110    glEnable(GL_LIGHT0);
111
112    // Light properties
113    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
114    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

```

```

115         glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
116
117         // Light doesnt turn everything grey
118         glEnable(GL_COLOR_MATERIAL);
119
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     }

```

3.1.2 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 backwards
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

```

3.1.3 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 }

```

3.1.4 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 occured
19     bool collideWalls();
20     // Determines if other collision occured
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

```

3.1.5 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 = 0.5;
24 const double INTERACT_RADIUS = 1; // Object interactivity radius
25 const double COLLIDE_RADIUS = 0.5;
26
27 void CollisionEngine::checkInteract()
28 {
29     activeSwitch = NULL;
30     activeTerminal = NULL;
31     // Auto don't work in these parts
32     for (unsigned int i = 0; i < switches.size(); i++)
33     {
34         double distance = pow((switches[i].getX() + Cam.x), 2) + pow((
35             switches[i].getY() + Cam.y), 2) + pow((switches[i].getZ() + Cam
36             .z), 2);
37         distance = sqrt(distance);
38         double radii = (PLAYER_RADIUS + INTERACT_RADIUS);
39
40         if (distance < radii && switches[i].checkIfOn())
41         {
42             interactivity = true;
43             activeSwitch = &switches[i];
44             return;
45         }
46     }
47
48     for (unsigned int i = 0; i < terminals.size(); i++)
49     {
50         double distance = pow((terminals[i].getX() + Cam.x), 2) + pow((
51             terminals[i].getY() + Cam.y), 2) + pow((terminals[i].getZ() +
52             Cam.z), 2);
53         distance = sqrt(distance);
54         double radii = (PLAYER_RADIUS + INTERACT_RADIUS);
55
56         if (distance < radii && terminals[i].checkIfOn())
57         {
58             interactivity = true;
59             activeTerminal = &terminals[i];
60             return;
61         }
62     }
63 }

```

```

57         }
58     }
59
60     interactivity = false;
61 }
62
63 bool CollisionEngine::collideObjects()
64 {
65     for (unsigned int i = 0; i < terminals.size(); i++)
66     {
67         double distance = pow((terminals[i].getX() + Cam.x), 2) + pow((
            terminals[i].getY() + Cam.y), 2) + pow((terminals[i].getZ() +
            Cam.z), 2);
68         distance = sqrt(distance);
69         double radii = (PLAYER_RADIUS + COLLIDE_RADIUS);
70
71         if (distance < radii && terminals[i].checkIfOn())
72         {
73             return true;
74         }
75     }
76
77     return false;
78 }
79
80 bool CollisionEngine::collideWalls()
81 {
82     // Gotta check doors first
83     // And if you hit an open door
84     // You just ignore collision
85     // Because otherwise you can't fit
86     for (auto i : doors)
87     {
88         double distance = fabs(Cam.x * i.a + Cam.y * i.b + Cam.z * i.c + i
            .d); // Distance from door
89
90         if ((distance / i.getNorm() < PLAYER_RADIUS) && i.isInBounds())
91         {
92             if (i.isOpen) return false;
93             else return true;
94         }
95     }
96
97     for (auto i : walls)
98     {
99         double distance = fabs(Cam.x * i.a + Cam.y * i.b + Cam.z * i.c + i
            .d); // Distance from wall
100
101         if ((distance / i.getNorm() < PLAYER_RADIUS) && i.isInBounds())
            return true;
102     }
103
104     return false;
105 }
106
107 bool CollisionEngine::collide()

```

```

108 {
109     if (!collision)
110     {
111         return false;
112     }
113
114     checkInteract();
115     return (collideWalls() || collideObjects());
116 }

```

3.1.6 Console.h

```

1  /*****\
2  * Connsole.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 Console Class, *
8  * As well as the Trip struct for holding three integers *
9  * The Developer Console takes input from the user and *
10 * Activates various effects based upon what the user has *
11 * Typed in. *
12 \*****/
13
14 #ifndef CONSOLE_H
15 #define CONSOLE_H
16
17 // To act as a circular buffer for console history
18 #include <deque>
19 // Stores actual console input
20 #include <vector>
21 // std::string
22 #include <string>
23 // For processing text
24 #include "TextEngine.h"
25
26 // Windows API
27 #include <shlobj.h>
28
29
30 // To make rgb calues easier to store
31 #include "Triple.h"
32
33 class Console
34 {
35 private:
36     /***** Variables for the console itself *****/
37
38     // Triples for good color, bad color, and nuetral colors
39     Triple VALID_COLOR, INVALID_COLOR, NEUTRAL_COLOR;
40     // What the console "says" (aka what appears on screen)
41     std::deque<std::string> console_log;
42     // The colors of said strings
43     std::deque<Triple> console_color;
44     // Contains the actual player input
45     std::vector<std::string> console_input;

```



```

46     // The current (finished) input being processed
47     std::string currentInput;
48     // The current (unfinished) input being type
49     std::string currentText;
50     // Console History
51     TextEngine log;
52
53     // Path to core.sav
54     char CHAR_PATH[MAX_PATH];
55     std::string SAVE_PATH;
56
57     bool isActive;
58
59     // The bottom of the console
60     const int SCREENBOTTOM = 500;
61
62     // Prints the current input and console_history
63     void printInput();
64     // Processes completed input
65     void processInput();
66
67     // Command functions
68
69     // Toggles collision on and off
70     void toggleCollision();
71
72     // Toggles godMode on and off
73     void toggleGod();
74
75     // Decrpyts the entry in core.sav
76     void decrpytSave();
77
78     // Shutdowns program
79     void halt();
80
81     // Clears the console log
82     void clear();
83
84     // Writes input to core.sav
85     void writeToSave(std::string input);
86
87     // Reads a bit from the file
88     void readFromFile(std::string input);
89
90     // Changes the currently played track
91     void playSong(std::string input);
92
93 public:
94     // Initializes VALID_COLOR, INVALID_COLOR, NEUTRAL_COLOR, and SAVE_PATH
95     Console();
96     // Manages console functions if input has been provided
97     void activate(std::string input, std::string text);
98     // Manages console function if input is still being provided
99     void activate(std::string text);
100    // Returns the console_input[count]
101    std::string getHist(int count);

```

```

102         // Returns console_input.size()
103         int getHistNum();
104
105     };
106
107 #endif

```

3.1.7 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();

```

```

102
103     else if (currentInput == "Clear")
104         clear();
105
106     else if (currentInput.substr(0, 5) == "Play ")
107         playSong(currentInput.substr(5)); // Process everything after "
108         Play "
109
110     else if (currentInput == "Goto Main")
111     {
112         isInMain = true;
113         isInConsole = false;
114         HUD.toggleConsole();
115     }
116
117     // Invalid command
118     else
119     {
120         console_log.push_back("ERROR: Do not recognize \"" + currentInput
121             + "\"");
122         console_color.push_back(INVALID_COLOR);
123     }
124
125     // Clears the top of the console if too much history is added
126     if (console_log.size() > 9)
127     {
128         console_log.pop_front();
129         console_color.pop_front();
130     }
131
132     // Store the current input
133     console_input.push_back(currentInput);
134 }
135
136 void Console::writeToSave(string input)
137 {
138     // Writes whatever is in input to the save file.
139     // Probably not going to be good for loading purposes
140
141     SaveManager Jesus;
142
143     Jesus.saveLevel();
144
145     console_log.push_back("Saved: " + input);
146     console_color.push_back(VALID_COLOR);
147 }
148
149 void Console::readFromFile(string input)
150 {
151     // Syntax = Read core.sav
152     if (input == "core.sav")
153     {
154         ifstream infile(SAVE_PATH);
155
156         string text;

```

```

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     // Syntax = Read TAG FILE
164     else
165     {
166         // There should be a space seperating the file and the tag. We
167         // find that space
168         size_t pos = input.find(' ');
169
170         // If there ain't no space
171         if (pos == string::npos)
172         {
173             console_log.push_back("ERROR: No tag detected");
174             console_color.push_back(INVALID_COLOR);
175         }
176
177         // Hooray! There's a space
178         else
179         {
180             string tag = input.substr(0, pos);
181             string file = input.substr(pos + 1); // +1 to avoid the
182             // space
183
184             const char* TEXT_PATH = "Resources\\Text\\";
185             string fullPath = TEXT_PATH + file;
186
187             // Simply to test for the file's existence
188             ifstream infile(fullPath);
189
190             string text;
191             getline(infile, text);
192
193             // If there ain't no file
194             if (!infile)
195             {
196                 console_log.push_back("ERROR: File \"" + file +
197                                     "\" not found");
198                 console_color.push_back(INVALID_COLOR);
199             }
200
201             // Hooray! There's a file
202             else
203             {
204                 console_log.push_back("Reading \"" + file + "\"
205                                     with tag \"" + tag + "\"");
206                 console_color.push_back(VALID_COLOR);
207
208                 vector<string> readText = log.getText(file, tag);
209
210                 vector<string>::iterator it;

```

```

207
208         for (it = readText.begin(); it != readText.end();
                it++)
209         {
210             // Push everything we found into the log
211             console_log.push_back(*it);
212             console_color.push_back(NEUTRAL_COLOR);
213
214             // So we don't grow too much, keep bounds
                checking
215             if (console_log.size() > 9)
216             {
217                 console_log.pop_front();
218                 console_color.pop_front();
219             }
220         }
221     }
222     infile.close();
223 }
224 }
225 }
226 }
227
228 void Console::toggleCollision()
229 {
230     console_log.push_back("Noclip toggled.");
231     console_color.push_back(VALID_COLOR);
232
233     collision = !collision;
234 }
235
236 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::decryptSave()
245 {
246     SaveManager Jesus;
247
248     console_log.push_back(Jesus.readSave());
249     console_color.push_back(VALID_COLOR);
250 }
251
252 void Console::halt()
253 {
254     Logger log;
255     log.logLine("Exiting via console");
256     exit(EXIT_OK);
257 }
258
259 void Console::clear()
260 {

```

```

261         console_log.clear();
262         console_color.clear();
263         console_input.clear();
264     }
265
266     void Console::playSong(string input)
267     {
268         int sNum = getSongNum(input);
269
270         if (sNum == -1) // Invalid input
271         {
272             console_log.push_back("ERROR: " + input + " not a valid song file
273                                 ");
274             console_color.push_back(INVALID_COLOR);
275         }
276         else // Valid input
277         {
278             songNum = sNum;
279             changeSong = true;
280             string song = getSongName(sNum);
281             console_log.push_back("Now playing " + song);
282             console_color.push_back(VALID_COLOR);
283         }
284     }
285
286     string Console::getHist(int count)
287     {
288         int size = console_input.size();
289         if (console_input.empty())
290         {
291             return "";
292         }
293
294         // If, somehow, a fool manages to get a variable that is out of bounds
295
296         else if (count >= size)
297         {
298             return console_input.back();
299         }
300
301         else if (count < 0)
302         {
303             return console_input.front();
304         }
305
306         else
307         {
308             return console_input[size - count - 1];
309         }
310     }
311
312     int Console::getHistNum()
313     {
314         return console_input.size();
315     }

```

3.1.8 Cylinder.h

```
1  #ifndef CYLINDER_H
2  #define CYLINDER_H
3
4  #include <cstdlib>
5
6  #include <GL\glut.h>
7
8  class Cylinder
9  {
10 private:
11     double baseRadius, topRadius, height;
12     int stacks, slices;
13     double translate[3], rotate[3], color[4];
14     GLUQuadric *quad;
15 public:
16     Cylinder(double _baseRadius, double _topRadius, double _height, int
        _stacks, int _slices,
17         const double(&_translate)[3], const double(&_rotate)[3], const
        double (&_color)[4]);
18
19     void Display();
20     ~Cylinder();
21 };
22
23 #endif
```

3.1.9 Cylinder.cpp

```
1  #include "Cylinder.h"
2
3  // For copying
4  #include <iterator>
5  #include <utility>
6
7  using namespace std;
8
9  Cylinder::Cylinder(double _baseRadius, double _topRadius, double _height, int
    _stacks, int _slices,
10     const double(&_translate)[3], const double(&_rotate)[3], const double(&
        _color)[4])
11  {
12     baseRadius = _baseRadius;
13     topRadius = _topRadius;
14     height = _height;
15     stacks = _stacks;
16     slices = _slices;
17
18     copy(begin(_color), end(_color), color);
19     copy(begin(_translate), end(_translate), translate);
20     copy(begin(_rotate), end(_rotate), rotate);
21
22     quad = gluNewQuadric();
23 }
24
```



```

25 Cylinder::~Cylinder()
26 {
27     //gluDeleteQuadric(quad);
28 }
29
30 void Cylinder::Display()
31 {
32     glColor4d(color[0], color[1], color[2], color[3]);
33
34     glPushMatrix();
35
36     glTranslated(translate[0], translate[1], translate[2]);
37     glRotated(rotate[0], 1, 0, 0);
38     glRotated(rotate[1], 0, 1, 0);
39     glRotated(rotate[2], 0, 0, 1);
40
41     gluCylinder(quad, baseRadius, topRadius, height, slices, stacks);
42
43     glPopMatrix();
44 }

```

3.1.10 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

```

3.1.11 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 defintion 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     }

```

3.1.12 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

```

3.1.13 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         else
63         {
64
65             }
66     }
67
68 void GameManager::motionPassive(int x, int y)
69 {
70     static int _x = 0, _y = 0;
71
72     // If nothing else is happening basically
73     if (!isPaused && !isInConsole && !isInTerminal && !isInMain)
74     {
75         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         // Save current progress after loading level
157         SaveManager Jesus; // saves
158         Jesus.saveLevel();
159     }
160
161     else
162     {
163         lvl.displayLevel();
164     }
165 }
166
167 void GameManager::manageScenes()
168 {
169     // If we need to change the song, we can do it here
170     if (changeSong)
171     {
172         manageMusic();
173     }
174
175     // Clears the previous drawing
176     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
177
178     if (isPaused)
179     {
180         glutSetCursor(GLUT_CURSOR_LEFT_ARROW);
181         pause.display();
182     }
183
184     else if (isInTerminal)
185     {
186         activeTerminal->DisplayScreen();
187     }
188
189     else if (isInMain)
190     {
191         // Enable using textures (pictures)
192         glutSetCursor(GLUT_CURSOR_LEFT_ARROW);

```

```

193         static MainMenu MM;
194
195         // For some reason, MM breaks horribly when it's a global or class
            member
196         // So we'll just handle mouse clicks in the display function
197         // Rather than the mouse click function
198         // Because I'm a competent programmer
199         if (processClick)
200         {
201             MM.getClick(mouse_x, mouse_y);
202             processClick = false;
203         }
204
205         MM.display();
206     }
207
208     // glutSetCursor(GLUT_CURSOR_LEFT_ARROW); Keypads maybe?
209
210     else
211     {
212         // Enable using textures (pictures)
213         glutSetCursor(GLUT_CURSOR_NONE);
214         draw();
215
216         // Moves the camera to the correct position
217         Cam.Display();
218         if (goDim)
219         {
220             HUD.goDim(30);
221             goDim = false;
222         }
223
224         else if (goDark)
225         {
226             HUD.goDark(30);
227             goDark = false;
228         }
229
230         // Prompt the user to interact if we should
231         if (interactivity) HUD.displayWarning("INTERACT");
232         else HUD.displayWarning("");
233
234         // Prints the HUD
235         HUD.DisplayHUD();
236     }
237
238     // Displays the current drawing
239     glutSwapBuffers();
240 }
241
242 void GameManager::manageMusic()
243 {
244     // All variables need to persist between frames
245     static SoundClass background;
246
247     SoundSystem.releaseSound(background);

```



```

248         changeSong = false;
249
250         // Because you can never have too much bounds checking
251         if (songNum >= 0 && songNum <= 9)
252         {
253             std::string song = getSongName(songNum);
254             SoundSystem.makeSound(&background, song.c_str());
255             SoundSystem.playSound(background);
256         }
257     }
258
259     // Normal key presses
260     void GameManager::normal(unsigned char key, int x, int y)
261     {
262         board.normal(key, x, y);
263     }
264
265     // Key releases
266     void GameManager::key_up(unsigned char key, int x, int y)
267     {
268         board.key_up(key, x, y);
269     }
270
271     // Special keys
272     void GameManager::special(int key, int x, int y)
273     {
274         board.special(key, x, y);
275     }

```

3.1.14 GCTypes.h

```

1  /*****\
2  * GCTypes.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 integer types corresponding to various *
8  * In game object types *
9  \*****/
10
11 #ifndef GC_TYPES_H
12 #define GC_TYPES_H
13
14 // Object Types
15
16 #define T_NULL 0 // Nothing
17 #define T_DOOR 1 // Door
18 #define T_TERMINAL 2 // Terminal
19 #define T_SWITCH 3 // Switch
20 #define T_LEVEL_END 4 // Switch that ends level
21
22 typedef int Gctype;
23
24 #endif

```

3.1.15 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 #include "Trigger.h"
25 #include "Cylinder.h"
26
27 // Remember that if you're doing anything else, globals are bad.
28 // But we're in the hellscape that is graphics
29 // There are no rules here
30 // Only madness dwells here
31
32 // Typedefs make life easy
33 typedef std::vector<Plane> vr;
34 typedef std::vector<Door> vd;
35 typedef std::vector<Switch> vs;
36 typedef std::vector<Terminal> vt;
37 typedef std::vector<Trigger> vtr;
38 typedef std::vector<Cylinder> vc;
39
40 // Pointers to various interactive objects
41 extern Switch *activeSwitch;
42 extern Terminal *activeTerminal;
43
44 // Vectors containing all of the level's assets
45 extern vr walls;
46 extern vd doors;
47 extern vs switches;
48 extern vt terminals;
49 extern vtr triggers;
50 extern vc cylinders;
51
52 extern bool
53     // Are we colliding / Can we die?
54     collision, godMode,
55     // Go dim or go dark?
56     goDim, goDark,

```

```

57         // Dunno if I actually need this one
58         loading,
59         // Is in various different stages of non-normal play
60         isInConsole, isPaused, isInTerminal, isInMain,
61         // Should we change the song?
62         changeSong,
63         // Is something in interaction range?
64         interactivity;
65
66 // Number of song to change to
67 extern int songNum;
68
69 // Current level (int and string)
70 extern int levelNum;
71 extern std::string curr_level;
72
73 // Constant strings of the song names
74 extern const char *SONG0, *SONG1, *SONG2, *SONG3, *SONG4, *SONG5,
75                 *SONG6, *SONG7, *SONG8, *SONG9;
76
77 // Lots of global objects
78 extern HeadsUpDisplay HUD;
79 extern CameraControl Cam;
80 extern PauseScreen pause;
81 extern Level lvl;
82
83 // Converts a songname to an integer
84 int getSongNum(std::string input);
85 // Converts an integer to a songname
86 std::string getSongName(int input);
87 // Converts a level name to an integer
88 int getLevelNum(std::string input);
89 // Converts level_num to a string in curr_level
90 std::string getLevelString(int input);
91
92 #endif

```

3.1.16 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 vtr triggers;
17 vc cylinders;
18

```

```

19 Switch *activeSwitch = NULL;
20 Terminal *activeTerminal = NULL;
21
22 bool collision = true;
23 bool godMode = false;
24 bool goDim = false;
25 bool goDark = false;
26 bool loading = true;
27 bool isInConsole = false;
28 bool isPaused = false;
29 bool isInTerminal = false;
30 bool isInMain = true;
31 bool changeSong = true;
32 bool interactivity = false;
33
34 int songNum = 0;
35
36 int levelNum = 0;
37 //int levelNum = 2;
38 std::string curr_level = "LEVELZERO";
39 //std::string curr_level = "LEVELTWO";
40
41 const char* SONG0 = "Dark Fog.mp3";
42 const char* SONG1 = "Mismer.mp3";
43 const char* SONG2 = "One Sly Move.mp3";
44 const char* SONG3 = "Hypnothis.mp3";
45 const char* SONG4 = "Cold Hope.mp3";
46 const char* SONG5 = "Spacial Harvest.mp3";
47 const char* SONG6 = "Lightless Dawn.mp3";
48 const char* SONG7 = "Zombie Flood.mp3";
49 const char* SONG8 = "Get on my Level.mp3";
50 const char* SONG9 = "Story of Life.mp3";
51
52 HeadsUpDisplay HUD;
53 CameraControl Cam;
54 PauseScreen pause;
55 Level lvl;
56
57 int getSongNum(std::string input)
58 {
59     if (input == SONG0 || input == "0")
60         return 0;
61     if (input == SONG1 || input == "1")
62         return 1;
63     if (input == SONG2 || input == "2")
64         return 2;
65     if (input == SONG3 || input == "3")
66         return 3;
67     if (input == SONG4 || input == "4")
68         return 4;
69     if (input == SONG5 || input == "5")
70         return 5;
71     if (input == SONG6 || input == "6")
72         return 6;
73     if (input == SONG7 || input == "7")
74         return 7;

```

```

75         if (input == SONG8 || input == "8")
76             return 8;
77         if (input == SONG9 || input == "9")
78             return 9;
79         return -1; // Invalid song
80     }
81
82     std::string getSongName(int input)
83     {
84         std::string ret;
85         switch (input)
86         {
87             case 0: ret = SONG0;
88                     break;
89             case 1: ret = SONG1;
90                     break;
91             case 2: ret = SONG2;
92                     break;
93             case 3: ret = SONG3;
94                     break;
95             case 4: ret = SONG4;
96                     break;
97             case 5: ret = SONG5;
98                     break;
99             case 6: ret = SONG6;
100                     break;
101             case 7: ret = SONG7;
102                     break;
103             case 8: ret = SONG8;
104                     break;
105             case 9: ret = SONG9;
106                     break;
107             default: ret = "\0";;
108                     break;
109         }
110
111         return ret;
112     }
113
114     int getLevelNum(std::string input)
115     {
116         if (input == "LEVELZERO" || input == "LEVELZERO\n")
117             return 0;
118         if (input == "LEVELONE" || input == "LEVELONE\n")
119             return 1;
120         if (input == "LEVELTWO")
121             return 2;
122         if (input == "LEVELTHREE")
123             return 3;
124         if (input == "LEVELFOUR")
125             return 4;
126         if (input == "LEVELFIVE")
127             return 5;
128         if (input == "LEVELSIX")
129             return 6;
130         if (input == "LEVELSEVEN")

```

```

131         return 7;
132     if (input == "LEVELEIGHT")
133         return 8;
134     if (input == "LEVELNINE")
135         return 9;
136     return -1; // Invalid song
137 }
138
139 std::string getLevelString(int input)
140 {
141     std::string ret;
142     switch (input)
143     {
144     case 0: ret = "LEVELZERO";
145             break;
146     case 1: ret = "LEVELONE";
147             break;
148     case 2: ret = "LEVELTWO";
149             break;
150     case 3: ret = "LEVELTHREE";
151             break;
152     case 4: ret = "LEVELFOUR";
153             break;
154     case 5: ret = "LEVELFIVE";
155             break;
156     case 6: ret = "LEVELSIX";
157             break;
158     case 7: ret = "LEVELSEVEN";
159             break;
160     case 8: ret = "LEVELEIGHT";
161             break;
162     case 9: ret = "LEVELNINE";
163             break;
164     default: ret = "ERROR";
165             break;
166     }
167
168     return ret;
169 }

```

3.1.17 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

```

3.1.18 HeadsUpDiplay.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 helmet
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     }

```

3.1.19 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

```

3.1.20 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         Cam.resetCam();
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         break;
315
316     case GLUT_KEY_DOWN:
317         if (isInConsole || isInTerminal)
318         {
319             getNext = true;

```

```

319         getPrev = false;
320
321         // To ensure that the input is updated BEFORE next key
           press
322         normal(0, 0, 0);
323     }
324     break;
325 }
326
327 if (fullScreen)
328 {
329     glutFullScreen();
330 }
331
332 else
333 {
334     glutReshapeWindow(1367, 767);
335     glutPositionWindow(50, 50);
336 }
337 }
338
339 void Keyboard::interact()
340 {
341     // Only do things if we actually can
342     if (interactivity)
343     {
344         if (activeSwitch != NULL)
345         {
346             activeSwitch->toggleTarget();
347
348             for (unsigned int i = 0; i < triggers.size(); i++)
349             {
350                 triggers[i].tryToTrigger(activeSwitch, T_SWITCH);
351             }
352         }
353
354         else if (activeTerminal != NULL)
355         {
356             isInTerminal = true;
357
358             for (unsigned int i = 0; i < triggers.size(); i++)
359             {
360                 triggers[i].tryToTrigger(activeTerminal,
                                           T_TERMINAL);
361             }
362         }
363     }
364 }

```

3.1.21 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 loadCylinders(sqlite3 *db);
40     void loadSwitches(sqlite3 *db);
41     void loadTerminals(sqlite3 *db);
42     void loadTriggers(sqlite3 *db);
43
44     // Binds the triggering object and target object to a single trigger
45     bool bindTrigger(std::string id, std::string trigger, std::string
        triggerType);
46     bool bindTarget(std::string id, std::string target, std::string targetType
        );
47 public:
48     // Manages the loading of the level
49     void loadLevel(std::string levelName);
50     // Draws the level
51     void displayLevel();
52 };
53
54 #endif

```

3.1.22 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 #include <iostream>
25
26 using namespace std;
27
28 void Level::loadWalls(sqlite3 *db)
29 {
30     walls.clear();
31     // Prepared Statement
32     sqlite3_stmt *stm;
33     // SQL command
34     string cmd;
35     // Connection Error Test
36     int err;
37     cmd = "SELECT * FROM walls WHERE LEVEL = \"" + currLevel + "\"";
38
39     err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
40
41     if (err != SQLITE_OK)
42     {
43         Logger log;
44         vector<string> output = { "FATAL ERROR: failed to load walls from", currLevel };
45         log.logLine(output);
46         exit(STATEMENT_ERROR);
47     }
48
49     // While we still get rows of output
50     while (sqlite3_step(stm) == SQLITE_ROW)
51     {
52         double x1, x2, x3, x4,
53             y1, y2, y3, y4,
54             z1, z2, z3, z4,
55             r, g, b, a;
56         string axis;
57
58         x1 = sqlite3_column_double(stm, 2);
59         x2 = sqlite3_column_double(stm, 3);

```

```

60         x3 = sqlite3_column_double(stm, 4);
61         x4 = sqlite3_column_double(stm, 5);
62
63         y1 = sqlite3_column_double(stm, 6);
64         y2 = sqlite3_column_double(stm, 7);
65         y3 = sqlite3_column_double(stm, 8);
66         y4 = sqlite3_column_double(stm, 9);
67
68         z1 = sqlite3_column_double(stm, 10);
69         z2 = sqlite3_column_double(stm, 11);
70         z3 = sqlite3_column_double(stm, 12);
71         z4 = sqlite3_column_double(stm, 13);
72
73         r = sqlite3_column_double(stm, 14);
74         g = sqlite3_column_double(stm, 15);
75         b = sqlite3_column_double(stm, 16);
76         a = sqlite3_column_double(stm, 17);
77
78         axis = reinterpret_cast<const char*>(sqlite3_column_text(stm, 18))
79             ;
80
81         char ax;
82         if (axis == "x") ax = 'x';
83         else if (axis == "y") ax = 'y';
84         else if (axis == "z") ax = 'z';
85         else ax = 0;
86
87         double verts[12] =
88         {
89             x1, y1, z1,
90             x2, y2, z2,
91             x3, y3, z3,
92             x4, y4, z4
93         };
94         double colors[4] = { r, g, b, a };
95
96         Plane rect(verts, colors, ax);
97
98         walls.push_back(rect);
99
100     }
101
102     /*
103     Logger log;
104     vector<string> output = { "Loaded walls on", currLevel };
105     log.logLine(output);
106     */
107
108     // Deconstructs the statement
109     sqlite3_finalize(stm);
110 }
111
112 void Level::loadDoors(sqlite3 *db)
113 {
114     doors.clear();
115     // Prepared Statement
116     sqlite3_stmt *stm;

```

```

115         // SQL command
116         string cmd;
117         // Connection Error Test
118         int err;
119         cmd = "SELECT * FROM doors WHERE LEVEL = \"" + currLevel + "\"";
120
121         err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
122
123         if (err != SQLITE_OK)
124         {
125             Logger log;
126             vector<string> output = { "FATAL ERROR: Can't load doors while
127                                     loading", currLevel };
128             log.logLine(output);
129             exit(STATEMENT_ERROR);
130         }
131
132         // While we still get rows of output
133         while (sqlite3_step(stm) == SQLITE_ROW)
134         {
135             double x1, x2, x3, x4,
136                   y1, y2, y3, y4,
137                   z1, z2, z3, z4,
138                   r, g, b, a;
139             string id;
140             string axis;
141
142             id = reinterpret_cast<const char*>(sqlite3_column_text(stm, 0));
143             x1 = sqlite3_column_double(stm, 2);
144             x2 = sqlite3_column_double(stm, 3);
145             x3 = sqlite3_column_double(stm, 4);
146             x4 = sqlite3_column_double(stm, 5);
147
148             y1 = sqlite3_column_double(stm, 6);
149             y2 = sqlite3_column_double(stm, 7);
150             y3 = sqlite3_column_double(stm, 8);
151             y4 = sqlite3_column_double(stm, 9);
152
153             z1 = sqlite3_column_double(stm, 10);
154             z2 = sqlite3_column_double(stm, 11);
155             z3 = sqlite3_column_double(stm, 12);
156             z4 = sqlite3_column_double(stm, 13);
157
158             r = sqlite3_column_double(stm, 14);
159             g = sqlite3_column_double(stm, 15);
160             b = sqlite3_column_double(stm, 16);
161             a = sqlite3_column_double(stm, 17);
162
163             a = sqlite3_column_double(stm, 17);
164
165             axis = reinterpret_cast<const char*>(sqlite3_column_text(stm, 18))
166                 ;
167
168             char ax;
169             if (axis == "x") ax = 'x';

```



```

169         else if (axis == "y") ax = 'y';
170         else if (axis == "z") ax = 'z';
171         else ax = 0;
172
173         double verts[12] =
174         {
175             x1, y1, z1,
176             x2, y2, z2,
177             x3, y3, z3,
178             x4, y4, z4
179         };
180         double colors[4] = { r, g, b, a };
181
182         Plane rect(verts, colors, ax);
183
184         doors.push_back(Door(rect, id));
185     }
186
187     Logger log;
188     vector<string> output = { "Loaded doors on", currLevel };
189     log.logLine(output);
190
191     // Deconstructs the statement
192     sqlite3_finalize(stm);
193 }
194
195 void Level::loadCylinders(sqlite3 *db)
196 {
197     cylinders.clear();
198     // Prepared Statement
199     sqlite3_stmt *stm;
200     // SQL command
201     string cmd;
202     // Connection Error Test
203     int err;
204     cmd = "SELECT * FROM cylinders WHERE LEVEL = \"" + currLevel + "\"";
205
206     err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
207
208     if (err != SQLITE_OK)
209     {
210         Logger log;
211         vector<string> output = { "FATAL ERROR: Can't load cylinders while
212             loading", currLevel };
213         log.logLine(output);
214
215         exit(STATEMENT_ERROR);
216     }
217
218     // While we still get rows of output
219     while (sqlite3_step(stm) == SQLITE_ROW)
220     {
221         double xt, yt, zt,
222             xr, yr, zr,
223             r, g, b, a,
224             baseRadius, topRadius, height;

```

```

224         int stacks, slices;
225
226
227         xt = sqlite3_column_double(stm, 1);
228         yt = sqlite3_column_double(stm, 2);
229         zt = sqlite3_column_double(stm, 3);
230
231         xr = sqlite3_column_double(stm, 4);
232         yr = sqlite3_column_double(stm, 5);
233         zr = sqlite3_column_double(stm, 6);
234
235         baseRadius = sqlite3_column_double(stm, 7);
236         topRadius = sqlite3_column_double(stm, 8);
237         height = sqlite3_column_double(stm, 9);
238
239         stacks = sqlite3_column_int(stm, 10);
240         slices = sqlite3_column_int(stm, 11);
241
242         r = sqlite3_column_double(stm, 12);
243         g = sqlite3_column_double(stm, 13);
244         b = sqlite3_column_double(stm, 14);
245         a = sqlite3_column_double(stm, 15);
246
247
248         double translate[3] = { xt, yt, zt };
249         double rotate[3] = { xr, yr, zr };
250         double colors[4] = { r, g, b, a };
251
252         cylinders.push_back(Cylinder(baseRadius, topRadius, height, stacks
253             , slices, translate, rotate, colors));
254     }
255
256     Logger log;
257     vector<string> output = { "Loaded cylinders on", currLevel };
258     log.logLine(output);
259
260     // Deconstructs the statement
261     sqlite3_finalize(stm);
262 }
263
264 void Level::loadSwitches(sqlite3 *db)
265 {
266     switches.clear();
267     // Prepared Statement
268     sqlite3_stmt *stm;
269     // SQL command
270     string cmd;
271     // Connection Error Test
272     int err;
273     cmd = "SELECT * FROM switches WHERE LEVEL = \"" + currLevel + "\"";
274
275     err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
276
277     if (err != SQLITE_OK)
278     {

```

```

279         Logger log;
280         vector<string> output = { "FATAL ERROR: Can't load switches while
           loading", currLevel };
281         log.logLine(output);
282
283         exit(STATEMENT_ERROR);
284     }
285
286     // While we still get rows of output
287     while (sqlite3_step(stm) == SQLITE_ROW)
288     {
289         double xt, yt, zt,
290               xr, yr, zr;
291         string target, s_type, id;
292         int i_type;
293         bool isOn;
294
295         id = reinterpret_cast<const char*>(sqlite3_column_text(stm, 0));
296         target = reinterpret_cast<const char*>(sqlite3_column_text(stm, 2)
           );
297         xt = sqlite3_column_double(stm, 3);
298         yt = sqlite3_column_double(stm, 4);
299         zt = sqlite3_column_double(stm, 5);
300
301         xr = sqlite3_column_double(stm, 6);
302         yr = sqlite3_column_double(stm, 7);
303         zr = sqlite3_column_double(stm, 8);
304
305         s_type = reinterpret_cast<const char*>(sqlite3_column_text(stm, 9)
           );
306
307         isOn = (bool)sqlite3_column_int(stm, 10);
308
309         double translate[3] = { xt, yt, zt };
310         double rotate[3] = { xr, yr, zr };
311
312         if (s_type == "DOOR")
313             i_type = T_DOOR;
314         else if (s_type == "TERMINAL")
315             i_type = T_TERMINAL;
316         else if (s_type == "LEVEL_END")
317             i_type = T_LEVEL_END;
318         else
319         {
320             Logger log;
321             vector<string> output = { "Failed to evaluate string type
           entry: ", s_type, "for switch ", id };
322             log.logLine(output);
323
324             exit(DATA_ENTRY_ERROR);
325         }
326
327         switches.push_back(Switch(translate, rotate, i_type, id, isOn));
328
329         bool assigned = false;
330

```

```

331         if (s_type == "LEVEL_END")
332         {
333             assigned = true;
334
335             Logger log;
336             vector<string> output = { "Switch ", id, " bound to end
                                   level" };
337             log.logLine(output);
338         }
339
340         else if (s_type == "DOOR")
341         {
342             for (unsigned int i = 0; i < doors.size(); i++)
343             {
344                 if (doors[i].getID() == target)
345                 {
346                     Logger log;
347                     vector<string> output = { "Binding switch
                                   ", id, " to door", target };
348                     log.logLine(output);
349
350                     switches[switches.size() - 1].assign(&(
                                   doors[i]));
351
352                     assigned = true;
353                 }
354             }
355         }
356
357         else if (s_type == "TERMINAL")
358         {
359             for (unsigned int i = 0; i < terminals.size(); i++)
360             {
361                 if (terminals[i].getID() == target)
362                 {
363                     Logger log;
364                     vector<string> output = { "Binding switch
                                   ", id, " to terminal", target };
365                     log.logLine(output);
366
367                     switches[switches.size() - 1].assign(&(
                                   terminals[i]));
368
369                     assigned = true;
370                 }
371             }
372         }
373
374         if (!assigned)
375         {
376             Logger log;
377             vector<string> output = { "Failed to bind switch ", id, "
                                   to a ", s_type };
378             log.logLine(output);
379
380             exit(BINDING_ERROR);

```

```

381         }
382     }
383
384     Logger log;
385     vector<string> output = { "Loaded switches on", currLevel };
386     log.logLine(output);
387
388     // Deconstructs the statement
389     sqlite3_finalize(stm);
390 }
391
392 void Level::loadTerminals(sqlite3 *db)
393 {
394     terminals.clear();
395     // Prepared Statement
396     sqlite3_stmt *stm;
397     // SQL command
398     string cmd;
399     // Connection Error Test
400     int err;
401     cmd = "SELECT * FROM terminals WHERE LEVEL = \"" + currLevel + "\"";
402
403     err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
404
405     if (err != SQLITE_OK)
406     {
407         Logger log;
408         vector<string> output = { "FATAL ERROR: Can't load terminals while
            loading", currLevel };
409         log.logLine(output);
410
411         exit(STATEMENT_ERROR);
412     }
413
414     // While we still get rows of output
415     while (sqlite3_step(stm) == SQLITE_ROW)
416     {
417         double xt, yt, zt,
418             xr, yr, zr;
419         string file, id;
420         id = reinterpret_cast<const char*>(sqlite3_column_text(stm, 0));
421         file = reinterpret_cast<const char*>(sqlite3_column_text(stm, 2));
422         xt = sqlite3_column_double(stm, 3);
423         yt = sqlite3_column_double(stm, 4);
424         zt = sqlite3_column_double(stm, 5);
425
426         xr = sqlite3_column_double(stm, 6);
427         yr = sqlite3_column_double(stm, 7);
428         zr = sqlite3_column_double(stm, 8);
429
430         double translate[3] = { xt, yt, zt };
431         double rotate[3] = { xr, yr, zr };
432
433         Logger log;
434         log.logLine(id);
435     }

```

```

436         terminals.push_back(Terminal(translate, rotate, file, id));
437     }
438
439
440     Logger log;
441     vector<string> output = { "Loaded terminals on", currLevel };
442     log.logLine(output);
443
444     // Deconstructs the statement
445     sqlite3_finalize(stm);
446 }
447
448 void Level::loadTriggers(sqlite3 *db)
449 {
450     triggers.clear();
451     // Prepared Statement
452     sqlite3_stmt *stm;
453     // SQL command
454     string cmd;
455     // Connection Error Test
456     int err;
457     cmd = "SELECT * FROM triggers WHERE LEVEL = \"" + currLevel + "\"";
458
459     err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);
460
461     if (err != SQLITE_OK)
462     {
463         Logger log;
464         vector<string> output = { "FATAL ERROR: Can't load triggers while
                                loading", currLevel };
465         log.logLine(output);
466
467         exit(STATEMENT_ERROR);
468     }
469
470     // While we still get rows of output
471     while (sqlite3_step(stm) == SQLITE_ROW)
472     {
473         string target, trigger, targetType, triggerType, id;
474         int i_targetType, i_triggerType;
475
476         id = reinterpret_cast<const char*>(sqlite3_column_text(stm, 0));
477         trigger = reinterpret_cast<const char*>(sqlite3_column_text(stm,
                                2));
478         target = reinterpret_cast<const char*>(sqlite3_column_text(stm, 3)
                                );
479         triggerType = reinterpret_cast<const char*>(sqlite3_column_text(
                                stm, 4));
480         targetType = reinterpret_cast<const char*>(sqlite3_column_text(stm
                                , 5));
481
482         if (triggerType == "SWITCH")
483             i_triggerType = T_SWITCH;
484         else if (triggerType == "TERMINAL")
485             i_triggerType = T_TERMINAL;
486         else

```

```

487         {
488             Logger log;
489             vector<string> output = { "Failed to evaluate string
                                     trigger type entry: ", triggerType, "for trigger ", id
                                     };
490             log.logLine(output);
491
492             exit(DATA_ENTRY_ERROR);
493         }
494
495         if (targetType == "SWITCH")
496             i_targetType = T_SWITCH;
497         else if (targetType == "TERMINAL")
498             i_targetType = T_TERMINAL;
499         else
500         {
501             Logger log;
502             vector<string> output = { "Failed to evaluate string
                                     trigger type entry: ", targetType, "for trigger ", id
                                     };
503             log.logLine(output);
504
505             exit(DATA_ENTRY_ERROR);
506         }
507
508         triggers.push_back(Trigger(i_triggerType, i_targetType));
509
510         bool assigned = bindTrigger(id, trigger, triggerType) &&
                        bindTarget(id, target, targetType);
511
512         if (!assigned)
513         {
514             Logger log;
515             vector<string> output = { "Failed to bind trigger ", id };
516             log.logLine(output);
517
518             exit(BINDING_ERROR);
519         }
520     }
521
522     Logger log;
523     vector<string> output = { "Loaded trigger on", currLevel };
524     log.logLine(output);
525
526     // Deconstructs the statement
527     sqlite3_finalize(stm);
528 }
529
530 bool Level::bindTrigger(string id, string trigger, string triggerType)
531 {
532     if (triggerType == "SWITCH")
533     {
534         for (unsigned int i = 0; i < switches.size(); i++)
535         {
536             if (switches[i].getID() == trigger)
537             {

```

```

538         Logger log;
539         vector<string> output = { "Binding trigger ", id,
540             " to trigger-switch", trigger };
541         log.logLine(output);
542         triggers[triggers.size() - 1].bindTrigger(&(
543             switches[i]));
544         return true;
545     }
546 }
547 }
548
549 else if (triggerType == "TERMINAL")
550 {
551     for (unsigned int i = 0; i < terminals.size(); i++)
552     {
553         if (terminals[i].getID() == trigger)
554         {
555             Logger log;
556             vector<string> output = { "Binding trigger ", id,
557                 " to trigger-terminal", trigger };
558             log.logLine(output);
559             triggers[triggers.size() - 1].bindTrigger(&(
560                 terminals[i]));
561             return true;
562         }
563     }
564 }
565
566 return false;
567 }
568
569 bool Level::bindTarget(string id, string target, string targetType)
570 {
571
572     if (targetType == "SWITCH")
573     {
574         for (unsigned int i = 0; i < switches.size(); i++)
575         {
576             if (switches[i].getID() == target)
577             {
578                 Logger log;
579                 vector<string> output = { "Binding trigger ", id,
580                     " to target-switch", target };
581                 log.logLine(output);
582                 triggers[triggers.size() - 1].bindTarget(&(
583                     switches[i]));
584                 return true;
585             }
586         }
587     }

```



```

588
589     else if (targetType == "TERMINAL")
590     {
591         for (unsigned int i = 0; i < terminals.size(); i++)
592         {
593             if (terminals[i].getID() == target)
594             {
595                 Logger log;
596                 vector<string> output = { "Binding trigger ", id,
                    " to target-terminal", target };
                    log.logLine(output);
                    triggers[triggers.size() - 1].bindTarget(&(
                        terminals[i]));
                    return true;
                }
            }
        }
        return false;
    }
}

608
609 void Level::loadLevel(std::string levelName)
610 {
611     Logger log;
612     vector<string> output = { "Starting to load", levelName };
613     log.logLine(output);
614
615     if (quadratic == NULL)
616     {
617         quadratic = gluNewQuadric();
618     }
619
620     currLevel = levelName;
621
622     // Connection to SQL database
623     sqlite3 *db;
624     // 1 if error with DB
625     int connectErr = sqlite3_open("Data.db", &db);
626
627     if (connectErr != SQLITE_OK)
628     {
629         Logger log;
630         log.logLine("FATAL ERROR: Can't access database");
631
632         exit(DATABASE_ERROR);
633     }
634
635     loadWalls(db);
636     loadDoors(db);
637     loadCylinders(db);
638     loadTerminals(db);
639
640     // Loading switches must be after doors/terminals to properly bind
641     loadSwitches(db);

```

```

642
643         // Loading triggers must be done last to properly bind
644         loadTriggers(db);
645
646         // Closes the database
647         sqlite3_close(db);
648
649         output[0] = "Finished loading";
650         log.logLine(output);
651
652         Cam.resetCam();
653
654         // Get out of wall
655         for (unsigned int i = 0; i < 10; i++)
656         {
657             Cam.moveForward(1);
658         }
659     }
660
661     void Level::displayLevel()
662     {
663         for (auto i : walls)
664         {
665             i.Display();
666         }
667
668         for (auto i : doors)
669         {
670             i.Display();
671         }
672
673         for (auto i : cylinders)
674         {
675             i.Display();
676         }
677
678         for (auto i : switches)
679         {
680             i.Display();
681         }
682
683         for (auto i : terminals)
684         {
685             i.Display();
686         }
687     }

```

3.1.23 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

```

3.1.24 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  }

```

3.1.25 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

```

3.1.26 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 #include "SaveManager.h"

```

```

21
22 using namespace std;
23
24 MainMenu::MainMenu()
25 {
26     texture = SOIL_load_OGL_texture
27         (
28             "Resources\\Images\\Main.png",    // Image to load
29             SOIL_LOAD_AUTO,                    //
30             ???
31             SOIL_CREATE_NEW_ID,
32             SOIL_FLAG_MIPMAPS | SOIL_FLAG_NTSC_SAFE_RGB |
33             SOIL_FLAG_COMPRESS_TO_DXT // !?!?!?!
34         );
35
36     if (texture == 0)
37     {
38         Logger log;
39         vector<string> output = {"FATAL ERROR: SOIL cannot load image",
40                                 SOIL_last_result()};
41         log.logLine(output);
42         exit(SOIL_ERROR);
43     }
44 }
45
46 void MainMenu::drawMainPic()
47 {
48     glEnable(GL_TEXTURE_2D);
49
50     glBindTexture(GL_TEXTURE_2D, texture); // Prepares the texture for usage
51
52     glColor3d(1, 1, 1);
53     glBegin(GL_QUADS);
54     glTexCoord2d(0, 0);    glVertex2d(SCREENLEFT, SCREENTOP);
55     glTexCoord2d(0, 1);    glVertex2d(SCREENLEFT, SCREENBOTTOM);
56     glTexCoord2d(1, 1);    glVertex2d(SCREENRIGHT, SCREENBOTTOM);
57     glTexCoord2d(1, 0);    glVertex2d(SCREENRIGHT, SCREENTOP);
58
59     glEnd();
60
61     glDisable(GL_TEXTURE_2D);
62 }
63
64 void MainMenu::drawClickBoxes()
65 {
66     glColor3d(1, 0, 0);
67
68     // Start a new game
69     glBegin(GL_LINE_LOOP);
70     glVertex2d(SCREENRIGHT / 20.0, SCREENBOTTOM / 2.2);
71     glVertex2d(SCREENRIGHT / 20.0, SCREENBOTTOM / 1.9);
72     glVertex2d(SCREENRIGHT / 3.0, SCREENBOTTOM / 1.9);
73     glVertex2d(SCREENRIGHT / 3.0, SCREENBOTTOM / 2.2);
74     glEnd();

```

```

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

```

```

130                                     //
131                                     }
132     }
133
134     // Exit
135     /*
136     if (x >= SCREENRIGHT / 20.0 && x <= SCREENRIGHT / 3.0)
137     {
138         if (y >= SCREENBOTTOM / 2.2 && y <= SCREENBOTTOM / 1.9)
139         {
140             exit(0);
141         }
142     }*/
143 }
144
145 void MainMenu::display()
146 {
147     prepare2D();
148
149     drawMainPic();
150
151     // Disable once finished
152     drawClickBoxes();
153
154     glEnd();
155
156     prepare3D();
157 }

```

3.1.27 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

```

3.1.28 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 }

```

3.1.29 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 functionality
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

```

3.1.30 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 }

```

3.1.31 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

```

3.1.32 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/1223596129 \\
39
40     // Calculate vector equation  $ax + by + cz + d = 0$ 
41     // Get two vectors from three of the corners
42     double AB[] = { vertices[3] - vertices[0], vertices[4] - vertices[1],
        vertices[5] - vertices[2] };
43     double AC[] = { vertices[6] - vertices[0], vertices[7] - vertices[1],
        vertices[8] - vertices[2] };
44     // Cross Product of AB and AC
45     a = (AB[1] * AC[2]) - (AB[2] * AC[1]);
46     b = (AB[2] * AC[0]) - (AB[0] * AC[2]);
47     c = (AB[0] * AC[1]) - (AB[1] * AC[0]);
48     d = (a * vertices[0] + b * vertices[1] + c * vertices[2]);
49
50     axis = _axis;
51 }
52
53 Plane::Plane(const double(&new_vertices)[12], const double(&new_color)[4])
54 {
55     // Copies the color
56     copy(begin(new_color), end(new_color), color);
57
58     // Copies the vertices
59     copy(begin(new_vertices), end(new_vertices), vertices);
60
61
62     // Somedays I wonder what I'm even doing \\
63     // When I forget what all this means: http://keisan.casio.com/exec/system/1223596129 \\
64
65     // Calculate vector equation  $ax + by + cz + d = 0$ 
66     // Get two vectors from three of the corners
67     double AB[] = { vertices[3] - vertices[0], vertices[4] - vertices[1],
        vertices[5] - vertices[2] };
68     double AC[] = { vertices[6] - vertices[0], vertices[7] - vertices[1],
        vertices[8] - vertices[2] };
69     // Cross Product of AB and AC

```

```

70     a = (AB[1] * AC[2]) - (AB[2] * AC[1]);
71     b = (AB[2] * AC[0]) - (AB[0] * AC[2]);
72     c = (AB[0] * AC[1]) - (AB[1] * AC[0]);
73     d = (a * vertices[0] + b * vertices[1] + c * vertices[2]);
74
75     axis = 0;
76 }
77
78 void Plane::Display()
79 {
80     // 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.y >= minY);
123     }

```



```

124         else if (axis == 'z')
125         {
126             vector<double> Z = { vertices[2], vertices[5], vertices[8],
                                vertices[11] };
127             double maxZ = *max_element(Z.begin(), Z.end());
128             double minZ = *min_element(Z.begin(), Z.end());
129
130             return (-Cam.z <= maxZ && -Cam.z >= minZ);
131         }
132         else return false;
133     }
134
135     double Plane::getNorm()
136     {
137         return sqrt(a * a + b * b + c * c);
138     }

```

3.1.33 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 #define FILE_NOT_FOUND 8
24
25 #endif

```

3.1.34 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 #define FILE_NOT_FOUND 8
24
25 #endif

```

3.1.35 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();
37     // Sets global variables to load game
38     bool loadGame();
39     // Returns the decrypted string in core.sav

```

```

40         std::string readSave();
41         // Returns true if core.save exists
42         bool checkSave();
43     };
44
45 #endif

```

3.1.36 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 #include "Globals.h"
18
19 #include "Logger.h"
20
21 using namespace std;
22
23 SaveManager::SaveManager()
24 {
25     HRESULT ret = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL,
26     SHGFP_TYPE_CURRENT, CHAR_PATH);
27     SAVE_PATH = CHAR_PATH;
28     SAVE_PATH += "\\The God Core\\core.sav";
29 }
30
31 string SaveManager::encryptData(string data)
32 {
33     string ret_str;
34     for (unsigned int i = 0; i < data.length()*3; i+=3)
35     {
36         ret_str += data[i/3] + 48;
37         ret_str += data[i/3] - 48;
38         ret_str += data[i/3] + 53;
39     }
40     return ret_str;
41 }
42
43 string SaveManager::decryptData(string data)
44 {
45     string ret_str;
46     for (unsigned int i = 0; i < data.length(); i+=3)
47     {
48         ret_str += data[i] - 48;

```

```

48         }
49
50         return ret_str;
51     }
52
53     string SaveManager::readSave()
54     {
55         Logger log;
56
57         ifstream save(SAVE_PATH);
58         log.logLine("Checking Save integrity.");
59
60         string enc_data; // Encrypted Data
61         string dcr_data; // Decrypted Data
62         save >> enc_data; // Read encrypted data from file
63         dcr_data = decryptData(enc_data); // Decrypt data
64
65         vector<string> output{ "Decrypted Data: ", dcr_data };
66         log.logLine(output);
67
68         save.close();
69
70         return dcr_data;
71     }
72
73     void SaveManager::saveLevel()
74     {
75         ofstream save(SAVE_PATH);
76
77         string input = curr_level + " " + to_string(songNum);
78
79         string encr_str = encrytData(input);
80
81         save << encr_str;
82
83         save.close();
84     }
85
86     bool SaveManager::loadGame()
87     {
88         // might change to vector<string> later
89         string data = readSave();
90         size_t pos = data.find(' ');
91
92         if (pos == string::npos) return false;
93         string savedLevel = data.substr(0, pos);
94         int savedSong = stoi(data.substr(pos + 1));
95
96         int temp_levelNum = getLevelNum(savedLevel);
97
98         if (temp_levelNum == -1) return false;
99
100         levelNum = temp_levelNum;
101         curr_level = getLevelString(levelNum);
102         songNum = savedSong;
103

```

```

104         loading = true;
105         changeSong = true;
106
107         return true;
108     }
109
110     bool SaveManager::checkSave()
111     {
112         ifstream save(SAVE_PATH);
113
114         if (save)
115         {
116             return true;
117         }
118
119         else
120         {
121             return false;
122         }
123     }

```

3.1.37 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 #include "PoweredObject.h"
18 // Terminal Class
19 #include "Terminal.h"
20
21 // Types
22 #include "GCTypes.h"
23
24 class Switch : public PoweredObject
25 {
26 private:
27     void* target; // The door that this switch activates
28     // Translation and rotation coordinates
29     double translate[3], rotate[3];
30
31     // One of the predefined types
32     GCType targetType;
33
34     std::string id;

```

```

35
36 public:
37     // Initializes the translation and rotation matrices
38     Switch(const double(&_translate)[3], const double(&_rotate)[3], GCtype
        _type, std::string _id, bool _isOn);
39     // Binds the target pointer to an object
40     void assign(void* _target);
41     // Opens/Closes the door
42     void toggleTarget();
43     // Actually draws the switch
44     void Display();
45
46     std::string getID();
47
48     // Gets the translation coordinates
49     double getX();
50     double getY();
51     double getZ();
52 };
53
54 #endif

```

3.1.38 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 Switch.h *
9  \*****/
10
11 // Class decleration
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], GCtype
    _type, string _id, bool _isOn)
27 {
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     if (_isOn) activate();
41     else deactivate();
42
43 }
44
45 void Switch::assign(void* _target)
46 {
47     target = _target;
48 }
49
50 void Switch::toggleTarget()
51 {
52     switch (targetType)
53     {
54         case T_DOOR:
55         {
56             Door* t = (Door*)target;
57             t->isOpen = !t->isOpen;
58             break;
59         }
60         case T_TERMINAL:
61         {
62             Terminal* t = (Terminal*)target;
63             t->toggle();
64             break;
65         }
66         case T_LEVEL_END:
67         {
68             levelNum++;
69             curr_level = getLevelString(levelNum);
70             loading = true;
71
72             // TEMP
73             songNum++;
74             changeSong = true;
75         }
76     }
77 }
78
79 void Switch::Display()
80 {
81     glPushMatrix();
82     glTranslated(translate[0], translate[1], translate[2]);
83     glRotated(rotate[0], 1, 0, 0);
84     glRotated(rotate[1], 0, 1, 0);
85     glRotated(rotate[2], 0, 0, 1);
86
87     glColor3d(0.9, 0.9, 0.9);
88     glutSolidCube(.1);

```

```

89
90     switch (targetType)
91     {
92     case T_DOOR:
93         glColor3d(0, 1, 0);
94         break;
95     case T_TERMINAL:
96         glColor3d(1, 0, 0);
97         break;
98     default:
99         glColor3d(0, 0, 1);
100    }
101
102    // If powered off, recolor to black
103    if (!checkIfOn()) glColor3d(0, 0, 0);
104
105    glScaled(.5, .5, 1.5);
106    glutSolidCube(.1);
107
108    glPopMatrix();
109 }
110
111 string Switch::getID()
112 {
113     return id;
114 }
115
116 double Switch::getX()
117 {
118     return translate[0];
119 }
120
121 double Switch::getY()
122 {
123     return translate[1];
124 }
125
126 double Switch::getZ()
127 {
128     return translate[2];
129 }

```

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

```

```

69         // Draws the 2D Terminal screen
70         void DisplayScreen();
71         // Shows the currently typed string
72         void getText(std::string text);
73         // Signifies a completed string to process
74         void getInput(std::string text);
75         // Returns an item in the terminal's log
76         std::string getHist(int count);
77         // Returns the number of items in the terminal's log
78         int getHistNum();
79
80         // Gets the translation coordinates
81         double getX();
82         double getY();
83         double getZ();
84
85         std::string getID();
86
87         Terminal(const double(&_translate)[3], const double(&_rotate)[3], std::
            string _file, std::string _id);
88
89     };
90
91 #endif

```

3.1.40 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 // Logger
28 #include "Logger.h"
29
30 // File I/O

```

```

31 #include <fstream>
32
33 using namespace std;
34
35 const char* Terminal::TERM_PATH = "Resources\\Text\\";
36
37 void Terminal::getText(std::string text)
38 {
39     currentText = text;
40 }
41
42 void Terminal::getInput(std::string text)
43 {
44     currentInput = text;
45 }
46
47 string Terminal::getHist(int count)
48 {
49     int size = history.size();
50     if (history.empty())
51     {
52         return "";
53     }
54
55     // If, somehow, a fool manages to get a variable that is out of bounds
56
57     else if (count >= size)
58     {
59         return history.back();
60     }
61
62     else if (count < 0)
63     {
64         return history.front();
65     }
66
67     else
68     {
69         return history[size - count - 1];
70     }
71 }
72
73 int Terminal::getHistNum()
74 {
75     return history.size();
76 }
77
78 void Terminal::draw()
79 {
80     // Completely black background
81     double colors[4] = { 0, 0, 0, 1 };
82     double vertices[12] =
83     {
84         SCREENLEFT, SCREENTOP, -1,
85         SCREENLEFT, SCREENBOTTOM, -1,
86         SCREENRIGHT, SCREENBOTTOM, -1,

```

```

87         SCREENRIGHT, SCREENTOP, -1
88     };
89
90     Plane background{ vertices, colors};
91     background.Display2D();
92
93
94     // Gotta do the banner manually
95     glEnable(GL_TEXTURE_2D);
96
97     glBindTexture(GL_TEXTURE_2D, bTexture); // Prepares the texture for usage
98
99     glColor3d(1, 1, 1);
100    glBegin(GL_QUADS);
101    glTexCoord2d(0, 0);    glVertex2d(SCREENLEFT, SCREENTOP);
102    glTexCoord2d(0, 1);    glVertex2d(SCREENLEFT, SCREENBOTTOM / 9.0);
103    glTexCoord2d(1, 1);    glVertex2d(SCREENRIGHT, SCREENBOTTOM / 9.0);
104    glTexCoord2d(1, 0);    glVertex2d(SCREENRIGHT, SCREENTOP);
105
106    glEnd();
107
108    glDisable(GL_TEXTURE_2D);
109 }
110
111 void Terminal::DisplayScreen()
112 {
113     prepare2D();
114
115     draw();
116
117     // If we need to proces a command
118     if (currentInput != "")
119     {
120         processInput();
121
122         history.push_back(currentInput);
123
124         currentInput.clear();
125     }
126
127     else
128     {
129         // Print all prompts
130         for (unsigned int i = 0; i < prompts.size(); i++)
131         {
132             text.printString(SCREENLEFT, PROMPT_START + 15 * i, 0, 1,
133                             0, prompts[i]);
134
135             // Print an error
136             text.printString(SCREENLEFT, ERROR_LINE, 1, 0, 0, error);
137             // Echo user text
138             text.printString(SCREENLEFT, INPUT_LINE, 0, 1, 0, ":> " +
139                             currentText);
140
141             // If needed, print content

```

```

141         if (num != -1 && num < (signed int)content.size())
142         {
143             text.openFile(SCREENLEFT, CONTENT_START, 0, 1, 0, file,
                           content[num]);
144         }
145     }
146
147     prepare3D();
148 }
149
150 void Terminal::processInput()
151 {
152     error = "";
153     if (currentInput == "exit" || currentInput == "Exit")
154     {
155         isInTerminal = false;
156         history.clear();
157     }
158
159     else if (currentInput == "clear" || currentInput == "Clear")
160     {
161         num = -1;
162     }
163
164     else if (currentInput == "help" || currentInput == "Help")
165     {
166         num = 0;
167     }
168
169     else
170     {
171         string first, last;
172         size_t pos = currentInput.find(" ");
173
174         first = currentInput.substr(0, pos); // First half of string
175         last = currentInput.substr(pos + 1); // Second half of string
176
177         if (first == "read" || first == "Read")
178         {
179             num = atoi(last.c_str());
180             if (num <= 0 || num >= (signed int)prompts.size())
181             {
182                 error = "ERROR: Invalid file number";
183                 num = -1;
184             }
185         }
186
187         else
188         {
189             error = "ERROR: Invalid Command: " + currentInput;
190             num = -1;
191         }
192     }
193 }
194
195 void Terminal::Display()

```

```

196 {
197     // Add two styles - Standing and wall mounted
198     glPushMatrix();
199
200     // Initial Positioning and rotation
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     //drawWallMounted();
207     drawStanding();
208
209     glPopMatrix();
210 }
211
212 void Terminal::drawStanding()
213 {
214     // Steel grey
215     glColor3d(.1, .1, .1);
216
217     // Draw Floor mount
218     glPushMatrix();
219     glTranslated(0, -1, 0);
220     glScaled(.5, .1, 1);
221     glutSolidCube(.5);
222     glPopMatrix();
223
224     // Draw leg
225     glPushMatrix();
226     glTranslated(0, -.6, 0);
227     glScaled(.1, .75, .1);
228     glutSolidCube(1);
229     glPopMatrix();
230
231     // Draw Monitor
232     glPushMatrix();
233     glScaled(.1, .5, .7);
234     glutSolidCube(1);
235
236     // Draw Screen
237     glPushMatrix();
238     // Change Screen based on power
239     if (checkIfOn())
240         glColor3d(0, 1, 1);
241     else
242         glColor3d(0, 0, 0);
243
244     glTranslated(-.3, 0, 0);
245     glutSolidCube(.7);
246
247     glPopMatrix();
248
249     glPopMatrix();
250 }
251

```

```

252 void Terminal::drawWallMounted()
253 {
254     glColor3d(0, 1, 1);
255     glutSolidSphere(1, 50, 50);
256 }
257
258 double Terminal::getX()
259 {
260     return translate[0];
261 }
262
263 double Terminal::getY()
264 {
265     return translate[1];
266 }
267
268 double Terminal::getZ()
269 {
270     return translate[2];
271 }
272
273 void Terminal::parseFile()
274 {
275     ifstream infile{ TERM_PATH + file};
276     string buff;
277
278     if (!infile)
279     {
280         Logger log;
281         vector<string> output = { "FATAL ERROR: File ", file, " NOT FOUND"
282                                 };
283         log.logLine(output);
284         exit(FILE_NOT_FOUND);
285     }
286
287     content.push_back("HELP"); // Help text is always the 0th tag in the
288                                // terminals
289
290     getline(infile, buff);
291     prompts.push_back(buff); // Push back the file tag
292     getline(infile, buff);
293
294     while (buff != "<TAGS>")
295     {
296         size_t pos = buff.find("--");
297         if (pos != string::npos)
298         {
299             prompts.push_back(buff.substr(0, pos));
300             content.push_back(buff.substr(pos + 3));
301         }
302         getline(infile, buff);
303     }
304
305 string Terminal::getID()

```

```

306 {
307     return id;
308 }
309
310 Terminal::Terminal(const double(&_translate)[3], const double(&_rotate)[3], string
    _file, string _id)
311 {
312     // Copies the color
313     copy(begin(_translate), end(_translate), translate);
314
315     // Copies the vertices
316     copy(begin(_rotate), end(_rotate), rotate);
317
318     bTexture = SOIL_load_OGL_texture
319         (
320             "Resources\\Images\\banner.png",    // Image to load
321             SOIL_LOAD_AUTO,                      // ???
322             SOIL_CREATE_NEW_ID,
323             SOIL_FLAG_MIPMAPS | SOIL_FLAG_COMPRESS_TO_DXT // !?!?!?!
324         );
325
326     if (bTexture == 0)
327     {
328         Logger log;
329         vector<string> output = { "FATAL ERROR: SOIL cannot load terminal
            banner", SOIL_last_result() };
330         log.logLine(output);
331         exit(SOIL_ERROR);
332     }
333
334     file = _file;
335
336     id = _id;
337
338     num = 0;
339
340     parseFile();
341 }

```

3.1.41 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

```

3.1.42 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 = 15;
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 }

```

3.1.43 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

```

3.1.44 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 }

```

3.1.45 Trigger.h

```

1  /*****\
2  * Trigger.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 Trigger class *
8  * Which can be bound to a trigger-object that, upon use, *
9  * Will activate a designated target-object. *
10 \*****/
11
12 #ifndef TRIGGER_H
13 #define TRIGGER_H
14
15 #include "Terminal.h"
16 #include "Switch.h"
17
18 #include "GCTypes.h"
19
20 class Trigger
21 {
22 private:
23     void* trigger; // The object that activates the target
24     void* target; // The object that is activated by the target
25
26     GCType triggerType; // The type (defined from GCTypes.h) of the trigger
27     GCType targetType; // The type (defined from GCTypes.h) of the target
28
29     void activateTarget();
30

```

```

31 public:
32     // Get the object type of the trigger
33     int getTriggerType();
34     // Attempts to trigger the target
35     bool tryToTrigger(void* input, Gctype type);
36     // Binds the triggering object
37     void bindTrigger(void* _trigger);
38     // Binds the target object
39     void bindTarget(void* _target);
40     // Constructor takes in trigger type and target type
41     Trigger(Gctype _triggerType, Gctype _targetType);
42
43 };
44
45 #endif

```

3.1.46 Trigger.cpp

```

1  /*****\
2  * Trigger.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 Trigger class *
8  * For more information, see Trigger.h *
9  \*****/
10
11 #include <cstdlib>
12 #include "Trigger.h"
13
14 int Trigger::getTriggerType()
15 {
16     return triggerType;
17 }
18
19 void Trigger::activateTarget()
20 {
21     switch (targetType)
22     {
23         case T_TERMINAL:
24         {
25             Terminal* t = (Terminal*)target;
26             t->activate();
27             break;
28         }
29         case T_SWITCH:
30         {
31             Switch* s = (Switch*)target;
32             s->activate();
33             break;
34         }
35         default:
36         {
37             break;
38         }
39     }

```

```

40 }
41
42 bool Trigger::tryToTrigger(void* input, GCtype type)
43 {
44     // If this trigger is the correct type
45     if (triggerType != type) return false;
46
47     // If this trigger is the correct object
48     if (trigger != input) return false;
49
50     activateTarget();
51
52     return true;
53 }
54
55 void Trigger::bindTrigger(void* _trigger)
56 {
57     trigger = _trigger;
58 }
59
60 void Trigger::bindTarget(void* _target)
61 {
62     target = _target;
63 }
64
65 Trigger::Trigger(GCtype _triggerType, GCtype _targetType)
66 {
67     trigger = NULL;
68     target = NULL;
69     triggerType = _triggerType;
70     targetType = _targetType;
71 }

```

3.1.47 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

3.1.48 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 }
```

3.1.49 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 class TwoD
16 {
17 protected:
18     // The pixel boundaries of the screen
19     const double SCREENTOP = 0, SCREENBOTTOM = 1080,
20             SCREENLEFT = 0, SCREENRIGHT = 1920;
21
22     // Prepares OpenGL draw in 2D
23     void prepare2D();
24
25     // "Resets" OpenGL to draw in 3D
26     void prepare3D();
27
28 };
29
```


30 #endif

3.1.50 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 depth testing
19     glDisable(GL_DEPTH_TEST);
20     // Disable writing to the z buffer
21     glDepthMask(GL_FALSE);
22     // Disables lighting
23     glDisable(GL_LIGHTING);
24
25     // Create an orthogonal matrix to write on
26     glMatrixMode(GL_PROJECTION);
27     glPushMatrix();
28     glLoadIdentity();
29     glOrtho(SCREENLEFT, SCREENRIGHT, SCREENBOTTOM, SCREENTOP, -1, 1);
30     glMatrixMode(GL_MODELVIEW);
31     glPushMatrix();
32     glLoadIdentity();
33 }
34
35 void TwoD::prepare3D()
36 {
37     // Discards the orthogonal matrices
38     glMatrixMode(GL_PROJECTION);
39     glPopMatrix();
40     glMatrixMode(GL_MODELVIEW);
41     glPopMatrix();
42
43     // Enable depth testing
44     glEnable(GL_DEPTH_TEST);
45     // Enables writing to the z buffer
46     glDepthMask(GL_TRUE);
47     // Renable lighting
48     glEnable(GL_LIGHTING);
49 }
```

3.2 Database

3.2.1 Walls

#	ID	LEVEL	X1	X2	X3	X4	Y1	Y2	Y3	Y4	Z1	Z2	Z3	Z4	R	G	B	A	Axis
1	lvcelling	LEVELZERO	-5	-5	8	8	1	1	1	1	-4	1	1	-4	0.7	0.7	0.7	1	0
2	lvffloor	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	room0rghtwall	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

3.2.2 Doors

3.2.3 Switches

3.2.4 Terminals

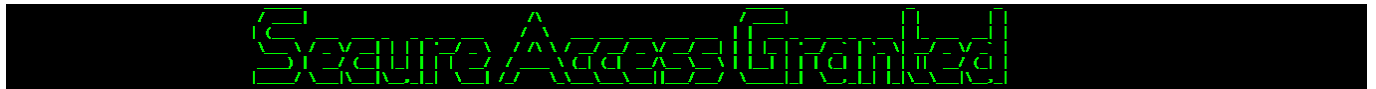
3.2.5 Triggers

3.3 Images

3.3.1 Main Menu



3.3.2 Terminal Banner



3.3.3 Game Icon



3.4 Music