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

Author: Jeremy Greenburg
Mentor: Dr. Joshua Guerin
Second Reader: Bob Bradley

March 24, 2017

Contents

1 Abstract **3**

2 Development Tools **4**
2.1 APIs 4
2.2 Development Environment 5

3 The Project **5**
3.1 The Game Engine 5
3.2 The Game 11

4 Challenges and Future Considerations **12**
4.1 Challenges that I Faced 12
4.2 challenges 12
4.3 Future Considerations 13

5 Acknowledgments **13**

6 Appendices **13**
6.1 Source Code 13
6.2 Database 113
6.3 Images 119
6.4 Music 120

1 Abstract

This project consists of a video game engine developed in C++ and a short video game developed on it. My goal throughout this project was to strengthen my software development skills and prepare me for a career. I developed skills not necessarily part of an ordinary Computer Science curriculum, such as the research, evaluation, and implementation of various APIs, creating a deployment module, and simply developing and maintaining a project for an extended amount of time. Developing this project also served to strengthen the programming principles that have been instilled in me throughout my undergraduate Computer Science experience.

2 Development Tools

2.1 APIs

API's, or *Application Programmer Interfaces*, are a set of methods and tools to allow a programmer to access a piece of software through code. They are useful when developing more complex applications, as you can incorporate useful, quality tools rather than spending time developing tools that have already been created.

2.1.1 OpenGL

OpenGL (Open Graphics Library) is one of the most widely used graphics libraries available. It provides access to matrix manipulation, keyboard and mouse input, windowing, and vector graphics. It provides the ability to draw in both 2D and 3D and gives access to primitives such as rectangles, triangles, and lines. With GLUT (OpenGL Utility Toolkit), OpenGL can also draw simple spheres and cylinders. It is the graphical backbone of both the Unity and Unreal Engines for Mac OS and Linux.

I chose to use OpenGL over its stronger competitor, Microsoft's DirectX, because it is cross platform which would reduce the amount of work needed to port the engine to a different operating system, and because there is a great deal of documentation easily available for OpenGL.

2.1.2 SOIL

SOIL (Simple OpenGL Interface Library), is a small extension to OpenGL that provides an easy to use interface for using textures in OpenGL, including saving images, loading and binding textures, and resizes textures. A *texture* is an image on the hard disk, such as a JPG or PNG file, that is loaded into memory and rendered over an OpenGL primitive, such as a rectangle. I use textures for the main menu and as part of the background for Terminals to make them look nicer, everything else rendered in game is an OpenGL or glut primitive.

2.1.3 FMOD

FMOD is a sound effects engine developed by Firelight Technologies that can play many different files types on numerous Operating Systems including but not limited to: Windows, OSX, IOS, Playstations and Xboxes, and Android; and it is is the primary audio system for many game engines including Unity, Unreal, CryEngine, and Havok. I decided to use FMOD because I was impressed with its flexibility and diversity, no other API that I looked at could read as many different sound files, particularly MP3 files which was the format of the sound files that I had acquired.

2.1.4 SQlite

Rather than store game data in a text file, I chose to store the data in a SQL database to use make full use of the SQL queries, which make it easy to request all data for specific levels and to parse the data that is received.

I decided to use SQLite over other implementations of SQL because it is a lightweight and simplified, stripping out features of SQL that I do not need in order to make queries faster and the database size smaller.

2.1.5 Windows API

The Windows API is distrubted with the Microsoft Software Development Kit and provides access to many features of the Windows operating system.

The game engine only utilizes one of its eight modules, the Shell Object module, which gives access to the operating system shell. Since programs do not have write permissions to their install folder in Windows, the Shell Object module can be used to used to locate the user's personal documents folder, where both the save file and the log file are written.

2.2 Development Environment

2.2.1 Microsoft Visual Studio

Microsoft Visual Studio is an IDE (Integrated Development Environment) developed for Windows that supports a variety of programming languages, and it is where I wrote all of the code for the game engine. I chose Visual Studio as my IDE because it give access to an Installer package, which allows me to create a windows installer for my game so that it can be installed on any Windows computer. The Installer packages together the executable source code, any resources that I have developed, as well as system resources such as the Microsoft C++ redistributes necessary to run programs developed in Visual Studio. Visual Studio also provides powerful analytic tools to monitor memory usage to avoid creating memory leaks.

I chose to develop my project in C++ because it is an extremely fast language, and speed is one of the most important qualities in real time software like video games. C++ also allows low level memory access with pointers and the ability to use typeless pointers, which are of great use in section **??**.

2.2.2 SQLite Studio

SQLite Studio is a third party GUI for SQLite that allows information to be analyzed and edited quickly and easily. It also provides utilities to export a database to a number of different file formats, allowing me to easily include the database for reference in section 6.2.

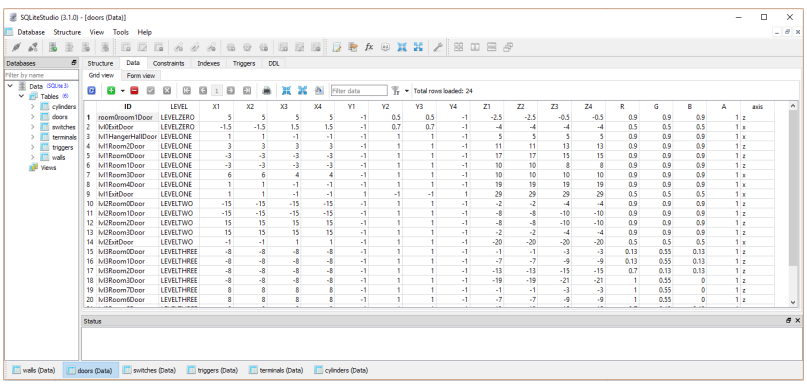


Figure 1: A picture of the *doors* table in the SQLite Studio editor.

2.2.3 GitHub

GitHub is an online Git repository that houses many open source projects. It also provides a great source control and branching system, where multiple branches can be created from a single point in the project to experiment with new features without any fear of damaging your code if they do not work out.

3 The Project

The project consists of two entities: the game engine and the game itself, which was developed on the engine. The engine and game are not two fully seperated entities as discussed in section 4.3.2.

3.1 The Game Engine

I developed the engine of my game in C++ during two, years starting in my second semester sophomore year and ending the first semester of my senior year. It consists of 49 C++ files, in which there are 3,308 lines of code and 1,122 lines of comments. The code can be found in the section 6.1, and it as also located on GitHub at *https://github.com/Jerrgree/The-God-Core-Source*.

The engine reads a SQLite database (Data.db) that is housed in the same directory as the game executable, and it recognizes six tables in the database that correspond to six different types of in game objects—Walls, Doors, Cylinders, Terminals, Switches, and Triggers. Due to limitations discussed in 4.3.2, it will only properly work with a game that has five levels.

3.1.1 Walls and Doors

Walls are, at their heart, an OpenGL rectangle with a wrapper for additional functionality. In the same vein, Doors are simply Walls with the ability to open and close.

Internally they consist of two arrays: a four dimensional array containing the *rgba* values and a twelve dimensional array containing the 4 *xyz* coordinates of the rectangle's corners.

The rectangles contain the most complex mathematics needed for collision, as the necessary calculus to correctly determine whether or not the player has collided with a rectangle involves determining if a sphere has collided with a plane in 3 Dimensional space.

In the constructor of a rectangle, after all values are initialized, the equation of the plane (Figure 2) is immediately calculated for future reference in collision detection.

$$aX + bY + cZ + d = 0$$

Figure 2: The equation of a plane.

This equation is calculated using the any three corners of the rectangle (A, B, and C) and then creating two vectors (B-A and C-a) using their dot product as show in Figure 3.

$$\vec{AB} = \begin{vmatrix} Bx - Ax \\ By - Ay \\ Bz - Az \end{vmatrix} \vec{AC} = \begin{vmatrix} Cx - Ax \\ Cy - Ay \\ Cz - Az \end{vmatrix}$$
$$a = \vec{AB}_2 * \vec{AC}_3 - \vec{AB}_3 * \vec{AC}_2$$
$$b = \vec{AB}_3 * \vec{AC}_1 - \vec{AB}_1 * \vec{AC}_3$$
$$c = \vec{AB}_1 * \vec{AC}_2 - \vec{AB}_2 * \vec{AC}_1$$
$$d = -(aAx + bAy + cAz)$$

Figure 3: Given three points of a rectangle (A, B, and C), the equation of the plane can be derived with the dot product of two vectors. [4]

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

3.1.2 Terminals

Terminals are an in-game computer that the player can access to read parts of the stories lore, as well as unlock new doors for them to explore.

Each terminal is bound to a unique *terminal file* that is heavily structured and contains its data. An example is show in Figure 4

```
<FILES>
[01] Name1 -- TAG
[02] Name2 -- TAG2
[03] Name3 -- TAG3

<TAGS>
$HELP$
Type Read <num> to read the corresponding file
Type Clear to clear a file from the screen
Type Exit to exit the terminal
Type Help to see this message again
$END$

$TAG$
Content 1
$END$

$TAG2$
Content 2
$END$

$TAG3$
Content 3
$END$
```

Figure 4: An example Terminal file

The program parses the file by first separating the in game content (the bracketed number and name) that should be displayed to the user from it's tag. The tags are stored in an array, where its index is equal to the bracketed number. The help display is always stored at the 0th index.

The terminal recognizes a number of commands.

- Help—Displays the help prompt.
- Read X—Reads the requested tag. If X is zero or greater than the highest number, an error is returned.
- Quit or Exit—Removes the player from the terminal and back into the world.

The terminal also stores a history of what the player types, and the up and down arrows can be used to cycle through previous commands.

3.1.3 Switches

A switch is a button that is attached to a wall and is visible on either side. Switches are primarily bound to doors and terminals and are used to open/close a door or power on/off a terminal. Switches are also the mechanism to change levels, each level except for the last contains a switch that, when activated, will trigger a level change. Switches serve as the primary means of progress, as the level change switch and many door switches will initially be off, and the player must navigate through the level to power on more switches and progress through the level.

Internally a switch consists of two three dimension vectors, one containing its xyz center and the other containing its xyz rotation. It also contains a void pointer to its target and an identifier as to what type of object the target is so that the pointer can be properly typecast.

3.1.4 Triggers

Triggers are not a tangible object in the game, rather, they serve as the description of an event. Triggers are a more sophisticated form of interaction between two different objects. The implementation was designed to be abstracted away from object types so that any arbitrary object could activate another.

The trigger holds two void pointers, one for a triggering object and one for the target object, as well as identifiers for which object type they are. Whenever an object is interacted with, every trigger in the game is tested and if the object is the same as the trigger pointer (no referencing needed as the pointers will always be equal), the target is dereferenced according to the appropriate type and activated. For the differences between triggers and switches, see section 4.3.1.

3.1.5 Cylinders

Cylinders solely exist as decoration. Internally it contains the radius of each base, the height, an xyz center, the $rgba$ values, and the number of "slices" or how smooth it should appear.

3.1.6 Camera Controller

The camera control describes how the player looks around and moves. Internally it contains the xyz rotation, which describes the direction that the player is looking, and the xyz coordinates where the player is physically located. The x rotation corresponds to left/right movement, the y coordinate refers to up/down movement, and z rotation would be similar to a barrel roll.

The player can move forwards and backwards, as well as strafe left and right, in respect to the direction that they are facing. The equation to determine movement is the formula to determine a point on the circumference of a circle as seen in Figure 5. Only the angle of x rotation is necessary for the formula, as looking left and right are the only things that impact where one would move. Given the formula, it is equally easy to implement other directions of movement by adding and subtracting 90° from the x angle.

z := z $\pm$ moveSpeed * cos(radian(θ))
x := x $\mp$ moveSpeed * sin(radian(θ))

Figure 5: This equation finds an arbitrary point on a circle's circumference in relation to a specified angle. [6]

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

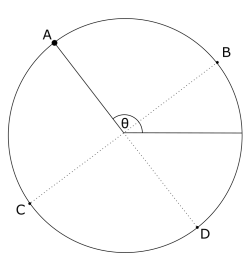


Figure 6: A graphical representation of movement. With the player at the center of the circle and θ = x rotation, point A would represent forward movement, B and C would represent strafing left and right, and point D would represent backwards movement.

3.1.7 Keyboard Controller

The Keyboard class primarily serves to encapsulate the OpenGL callbacks that receive keystrokes: the normal function that accepts all alphanumeric and punctuation keys, and the special function that handles function keys and arrows. The keyboard function acts differently depending on what mode the player is currently in.

Under normal circumstances, the only normal keystrokes accepted are the WASD keys for movement, the E key for interaction, the ' ' key for toggling the development console, and the escape key which will return the player to the main menu.

When in either a terminal or the development console, all keys are immediately concatenated to an input string with the exception of the ' ' which will close the development console, or the enter key which will send the input string to it's appropriate destination to be parsed and interpreted, after which the input string is cleared so that a new command can be entered.

Also accepted are the up and down arrow keys, which will cycle through the console/terminals command history.

When the user is in the main menu, no keyboard keys are accepted other than F2, which will close the game under any circumstances.

3.1.8 Music Controller

To play background music, I created a class that uses the FMOD Low Level API that knows the directory that all sound files are stored and will play a designated one on infinite repeat until the prompted to change songs.

Each song in game is mapped to an integer. On each level change, the song number is incremented and a boolean flag is tripped, which signals the music controller to play the next song. Each song is dynamically allocated, so it is important to properly deallocate the songs before the next one is played. Considerations on how to change the Music Controller are noted in section 4.3.2.

3.1.9 Text Controller

The Text Engine handles displaying all text to the screen, from prompts on the HUD to each Terminal screen. It uses OpenGL's glutBitmapCharacter function to display clear, concise text.

Every function to display text takes the xy coordinates for where on the screen to start printing, and the RGB color values for the text. There are two functions for displaying text, the simpler one merely takes in a string and prints it on the corresponding location on the screen. The more complex function takes in a file and a content tag which needs to be parsed. The text files are similar to terminal files as seen in Figure 7.

```
$TAG 1$
Content 1
$END$

$Tag 2$
Content 2
$END$
```

Figure 7: An example text file, very similar to a terminal file

The Text Engine searches through the designated file line by line until it discovers the line containing the proper tag. Then, until it reaches the closing 'END' tag, it stores every line inside of a vector. Once it has retrieved all of the necessary content, it will print the vector line by line.

3.1.10 Collision Engine

A *collision* occurs when two or more objects in game attempt to occupy the space. The collision engine handles detecting and preventing any collision between the player and all objects in the level. In this engine 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, the collision can be detected easily using the distance formula and the radii of the spheres as seen in Figure 8.

$$\sqrt{(x_2 - x_1) + (y_2 - y_1) + (z_2 - z_1)} < r_2 + r_1$$

Figure 8: If the distance between the sphere is shorter than the combined radii of the spheres, a collision has occurred.

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 and usually encompass the player entirely.

To rectify that, the collision is split into two phases. In the first phase, we use the plane equation that is derived in the section 3.1.1. Using the equation from Figure 9. If the resulting value is less than the radius of the player's bounding sphere, the player is close enough to collide with that plane. However, a plane is an infinite length, and relying solely on this method would induce false collisions if the player was close to the wall but beyond it.

$$\frac{ax+by+cz+d}{\sqrt{a^2+b^2+c^2}} < r$$

Figure 9: By substituting the player's xyz coordinates and dividing by the norm of the plane, we can determine the distance to the plane.

For the second phase, each wall is aligned on an axis: x, y, or z. For the axis that it is aligned on, largest and smallest values of the coordinates are compared to the player's coordinates. palyer's corresponding coordinate is in between the two values, the player has hit the wall. Otherwise, they hit the plane but not the wall, and they do not collide.

3.1.11 2D

As the player's HUD, the main menu, the developer console, and terminals all need to display in a 2D environment, I extracted the ability to draw in 2D into is own class that is inherited by them in order to reduce reused code.

To convert OpenGL into 2D frame, lighting and depth masking must be disabled. Next an *orthogonal* matrix is pushed onto OpenGL's matrix stack using the length and width of the screen so that all matrix transformations corresponded to a pixel on the screen. Re-enabeling 3D is as simple as popping the orthogonal matrix from the stack and re-enabling depth testing and masking.

3.1.12 Level Management

Loading each level involves a series of operations through the SQLite API. First a connection with the database is opened, and then a series of queries are made to the database for each table in the database in turn. All important data from the database is stored in a class of the appropriate type, unnecessary data is discarded, and in the end each class is pushed into a vector of the appropriate type.

The data is loaded in a strict order, due to some objects having dependencies on others (that is, some objects require other objects to already exist). Thus the first things that are loaded are purely independent objects, all doors, walls, cylinders, terminals. Next switches are loaded, because they require both doors and terminals to already exist. Finally, the triggers are loaded, because they require both switches and terminals.

When loading switches and triggers, the objects also need to be *bound* to their appropriate required object(s). This is why doors, switches, and terminals all carry their ID's into the program with them, while triggers and walls discard their ID. Once all of the objects that need to be bound are loaded into the game, the game proceeds to bind them to their target. For each switch that needs to be bound, the game loops through the list of possible target objects and creates a pointer to the correct object inside of the switch, thus ensuring that the switch can toggle its target instantly without needing to search every time it is triggered. The triggers are bound similar, with the difference that each object must perform two searches, one for the triggering object and one for the target object.

If there is any data error in regards to binding — that is, an object attempts to bind to an object that does not exist, the error is considered fatal and the game immediately shuts down after logging the error.

The OpenGL display function calls upon the Level class to display all in game objects. This is a simple matter, because each object knows how to display itself. Thus it is a simple matter to loop through each vector and tell each object to display itself.

3.1.13 Game Saving and Loading

Game saving and loading is a simple file transaction. To save, the current level (as a string) and the song that is playing are appended to each other. The string is then encrypted and written to the save file.

Loading is a two step process. First the contents of the save file are read, decrypted, and parsed into the saved level and the saved song. Next the contents are verified so that they are valid levels to load and songs to play. If either one is invalid, the save file is considered corrupted and the game will refuse to complete the load.

3.1.14 Console and Logging

To aid in debugging a created a Developer Console and a game log. The developer console accepts user commands to perform actions such as writing to the save file, reading the save file, disabling collision, and changing what song is playing.

The logger writes to a log file as the game runs to report on the status of operations, primarily the loading of each level. If an error occurs and the game aborts (without crashing), the appropriate error and error code is written at the end of the log file. There is only ever one log file, which is erased when the game is launched and new data is appended to it as the game runs.

3.2 The Game

The game itself was also developed over a period of two years, starting first semester junior year and ending second semester senior year. The game itself consists of the SQLite database containging the game objects, as well as all text, terminal, image, and sound files that the game engine uses. The game is relatively short, an average play through would take 15-20 minutes, although it can easily be speed-ran in approximately five minutes.

3.2.1 Setting

The game takes place in the year 300x. Humanity has united under British rule, and the New British Empire has turned its eyes to colonizing the stars. The player is Special Constable Rikker, an elite agent of the Empire who has been called to scout out one of the Crown's top secret research facilities, The Daedalus, that has stopped communicating in preparation for a full investigative team that will follow behind them.

3.2.2 Level Zero

Level zero is an introductory tutorial level where the player can get a grasp with the controls. It takes place on the Constable's personal shuttle, and there is a terminal that gives access to the prior information.

3.2.3 Level One

In level one, the player explores one hanger of the ship. There they can read logs detailing power outages and mechanical fluctuations, as well as security and intake reports detailing a few objects that the ship has taken on. The last report details them taking on the titular God Core.

3.2.4 Level Two

The next level takes place in offices, where the Constable learns more about the Daedalus. It exists to house strange, unexplainable, and possibly deadly objects that are too dangerous or unknown to be left alone. Ever since taking on the God Core, crew have been slowly disappearing and communications equipment have started to fail. It comes to light that the God Core warps reality itself, and no one can determine if the missing crew has been transported elsewhere, or if they have simple disappeared from reality.

3.2.5 Level Three

Level three takes place in a gallery housing access to eight different objects. Although the Constable can only access the exhibit containing the God Core, the Constable can read information about the abilities and containment procedures for various objects.

3.2.6 Level Four

The final level takes place in the God Core's exhibit. The player can only stare at the God Core as reality starts to deform around them, before they vanish into the unknown, ending the game.

4 Challenges and Future Considerations

4.1 Challenges that I Faced

4.2 challenges

Of course, the development was not always smooth, and I did encounter plenty of problems during development.

4.2.1 The Death of a Computer

I changed computers twice. I went from a laptop to a more powerful desktop junior year, and senior year my operating system got corrupted and I had to perform a fresh reinstall. Both times I was able to get back up to speed in only a few hours thanks to GitHub.

4.2.2 Clipping Issues

OpenGL There is a problem when getting to close to a wall or a door where the player can see through parts of the wall, no doubt to do OpenGL's depth buffering in relation to distance to the wall. Part of the development processes involved balancing collision distance to the walls, at distances where the user could not see through walls the user was a noticeable distance from the wall, so I chose to allow the user to get close enough to slightly peer through the wall, as the distance felt more natural.

4.2.3 APIs and Naming

C++ uses unique identifiers. This means that no class, variable, or function can share the same name. Initially the Plane class was named Rectangle, as it more clearly showed that it was a wrapper for the OpenGL rectangle primitive, but when I included the Windows API I discovered that it had its own function named Rectangle, rendering my program uncompilable until I renamed every instance of my class.

4.3 Future Considerations

Of course, time itself was a constraining factor, I only had two years to develop the project. As such every design choice that I made bore this limit in mind, and there portions of the engine that I was unable to optimize or implement due to lack of development time. Rarely is a project ever done, and there are always changes and improvements that could be made.

4.3.1 Triggers and Switches

Given time, I would have liked to implement the back end of a switch's interaction into triggers.The primary difference between triggers and switches is that a trigger is a one time activation, whereas a switch is a potentially many time toggle, but it would be a simple matter to add in different trigger types.

4.3.2 Engine and Data Separation

A true game engine is fully separate from the game it runs on, however I had a few roadblocks preventing me from completely separating the two of them.

The music files and maximum song number is hard coded into the engine, as are the maximum number of levels in the game. Music and levels could easily be given their own table in the database after reworking the code that directly deals with them, but the major hurdle that led me to hard coding levels is the lack of animation. I was unable to encapsulate animation in a database, which leads to parts of the fourth level's rendering to be hard coded into the game.

5 Acknowledgments

I would like to offer thanks to the following people:

- Kevin MacLeod, Devin Powers, and Arseniy Shkljaev for the music
- Robert Deyoso III for the movement equation
- Cody Robertson for bouncing ideas off of
- Brian Affolter and Lauren Ball for helping test
- Dr. Joshua Guerin for mentoring the project

6 Appendices

6.1 Source Code

6.1.1 main.cpp

```
/*************************************************************\
 * main.cpp                                                  *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file creates an OpenGL window to display the game    *
 * and promptly passes control over to the GameManager object*
\*************************************************************/

// Because doth openGL demandeth
```

```
#include <cstdlib>
// OpenGL API
#include <GL\glut.h>
// time
#include <ctime>

// The Game manger
#include "GameManager.h"
GameManager Overlord;
// Save manager
#include "SaveManager.h"
// Return codes
#include "Return.h"
// System log
#include "Logger.h"
// Global variables
#include "Globals.h"

// Normal key presses
void normal(unsigned char key, int x, int y);

// For key releases
void key_up(unsigned char key, int x, int y);

// For Special keys
void special(int key, int x, int y);

// Mouse clicks
void mouse(int button, int state, int x, int y);

// Mouse movement
void motionPassive(int x, int y);

// Changing Window size (Not exactly working as hoped...
void changeSize(int w, int h);

// Initializes GLUT callbacks and returns true if core.sav exists (false otherwise
    )
bool initGame(int argc, char **argv);

// Manages the game's scenes
void manageScenes();

GLfloat light_diffuse[] = { 0.3f, 0.3f, 0.3f, 0.3f };
GLfloat light_position[] = { 0.0f, 0.0f, 0.0f, 0.0f }; // Currently nonexistant
    until I can figure out how lighting works
GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat mat_shininess[] = { 75 };
GLfloat lmodel_ambient[] = { 0.6f, 0.6f, 0.6f, 1.0f };

using namespace std;

//****** FUNCTION DEFINITIONS ******\\

int main(int argc, char **argv)
{
```

```
        Overlord.canContinue = initGame(argc, argv);

        // Begin the game
        glutMainLoop();

        // If we ever get here, something bad happened

        Logger log;
        log.logLine("ERROR: GlutMainLoop exited early");

        return EXIT_EARLY;
}

bool initGame(int argc, char **argv)
{
        // Obliderate log file
        Logger log;
        log.nuke();

        // Initialize GLUT
        glutInit(&argc, argv);

        // Create window
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(50, 50);
        glutInitWindowSize(500, 500);
        glutCreateWindow("The God Core");

        // register callbacks
        glutDisplayFunc(manageScenes);
        glutReshapeFunc(changeSize);
        glutIdleFunc(manageScenes);
        glutPassiveMotionFunc(motionPassive);
        glutMouseFunc(mouse);
        glutKeyboardFunc(normal);
        glutKeyboardUpFunc(key_up);
        glutSpecialFunc(special);

        // Prebuilt function that works transparency
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        // Enable transparency
        glEnable(GL_BLEND);
        // Enable depth buffer
        glEnable(GL_DEPTH_TEST);
        // Let there be light!
        glEnable(GL_LIGHTING);
        // First light source
        glEnable(GL_LIGHT0);

        // Light properties
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
        glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);

        // Light doesnt turn everything grey
```

```
        glEnable(GL_COLOR_MATERIAL);

        glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);

        glutWarpPointer(300, 300);

        // Start in Fullscreen
        glutFullScreen();

        srand(time(NULL));

        HUD.setStatus("INFO-WELL");

        SaveManager SaveSystem;
        return SaveSystem.checkSave();
}

// Everything below here is just passed along to the overlord

void mouse(int button, int state, int x, int y)
{
        Overlord.mouse(button, state, x, y);
}

void motionPassive(int x, int y)
{
        Overlord.motionPassive(x, y);
}

void changeSize(int w, int h)
{
        Overlord.changeSize(w, h);
}

void manageScenes()
{
        Overlord.manageScenes();
}

void normal(unsigned char key, int x, int y)
{
        Overlord.normal(key, x, y);
}

void key_up(unsigned char key, int x, int y)
{
        Overlord.key_up(key, x, y);
}

void special(int key, int x, int y)
{
        Overlord.special(key, x, y);
}
```

6.1.2 CameraControl.h

```
/*************************************************************\
 * CameraControl.h                                           *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the CameraControl   *
 * Class, which stores:                                      *
 *      The x, y, z ordered triple of the player's location  *
 *      The degree to which the player is turned, along      *
 *           the x, y, and z axes                            *
 * And contains methods to translate the player along        *
 * 3D space                                                  *
\*************************************************************/

#ifndef CAMERA_CONTROL_H
#define CAMERA_CONTROL_H

class CameraControl
{
private:
        // Speeds for moving and rotating
        double moveSpeed = 0.1f, turnSpeed = 0.5f;

public:
        // Negatively adjusts angle and modifies lx
        void lookLeft();
        // Positively adjusts angle and modifies lx
        void lookRight();
        // Positively adjusts angle and modifies ly
        void lookUp();
        // Negatively adjusts angle and modifies ly
        void lookDown();
        // Translate the camera to the left
        void strafeLeft();
        // Translates the to the right
        void strafeRight();
        // Translates the camera forwards
        void moveForward(int mod);
        // Translate the camera backards
        void moveBackward(int mod);
        // Moves the camera positively along the Y axis
        void moveUp();
        // Moves the camera negatively along the Z axis
        void moveDown();
        // Flips the camera
        void invertCam();
        // If the player begins to run
        void increaseSpeed();
        // If the player begins to walk
        void decreaseSpeed();
        // Resets the camera to it's initial state
        void resetCam();
        // calls gluLookAt
```

```
        void Display();

        // Location of the camera
        double x =0.0, y = 0.0, z = -1.0;
        double prevx, prevz;
        // Angles of rotation
        double x_angle = 0.0, y_angle = 0.0, z_angle = -1.0;
};

#endif
```

6.1.3 CameraControl.cpp

```
/*************************************************************\
 * CameraControl.cpp                                          *
 * This file was created by Jeremy Greenburg                  *
 * As part of The God Core game for the University of         *
 * Tennessee at Martin's University Scholars Organization     *
 *                                                            *
 * This file contains the definition of the CameraControl     *
 * Class. For more information, see CameraControl.h           *
\*************************************************************/

// Class definition
#include "CameraControl.h"

// For sin()
#include <cmath>

// glut is unhappy when cstdlib isn't here :/
#include <cstdlib>

// OpenGL API
#include <GL\glut.h>

// To display Suit Warnings
#include "TextEngine.h"

// To include Globals Variables
#include "Globals.h"

// For converting degrees to radians
const double PI = 3.14159;

// Takes in an angle, in degrees, and returns the angle in radians
double toRadian(double angle)
{
        return angle * PI / 180;
}

void CameraControl::lookLeft()
{
        x_angle -= 3 * turnSpeed;

        // To avoid potential underflow errors
        if (x_angle < 0)
        {
```

```
                x_angle += 360;
        }
}
void CameraControl::lookRight()
{
        x_angle += 3 * turnSpeed;

        // To avoid potential overflow errors
        if (x_angle > 360)
        {
                x_angle -= 360;
        }
}

void CameraControl::lookUp()
{
        y_angle -= 2 * turnSpeed;

        // To avoid potential underflow errors
        if (y_angle < 0)
        {
                y_angle += 360;
        }
}

void CameraControl::lookDown()
{
        y_angle += 2 * turnSpeed;

        // To avoid potential overflow errors
        if (y_angle > 360)
        {
                y_angle -= 360;
        }
}

void CameraControl::strafeLeft()
{
        prevz = z;
        prevx = x;
        // Angles + 90 degrees for an angle that is perpendicular to x_angle
        z = z + moveSpeed * cos(toRadian(x_angle + 90));
        x = x - moveSpeed * sin(toRadian(x_angle + 90));
}

void CameraControl::strafeRight()
{
        prevz = z;
        prevx = x;
        // Angles - 90 degrees for an angle that is perpendicular to x_angle
        z = z + moveSpeed * cos(toRadian(x_angle - 90));
        x = x - moveSpeed * sin(toRadian(x_angle - 90));
}

void CameraControl::moveForward(int mod)
{
```

```
        prevz = z;
        prevx = x;
        z = z + moveSpeed * mod * cos(toRadian(x_angle));
        x = x - moveSpeed * mod * sin(toRadian(x_angle));
}

void CameraControl::moveBackward(int mod)
{
        prevz = z;
        prevx = x;
        z = z - moveSpeed * mod * cos(toRadian(x_angle));
        x = x + moveSpeed * mod * sin(toRadian(x_angle));
}

void CameraControl::moveUp()
{
        y -= moveSpeed;
}

void CameraControl::moveDown()
{
        y += moveSpeed;
}

void CameraControl::invertCam()
{
        z_angle += 180;
}

void CameraControl::resetCam()
{
        x = 0.0;
        y = 0.0;
        z = -1.0;
        x_angle = 0.0;
        y_angle = 0.0;
        z_angle = 0.0;

}

void CameraControl::increaseSpeed()
{
        moveSpeed *= 2;
}

void CameraControl::decreaseSpeed()
{
        moveSpeed /= 2;
}

void CameraControl::Display()
{
        // To stop eternal movement
        glLoadIdentity();

        // Rotate along proper axes
```

```
        glRotatef(y_angle, 1, 0, 0);
        glRotatef(x_angle, 0, 1, 0);
        glRotatef(z_angle, 0, 0, 1);

        // Translate along the Plane
        glTranslatef(x, y, z);
}
```

6.1.4 CollisionEngine.h

```
/*************************************************************\
 * CollisionEngine.h                                         *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file creates the decleration of the CollisionEngine  *
 * class, which uses sweet sweet math to determine how the   *
 * player interacts with his environment                     *
\*************************************************************/

#ifndef COLLISION_ENGINE_H
#define COLLISION_ENGINE_H

class CollisionEngine
{
private:
        // Determines if wall/door collision occured
        bool collideWalls();
        // Determines if other collision occured
        bool collideObjects();
        // Determines if an object can be interacted with
        void checkInteract();
public:
        // Master function that calls others
        bool collide();

};

#endif
```

6.1.5 CollisionEngine.cpp

```
/*************************************************************\
 * CollisionEngine.h                                         *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the definition of the CollisionEngine  *
 * class. For more information, see SaveManager.h            *
\*************************************************************/

#include "CollisionEngine.h"

// For the Cam
#include "Globals.h"
```

```
// absolute value
#include <cmath>

// System Log
#include "Logger.h"

using namespace std;

const double PLAYER_RADIUS = 0.5;
const double INTERACT_RADIUS = 1; // Object interactivity radius
const double COLLIDE_RADIUS = 0.5;

void CollisionEngine::checkInteract()
{
        activeSwitch = NULL;
        activeTerminal = NULL;
        // Auto don't work in these parts
        for (unsigned int i = 0; i < switches.size(); i++)
        {
                double distance = pow((switches[i].getX() + Cam.x), 2) + pow((switches[i].getY() + Cam.y), 2) + pow((switches[i].getZ() + Cam.z), 2);
                distance = sqrt(distance);
                double radii = (PLAYER_RADIUS + INTERACT_RADIUS);

                if (distance < radii && switches[i].checkIfOn())
                {
                        interactivity = true;
                        activeSwitch = &switches[i];
                        return;
                }
        }

        for (unsigned int i = 0; i < terminals.size(); i++)
        {
                double distance = pow((terminals[i].getX() + Cam.x), 2) + pow((terminals[i].getY() + Cam.y), 2) + pow((terminals[i].getZ() + Cam.z), 2);
                distance = sqrt(distance);
                double radii = (PLAYER_RADIUS + INTERACT_RADIUS);

                if (distance < radii && terminals[i].checkIfOn())
                {
                        interactivity = true;
                        activeTerminal = &terminals[i];
                        return;
                }
        }

        interactivity = false;
}

bool CollisionEngine::collideObjects()
{
        for (unsigned int i = 0; i < terminals.size(); i++)
        {
```

```
                double distance = pow((terminals[i].getX() + Cam.x), 2) + pow((terminals[i].getY() + Cam.y), 2) + pow((terminals[i].getZ() + Cam.z), 2);
                distance = sqrt(distance);
                double radii = (PLAYER_RADIUS + COLLIDE_RADIUS);

                if (distance < radii && terminals[i].checkIfOn())
                {
                        return true;
                }
        }

        return false;
}

bool CollisionEngine::collideWalls()
{
        // Gotta check doors first
        // And if you hit an open door
        // You just ignore collision
        // Because otherwise you can't fit
        for (auto i : doors)
        {
                double distance = fabs(Cam.x * i.a + Cam.y * i.b + Cam.z * i.c + i.d); // Distance from door

                if ((distance / i.getNorm() < PLAYER_RADIUS) && i.isInBounds())
                {
                        if (i.isOpen) return false;
                        else return true;
                }
        }

        for (auto i : walls)
        {
                double distance = fabs(Cam.x * i.a + Cam.y * i.b + Cam.z * i.c + i.d); // Distance from wall

                if ((distance / i.getNorm() < PLAYER_RADIUS) && i.isInBounds()) return true;
        }

        return false;
}

bool CollisionEngine::collide()
{
        if (!collision)
        {
                return false;
        }

        checkInteract();
        return (collideWalls() || collideObjects());
}
```

6.1.6 Console.h

```
/*********************************************************\
 * Connsole.h                                            *
 * This file was created by Jeremy Greenburg             *
 * As part of The God Core game for the University of    *
 * Tennessee at Martin's University Scholars Organization *
 *                                                       *
 * This file contains the declaration of the Console Class, *
 * As well as the Trip struct for holding three integers *
 * The Developer Console takes input from the user and   *
 * Activates various effects based upon what the user has *
 * Typed in.                                             *
\*********************************************************/

#ifndef CONSOLE_H
#define CONSOLE_H

// To act as a circular buffer for console history
#include <deque>
// Stores actual console input
#include <vector>
// std::string
#include <string>
// For processing text
#include "TextEngine.h"

// Windows API
#include <shlobj.h>


// To make rgb calues easier to store
#include "Triple.h"

class Console
{
private:
        /***** Variables for the console itself ****/

        // Triples for good color, bad color, and nuetral colors
        Triple VALID_COLOR, INVALID_COLOR, NEUTRAL_COLOR;
        // What the console "says" (aka what appears on screen)
        std::deque<std::string> console_log;
        // The colors of said strings
        std::deque<Triple> console_color;
        // Contains the actual player input
        std::vector<std::string> console_input;
        // The current (finished) input being processed
        std::string currentInput;
        // The current (unfinished) input being type
        std::string currentText;
        // Console History
        TextEngine log;

        // Path to core.sav
        char CHAR_PATH[MAX_PATH];
```

```
        std::string SAVE_PATH;

        // Is the console active or not
        bool isActive;

        // The bottom of the console
        const int SCREENBOTTOM = 500;

        // Prints the current input and console_history
        void printInput();
        // Processes completed input
        void processInput();

        // Command functions

        // Toggles collision on and off
        void toggleCollision();

        // Toggles godMode on and off
        void toggleGod();

        // Decrpyts the entry in core.sav
        void decrpytSave();

        // Shutdowns program
        void halt();

        // Clears the console log
        void clear();

        // Writes input to core.sav
        void writeToSave(std::string input);

        // Reads a bit from the file
        void readFromFile(std::string input);

        // Changes the currently played track
        void playSong(std::string input);

public:
        // Initializes VALID_COLOR, INVALID_COLOR, NEUTRAL_COLOR, and SAVE_PATH
        Console();
        // Manages console functions if input has been provided
        void activate(std::string input, std::string text);
        // Manages console function if input is still being provided
        void activate(std::string text);
        // Returns the console_input[count]
        std::string getHist(int count);
        // Returns console_input.size()
        int getHistNum();

};

#endif
```

6.1.7 Console.cpp

```
/*************************************************************\
 * Console.cpp                                               *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the definition of the Console class    *
 * For more information, see Console.cpp                     *
\*************************************************************/

// File I/O
#include <fstream>

// Class declaration
#include "Console.h"

// For saving and loading
#include "SaveManager.h"

// System log
#include "Logger.h"

// Contains global environment variables
#include "Globals.h"

// Return codes
#include "Return.h"

using namespace std;

Console::Console()
{
        // Green!
        VALID_COLOR = makeTrip(0, 1, 0);
        // Red!
        INVALID_COLOR = makeTrip(1, 0, 0);
        // Gray!
        NEUTRAL_COLOR = makeTrip(1, 1, 1);

        // Get path to documents
        HRESULT ret = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL,
            SHGFP_TYPE_CURRENT, CHAR_PATH);
        // Assign to SAVE_PATH
        SAVE_PATH = CHAR_PATH;
        // Concatenate save file
        SAVE_PATH += "\\The God Core\\core.sav";
}

void Console::activate(string input, string text)
{
        currentInput = input;
        // This should be empty. But just incase.
        currentText = text;

        processInput();
        printInput();
```

```
}

void Console::activate(string text)
{
        currentText = text;

        printInput();
}

void Console::printInput()
{
        deque<string>::iterator it = console_log.begin();
        deque<Triple>::iterator jt = console_color.begin();
        // Iterates through the console's current log and prints it to the screen
        for (it; it != console_log.end(); it++, jt++)
        {
                //                                        Index of it
                log.printString(0, 10 + 20 * (it - console_log.begin()),
                        jt->a, jt->b, jt->c, *it);
        }

        // Prints whatever the user is typing
        log.printString(0, SCREENBOTTOM / 2.4, 1, 1, 1, currentText);
}

void Console::processInput()
{
        // TODO: Break this behemoth up into little, managable functions

        if (currentInput == "TogClip")
                toggleCollision();

        else if (currentInput == "TogGod")
                toggleGod();

        else if (currentInput.substr(0, 5) == "Save ")
                writeToSave(currentInput.substr(5)); // Save everything after "
                    Save "

        else if (currentInput == "Decrypt")
                decrpytSave();

        else if (currentInput.substr(0, 5) == "Read ")
                readFromFile(currentInput.substr(5)); // Read everything after "
                    Read "

        else if (currentInput == "Halt")
                halt();

        else if (currentInput == "Clear")
                clear();

        else if (currentInput.substr(0, 5) == "Play ")
                playSong(currentInput.substr(5)); // Process everything after "
                    Play "

```

```
        else if (currentInput == "Goto Main")
        {
                isInMain = true;
                isInConsole = false;
                HUD.toggleConsole();
        }

        // Invalid command
        else
        {
                console_log.push_back("ERROR: Do not recognize \"" + currentInput + '\"');
                console_color.push_back(INVALID_COLOR);
        }

        // Clears the top of the console if too much history is added
        if (console_log.size() > 9)
        {
                console_log.pop_front();
                console_color.pop_front();
        }

        // Store the current input
        console_input.push_back(currentInput);
}

void Console::writeToSave(string input)
{
        // Writes whatever is in input to the save file.
        // Probably not going to be good for loading purposes

        SaveManager Jesus;

        Jesus.saveLevel();

        console_log.push_back("Saved: " + input);
        console_color.push_back(VALID_COLOR);
}

void Console::readFromFile(string input)
{
        // Syntax = Read core.sav
        if (input == "core.sav")
        {
                ifstream infile(SAVE_PATH);

                string text;

                // For now, core.sav only has one line. Hopefully I'll update this when I change that
                infile >> text;

                console_log.push_back(text);
                console_color.push_back(VALID_COLOR);
        }

```

```
        // Syntax = Read TAG FILE
        else
        {
                // There should be a space seperating the file and the tag. We find that space
                size_t pos = input.find(' ');

                // If there ain't no space
                if (pos == string::npos)
                {
                        console_log.push_back("ERROR: No tag detected");
                        console_color.push_back(INVALID_COLOR);
                }

                // Hooray! There's a space
                else
                {
                        string tag = input.substr(0, pos);
                        string file = input.substr(pos + 1); // +1 to avoid the space

                        const char* TEXT_PATH = "Resources\\Text\\";
                        string fullPath = TEXT_PATH + file;

                        // Simply to test for the file's existence
                        ifstream infile(fullPath);

                        string text;
                        getline(infile, text);

                        // If there ain't no file
                        if (!infile)
                        {
                                console_log.push_back("ERROR: File \"" + file + "\" not found");
                                console_color.push_back(INVALID_COLOR);
                        }

                        // Hooray! There's a file
                        else
                        {
                                console_log.push_back("Reading \"" + file + "\" with tag \"" + tag + '\"');
                                console_color.push_back(VALID_COLOR);

                                vector<string> readText = log.getText(file, tag);

                                vector<string>::iterator it;

                                for (it = readText.begin(); it != readText.end(); it++)
                                {
                                        // Push everything we found into the log
                                        console_log.push_back(*it);
                                        console_color.push_back(NEUTRAL_COLOR);

```

```
                                        // So we don't grow too much, keep bounds checking
                                        if (console_log.size() > 9)
                                        {
                                                console_log.pop_front();
                                                console_color.pop_front();
                                        }
                                }
                        }

                        infile.close();
                }
        }
}

void Console::toggleCollision()
{
        console_log.push_back("Noclip toggled.");
        console_color.push_back(VALID_COLOR);

        collision = !collision;
}

void Console::toggleGod()
{
        console_log.push_back("God Mode toggled.");
        console_color.push_back(VALID_COLOR);

        godMode = !godMode;
}

void Console::decrpytSave()
{
        SaveManager Jesus;

        console_log.push_back(Jesus.readSave());
        console_color.push_back(VALID_COLOR);
}

void Console::halt()
{
        Logger log;
        log.logLine("Exiting via console");
        exit(EXIT_OK);
}

void Console::clear()
{
        console_log.clear();
        console_color.clear();
        console_input.clear();
}

void Console::playSong(string input)
{
        int sNum = getSongNum(input);
```

```

        if (sNum == -1) // Invalid input
        {
                console_log.push_back("ERROR: " + input + " not a valid song file
                    ");
                console_color.push_back(INVALID_COLOR);
        }

        else // Valid input
        {
                songNum = sNum;
                changeSong = true;
                string song = getSongName(sNum);
                console_log.push_back("Now playing " + song);
                console_color.push_back(VALID_COLOR);
        }
}

string Console::getHist(int count)
{
        int size = console_input.size();
        if (console_input.empty())
        {
                return "";
        }

        // If, somehow, a fool manages to get a variable that is out of bounds

        else if (count >= size)
        {
                return console_input.back();
        }

        else if (count < 0)
        {
                return console_input.front();
        }

        else
        {
                return console_input[size - count - 1];
        }
}

int Console::getHistNum()
{
        return console_input.size();
}
```

6.1.8 Cylinder.h

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

```
 * This file contains the declaration of the Cylinder Class, *
 * Which contains the functionality to load and display a    *
 * Cylindrical object in game                                *
\*********************************************************/

#ifndef CYLINDER_H
#define CYLINDER_H

#include <cstdlib>

#include <GL\glut.h>

class Cylinder
{
private:
        // A few variables to control the shape of the cylinder
        double baseRadius, topRadius, height;
        int stacks, slices;

        // Arrays for the location, orientation, and color of the cylinder
        double translate[3], rotate[3], color[4];
        // A thingamajig for glut
        GLUquadric *quad;
public:
        Cylinder(double _baseRadius, double _topRadius, double _height, int _stacks, int _slices,
                const double(&_translate)[3], const double(&_rotate)[3], const double (&_color)[4]);

        void Display();
        ~Cylinder();
};

#endif
```

6.1.9 Cylinder.cpp

```
/*********************************************************\
 * Cylinder.cpp                                           *
 * This file was created by Jeremy Greenburg              *
 * As part of The God Core game for the University of     *
 * Tennessee at Martin's University Scholars Organization *
 *                                                        *
 * This file contains the defintion of the Cylinder class. *
 * for more information, see Cylinder.h                   *
\*********************************************************/

#include "Cylinder.h"

// For copying
#include <iterator>
#include <utility>

using namespace std;

Cylinder::Cylinder(double _baseRadius, double _topRadius, double _height, int _stacks, int _slices,
```

```
        const double(&_translate)[3], const double(&_rotate)[3], const double(&
            _color)[4])
{
        baseRadius = _baseRadius;
        topRadius = _topRadius;
        height = _height;
        stacks = _stacks;
        slices = _slices;

        copy(begin(_color), end(_color), color);
        copy(begin(_translate), end(_translate), translate);
        copy(begin(_rotate), end(_rotate), rotate);

        quad = gluNewQuadric();
}

Cylinder::~Cylinder()
{
        //gluDeleteQuadric(quad);
}

void Cylinder::Display()
{
        glColor4d(color[0], color[1], color[2], color[3]);

        glPushMatrix();

        glTranslated(translate[0], translate[1], translate[2]);
        glRotated(rotate[0], 1, 0, 0);
        glRotated(rotate[1], 0, 1, 0);
        glRotated(rotate[2], 0, 0, 1);

        gluCylinder(quad, baseRadius, topRadius, height, slices, stacks);

        glPopMatrix();
}
```

6.1.10 Door.h

```
/*************************************************************\
 * Door.h                                                    *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the Door class      *
 * It's mostly a fancy wrapper for a Plane with a bit        *
 * Of added functionality                                    *
\*************************************************************/

#ifndef DOOR_H
#define DOOR_H

// Class decleration
#include "Plane.h"
// std::string
#include <string>
```

```

// Figure out a way to bind a controller to the door to activate it.
class Door
{
private:
        // Name, so a switch can find it
        std::string id;
        // The physical door
        Plane rect;
public:
        // Is the door open?
        bool isOpen;
        // Plane's a, b, c, and d.
        // For easier access
        double a, b, c, d;

        // Takes in the initial Plane and name
        Door(Plane _rect, std::string _id);
        // Calls rect.Display()
        void Display();
        // Returns rect.getNorm()
        double getNorm();
        // Returns id
        std::string getID();
        // Returns rect.isInBounds()
        bool isInBounds();
};

#endif
```

6.1.11 Door.cpp

```
/*************************************************************\
 * Door.cpp                                                  *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the defintion of the Door class.       *
 * for more information, see Door.h                          *
\*************************************************************/

// Class declaration
#include "Door.h"

using namespace std;

Door::Door(Plane _rect, std::string _id) : rect(_rect), id(_id)
{
        isOpen = false;
        a = rect.a;
        b = rect.b;
        c = rect.c;
        d = rect.d;
};

void Door::Display()
```

```
{
        if (!isOpen) rect.Display();
}

double Door::getNorm()
{
        return rect.getNorm();
}

string Door::getID()
{
        return id;
}

bool Door::isInBounds()
{
        return rect.isInBounds();
}
```

6.1.12 GameManager.h

```
/*************************************************************\
 * GameManager.h                                             *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the GameManger class*
 * Which oversees and manages the flow of the game           *
\*************************************************************/

#ifndef GAMEMANAGER_H
#define GAMEMANAGER_H

//****** LIBRARIES AND CLASSES ******\\

// For the keyboard functionality
#include "Keyboard.h"

// glut really wants cstdlib here
#include <cstdlib>

// For arrays of strings
#include <string>
#include <vector>

// OpenGL API
#include <GL\glut.h>

// Standard I/O for debugging
#include <iostream>

// To manage background music
#include "MusicManager.h"

// To manage saving and loading
#include "SaveManager.h"
```

```

class GameManager
{
private:
        // Variables

        // Objects
        MusicManager SoundSystem;
        Keyboard board;

        // Because the main menu is dumb, we have to know when to get a click
        bool processClick = false;

        // When in the main menu, mouse coords of a click
        int mouse_x, mouse_y;

        // Functions

public:

        // Captures mouse clicks
        void mouse(int button, int state, int x, int y);
        // Captures mouse motion
        void motionPassive(int x, int y);
        // CHanges window size
        void changeSize(int w, int h);
        // Manages scene display
        void manageScenes();
        // Displaying function
        void draw();
        // Function to bring about game end on Level 4
        void endGame();
        // Normal key presses
        void normal(unsigned char key, int x, int y);
        // Key releases
        void key_up(unsigned char key, int x, int y);
        // Special keys
        void special(int key, int x, int y);
        // To manage playing and releasing music
        void manageMusic();

        // Wether or not core.sav exists
        bool canContinue;

};

#endif
```

6.1.13 GameManager.cpp

```
/*********************************************************\
 * GameManager.cpp                                        *
 * This file was created by Jeremy Greenburg              *
 * As part of The God Core game for the University of     *
 * Tennessee at Martin's University Scholars Organization *
 *                                                        *
 * This file contains the defintion of the GameManager class.*
```

```
 * for more information, see GameManager.h                    *
\*************************************************************/

// Class declaration
#include "GameManager.h"
// Globals
#include "Globals.h"
// Level
#include "Level.h"
// Main Menu
#include "MainMenu.h"

#include "Logger.h"

#include "Return.h"

using namespace std;

void GameManager::mouse(int button, int state, int x, int y)
{
        if (button == GLUT_RIGHT_BUTTON)
        {
                // Jokes on my I never ended up using the right button
                if (state == GLUT_DOWN)
                {

                }

                else
                {

                }
        }

        else if (button == GLUT_LEFT_BUTTON)
        {
                if (state == GLUT_DOWN)
                {
                        if (isInMain)
                        {
                                mouse_x = x;
                                mouse_y = y;
                                processClick = true;
                        }

                        Logger log;
                        vector<string> output = { "X: ", to_string(x), " ", "Y:",
                            to_string(y) };
                        log.logLine(output);
                }

                else
                {

                }
        }
```

```
}

void GameManager::motionPassive(int x, int y)
{
        static int _x = 0, _y = 0;

        // If nothing else is happening basically
        if (!isInConsole && !isInTerminal && !isInMain)
        {
                if (x > _x)
                {
                        Cam.lookRight();
                        _x = x;
                }

                else if (x < _x)
                {
                        Cam.lookLeft();
                        _x = x;
                }

                if (y < _y)
                {
                        Cam.lookUp();
                        _y = y;
                }

                else if (y > _y)
                {
                        Cam.lookDown();
                        _y = y;
                }

                // Loop around to the other side of the screen

                bool updateMouse = false;
                int newY = y, newX = x;
                if (y == 0 || y > 700)
                {
                        updateMouse = true;
                        newY = 300;
                        _y = 300;
                }

                if (x == 0 || x > 700)
                {
                        updateMouse = true;
                        newX = 300;
                        _x = 300;
                }

                if (updateMouse)
                {
                        glutWarpPointer(newX, newY);
                }
        }
```

```
}

void GameManager::changeSize(int w, int h)
{
        // Don't want to divide by zero
        if (h == 0)
                h = 1;

        double ratio = w * 1.0 / h;

        // Use the Projection Matrix
        glMatrixMode(GL_PROJECTION);

        // Reset Matrix
        glLoadIdentity();

        // Set the viewport to be the entire window
        glViewport(0, 0, w, h);

        // Set the correct perspective.
        gluPerspective(45, ratio, 1, 100);

        // Get Back to the Modelview
        glMatrixMode(GL_MODELVIEW);
}

void GameManager::draw()
{
        if (loading)
        {
                lvl.loadLevel(curr_level);

                loading = false;

                // Save current progress after loading level
                SaveManager Jesus; // saves
                Jesus.saveLevel();
        }

        else
        {
                lvl.displayLevel();
        }
}

void GameManager::endGame()
{
        if (loading)
        {
                lvl.loadLevel(curr_level);

                loading = false;

                // Save current progress after loading level
                SaveManager Jesus; // saves
                Jesus.saveLevel();
```

```
        }

        else
        {
                // The time left for each segment
                static int timeLeft = 1000;
                // The last level is divided into 3 segments
                static int segment = 1;
                // Wether the current segment has been initialized yet
                static bool initSegment = true;

                // The last portion of the game is divided into 3 segments
                if (segment == 1)
                {
                        HUD.displayWarning("");
                        glClearColor(1, 1, 1, 1);
                }

                else if (segment == 2)
                {
                        if (initSegment)
                        {
                                HUD.displayWarning("QUANT");
                                initSegment = false;
                        }

                        for (unsigned i = 0; i < walls.size(); i++)
                        {
                                walls[i].mutate();
                        }
                }

                else if (segment == 3)
                {
                        if (initSegment)
                        {
                                HUD.goFade(15);
                                HUD.setStatus("INFO-UN");
                                initSegment = false;
                        }

                        for (unsigned i = 0; i < walls.size(); i++)
                        {
                                walls[i].mutate();
                        }
                }

                else
                {
                        // Return to main menu at game end
                        isInMain = true;

                        // Return everything to as it was before level 4
                        HUD.setStatus("INFO-WELL");
                        HUD.displayWarning("");
                        segment = 1;
```

```
                        HUD.goDark(0);
                }

                // Switch segments
                if (timeLeft == 0)
                {
                        timeLeft = 1000;
                        segment++;
                        initSegment = true;
                }
                timeLeft--;


                // Draw the titular object
                glPushMatrix();
                glTranslated(0, 0, -7);
                glColor4d(.9, .9, .9, 1);
                glutSolidSphere(3, 50, 50);
                glPopMatrix();

                lvl.displayLevel();
        }
}

void GameManager::manageScenes()
{
        // If we need to change the song, we can do it here
        if (changeSong)
        {
                manageMusic();
        }

        // Clears the previous drawing
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        if (isInTerminal)
        {
                activeTerminal->DisplayScreen();
        }

        else if (isInMain)
        {
                // Enable using textures (pictures)
                glutSetCursor(GLUT_CURSOR_LEFT_ARROW);
                static MainMenu MM;

                // For some reason, MM breaks horribly when it's a global or class
                    member
                // So we'll just handle mouse clicks in the display function
                // Rather than the mouse click function
                // Because I'm a competent programmer
                if (processClick)
                {
                        MM.getClick(mouse_x, mouse_y);
                        processClick = false;
                }
```

```
286
287                 MM.display();
288         }
289
290         // glutSetCursor(GLUT_CURSOR_LEFT_ARROW); Keypads maybe?
291
292         else
293         {
294                 // Enable using textures (pictures)
295                 glutSetCursor(GLUT_CURSOR_NONE);
296
297                 if (curr_level != "LEVELFOUR") draw();
298
299                 else endGame();
300
301                 // Moves the camera to the correct position
302                 Cam.Display();
303
304                 // Prompt the user to interact if we should
305                 if (interactivity) HUD.displayWarning("INTERACT");
306                 else if (curr_level != "LEVELFOUR") HUD.displayWarning("");
307
308                 // Prints the HUD
309                 HUD.DisplayHUD();
310         }
311
312         // Displays the current drawing
313         glutSwapBuffers();
314 }
315
316 void GameManager::manageMusic()
317 {
318         // All variables need to persist between frames
319         static SoundClass background;
320
321         SoundSystem.releaseSound(background);
322         changeSong = false;
323
324         // Because you can never have too much bounds checking
325         if (songNum >= 0 && songNum <= 9)
326         {
327                 std::string song = getSongName(songNum);
328                 SoundSystem.makeSound(&background, song.c_str());
329                 SoundSystem.playSound(background);
330         }
331 }
332
333 // Normal key presses
334 void GameManager::normal(unsigned char key, int x, int y)
335 {
336         board.normal(key, x, y);
337 }
338
339 // Key releases
340 void GameManager::key_up(unsigned char key, int x, int y)
341 {
```

```
        board.key_up(key, x, y);
}

// Special keys
void GameManager::special(int key, int x, int y)
{
        board.special(key, x, y);
}
```

6.1.14 GCTypes.h

```
/*************************************************************\
 * GCTypes.h                                                 *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains integer types corresponding to various *
 * In game object types                                      *
\*************************************************************/

#ifndef GC_TYPES_H
#define GC_TYPES_H

// Object Types

#define T_NULL 0                // Nothing
#define T_DOOR 1                // Door
#define T_TERMINAL 2    // Terminal
#define T_SWITCH 3              // Switch
#define T_LEVEL_END 4   // Switch that ends level

typedef int GCtype;

#endif
```

6.1.15 Globals.h

```
/*************************************************************\
 * Globals.h                                                 *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the Globals         *
 * All of them.                                              *
 * Thers a lot of them                                       *
\*************************************************************/

#ifndef GLOBALS_H
#define GLOBALS_H

// ALLLLLLL the classes
#include "HeadsUpDisplay.h"
#include "CameraControl.h"
#include "Level.h"
#include "Terminal.h"
```

```
#include "Door.h"
#include "Switch.h"
#include "Plane.h"
#include "Trigger.h"
#include "Cylinder.h"

// Remember that if you're doing anything else, globals are bad.
// But we're in the hellscape that is graphics
// There are no rules here
// Only madness dwells here

// Typedefs make life easy
typedef std::vector<Plane> vr;
typedef std::vector<Door> vd;
typedef std::vector<Switch> vs;
typedef std::vector<Terminal> vt;
typedef std::vector<Trigger> vtr;
typedef std::vector<Cylinder> vc;

// Pointers to various interactive objects
extern Switch *activeSwitch;
extern Terminal *activeTerminal;

// Vectors containing all of the level's assets
extern vr walls;
extern vd doors;
extern vs switches;
extern vt terminals;
extern vtr triggers;
extern vc cylinders;

extern bool
        // Are we colliding / Can we die?
        collision, godMode,
        // Go dim or go dark?
        goDim, goDark,
        // Dunno if I actually need this one
        loading,
        // Is in varius different stages of non-normal play
        isInConsole, isInTerminal, isInMain,
        // Should we change the song?
        changeSong,
        // Is something in interaction range?
        interactivity;

// Number of song to change to
extern int songNum;

// Current level (int and string)
extern int levelNum;
extern std::string curr_level;

// Constant strings of the song names
extern const char *SONG0, *SONG1, *SONG2, *SONG3, *SONG4, *SONG5,
                                        *SONG6, *SONG7, *SONG8, *SONG9;

```

```
// A few global objects
extern HeadsUpDisplay HUD;
extern CameraControl Cam;
extern Level lvl;

// Converts a songname to an integer
int getSongNum(std::string input);
// Converts an integer to a songname
std::string getSongName(int input);
// Converts a level name to an integer
int getLevelNum(std::string input);
// Converts level_num to a string in curr_level
std::string getLevelString(int input);
// Safely advance the song
void advanceMusic();

#endif
```

6.1.16 Globals.cpp

```
/*************************************************************\
 * Globals.cpp                                               *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file instantiates the global variables               *
\*************************************************************/

#include "Globals.h"

vr walls;
vd doors;
vs switches;
vt terminals;
vtr triggers;
vc cylinders;

Switch *activeSwitch = NULL;
Terminal *activeTerminal = NULL;

bool collision = true;
bool godMode = false;
bool goDim = false;
bool goDark = false;
bool loading = true;
bool isInConsole = false;
bool isInTerminal = false;
bool isInMain = true;
bool changeSong = true;
bool interactivity = false;

int songNum = 0;

int levelNum = 0;
std::string curr_level = "LEVELZERO";

```

```
const char* SONG0 = "Dark Fog.mp3";
const char* SONG1 = "Mismer.mp3";
const char* SONG2 = "One Sly Move.mp3";
const char* SONG3 = "Hypnothis.mp3";
const char* SONG4 = "Cold Hope.mp3";
const char* SONG5 = "Spacial Harvest.mp3";

HeadsUpDisplay HUD;
CameraControl Cam;
Level lvl;

int getSongNum(std::string input)
{
        if (input == SONG0 || input == "0")
                return 0;
        if (input == SONG1 || input == "1")
                return 1;
        if (input == SONG2 || input == "2")
                return 2;
        if (input == SONG3 || input == "3")
                return 3;
        if (input == SONG4 || input == "4")
                return 4;
        if (input == SONG5 || input == "5")
                return 5;
        return -1; // Invalid song
}

std::string getSongName(int input)
{
        std::string ret;
        switch (input)
        {
        case 0: ret = SONG0;
                break;
        case 1: ret = SONG1;
                break;
        case 2: ret = SONG2;
                break;
        case 3: ret = SONG3;
                break;
        case 4: ret = SONG4;
                break;
        case 5: ret = SONG5;
                break;
        default: ret = "\0";;
                break;
        }

        return ret;
}

int getLevelNum(std::string input)
{
        if (input == "LEVELZERO" || input == "LEVELZERO\n")
                return 0;
```

```
        if (input == "LEVELONE" || input == "LEVELONE\n")
                return 1;
        if (input == "LEVELTWO")
                return 2;
        if (input == "LEVELTHREE")
                return 3;
        if (input == "LEVELFOUR")
                return 4;
        return -1; // Invalid song
}

std::string getLevelString(int input)
{
        std::string ret;
        switch (input)
        {
        case 0: ret = "LEVELZERO";
                break;
        case 1: ret = "LEVELONE";
                break;
        case 2: ret = "LEVELTWO";
                break;
        case 3: ret = "LEVELTHREE";
                break;
        case 4: ret = "LEVELFOUR";
                break;
        default: ret = "ERROR";;
                break;
        }

        return ret;
}

void advanceMusic()
{
        songNum++;
        if (songNum > 5) songNum = 0;
}
```

6.1.17 HeadsUpDisplay.h

```
/*************************************************************\
 * HeadsUpDisplay.h                                          *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the HeadsUpDisplay  *
 * Class, which created an Orthoganl Matrix infront of the   *
 * Screen which allows for a 2D Heads Up Display to be       *
 * Printed before the user at any time                       *
 * It also passes input to the developer console             *
\*************************************************************/

#ifndef HEADSUPDISPLAY
#define HEADSUPDISPLAY

```

```
// Base class for 2D operations
#include "TwoD.h"

// For displaying text in the HUD
#include "TextEngine.h"
// The Developer Console
#include "Console.h"

class HeadsUpDisplay : public TwoD
{
private:
        // Duration of time to dim screen (Goes from black to clear as time progresses)
        int dimTime = 0;
        // Duration of time to go dark (completely black)
        int darkTime = 0;
        // Duration of the time to fade the screen (goes from clear to black as time progresses)
        int fadeTime = 0;
        // Wether or not to dim
        bool dimNow = false;
        // Wether or not to darken
        bool darkNow = false;
        // Wether or not to fade
        bool fadeNow = false;
        // Wether or not we are in developer console
        bool devConsole = false;

        // Tag to current alert
        std::string currentAlert;
        // Tag to current status
        std::string currentStatus;
        // Text to print to the screen
        std::string currentText;
        // What the user is typing
        std::string currentInput;

        // To Display text
        TextEngine helmet;
        // Dev Console
        Console dev;

        // Draws an info bar at the top of the screen
        void drawHelmetBounds();
        // Displays suit alerts
        void DisplayAlerts();
        // Draws the Heads Up Display
        void drawHUD();
        // Manages the dimming of the screen
        void dim();
        // Manages the darkening of the screen
        void dark();
        // Manages the fading of the screen
        void fade();
        // Draws the box which stores the info text
        void drawInfoBox();
```

```
        // Draws the developer console window
        void drawConsole();
        // Displays standard info in the top left corner
        void displayInfo(std::string tag);


public:
        // Manages the HUD
        void DisplayHUD();

        /****           ALTERATION FUNCTIONS            *****\
        \**** Should always be called before DisplayHud *****/

        // Tells the HUD how long to dim
        void goDim(int time);

        //Tells the HUD how long to go dark
        void goDark(int time);
        // Tels the HUD how long to fade for
        void goFade(int time);
        // Flips dev_console
        void toggleConsole();

        // Takes in a tag to print to screen
        void displayWarning(std::string warning);

        // Takes in a string to display in the status box
        void setStatus(std::string status);

        // Takes in a string to print to screen
        void printToConsole(std::string text);

        // Signifies a completed input to the console
        void inputString(std::string text);

        // Returns an item of the console's log
        std::string getHist(int count);

        // Returns the number of items in the console's log
        int getHistNum();
};

#endif
```

6.1.18 HeadsUpDiplay.cpp

```
/*************************************************************\
 * HeadsUpDisplay.cpp                                        *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the definition of the HeadsUpDisplay   *
 * Class. For more information, see HeadsUpDisplay.h         *
\*************************************************************/

// Class Declaration
```

```
#include "HeadsUpDisplay.h"

// OpenGL API
#include <gl\glut.h>

// For counting seconds
#include <ctime>

// For displaying Planes
#include "Plane.h"

// For displaying triangles
#include "Triangle.h"

using namespace std;

void HeadsUpDisplay::drawHelmetBounds()
{
        // Helmet bounds are black
        double colors[4] = { 0, 0, 0, 1 };

        // The top of the helmet
        double top_vertices[9] =
        {
                SCREENRIGHT, SCREENTOP, -1,
                SCREENLEFT, SCREENTOP, -1,
                SCREENRIGHT / 2.0, SCREENBOTTOM / 20.0, -1
        };

        // The left of the hemlet
        double left_vertices[9] =
        {
                SCREENLEFT, SCREENBOTTOM, -1,
                SCREENLEFT, SCREENTOP, -1,
                SCREENRIGHT / 20.0, 3 * SCREENBOTTOM / 5.0, -1
        };

        // The back of the helmet
        double right_vertices[9] =
        {
                SCREENRIGHT, SCREENBOTTOM, -1,
                SCREENRIGHT, SCREENTOP, -1,
                19 * SCREENRIGHT / 20.0, 3 * SCREENBOTTOM / 5.0, -1
        };

        Triangle top_helm{ top_vertices, colors };
        Triangle left_helm{ left_vertices, colors };
        Triangle right_helm{ right_vertices, colors };

        top_helm.Display2D();
        left_helm.Display2D();
        right_helm.Display2D();
}

void HeadsUpDisplay::DisplayAlerts()
{
```

```
        helmet.openFile(.5 * SCREENRIGHT, .5 * SCREENBOTTOM,
                1, 1, 1,
                "suitAlerts.log", currentAlert);
}

void HeadsUpDisplay::dim()
{
        static int startTime;
        static bool timeSet = false;
        if (dimNow)
        {
                if (!timeSet)
                {
                        startTime = time(NULL);
                        timeSet = true;
                }

                int currentTime = time(NULL);
                int timeElapsed = currentTime - startTime;
                if (timeElapsed < dimTime)
                {
                        // A black square that grows more transparent as time
                            passes
                        double colors[4] = { 0, 0, 0, (double)(dimTime -
                            timeElapsed) / dimTime };
                        double dimVert[12] =
                        {
                                SCREENLEFT, SCREENTOP, -1,
                                SCREENLEFT, SCREENBOTTOM, -1,
                                SCREENRIGHT, SCREENBOTTOM, -1,
                                SCREENRIGHT, SCREENTOP, -1
                        };

                        Plane black{ dimVert, colors };
                        black.Display2D();
                }

                else
                {
                        dimNow = false;
                        timeSet = false;
                }
        }
}

void HeadsUpDisplay::fade()
{
        static int startTime;
        static bool timeSet = false;
        if (fadeNow)
        {
                if (!timeSet)
                {
                        startTime = time(NULL);
                        timeSet = true;
                }
```

```

                int currentTime = time(NULL);
                int timeElapsed = currentTime - startTime;
                if (timeElapsed < fadeTime)
                {
                        // A black square that grows more transparent as time passes
                        double colors[4] = { 0, 0, 0, 1 - ((double)(fadeTime - timeElapsed) / fadeTime) };
                        double dimVert[12] =
                        {
                                SCREENLEFT, SCREENTOP, -1,
                                SCREENLEFT, SCREENBOTTOM, -1,
                                SCREENRIGHT, SCREENBOTTOM, -1,
                                SCREENRIGHT, SCREENTOP, -1
                        };

                        Plane black{ dimVert, colors };
                        black.Display2D();
                }

                else
                {
                        fadeNow = false;
                        timeSet = false;


                        // Go dark till game ends
                        darkNow = true;
                        darkTime = 1000;
                }
        }
}

void HeadsUpDisplay::dark()
{
        static int startTime;
        static bool timeSet = false;
        if (darkNow)
        {
                if (!timeSet)
                {
                        startTime = time(NULL);
                        timeSet = true;
                }

                int currentTime = time(NULL);
                int timeElapsed = currentTime - startTime;
                if (timeElapsed < darkTime)
                {
                        // A black square that obscures vision
                        double colors[4] = { 0, 0, 0, 1 };
                        double dimVert[12] =
                        {
                                SCREENLEFT, SCREENTOP, -1,
                                SCREENLEFT, SCREENBOTTOM, -1,
```

```
                                SCREENRIGHT, SCREENBOTTOM, -1,
                                SCREENRIGHT, SCREENTOP, -1
                        };

                        Plane black{ dimVert, colors };
                        black.Display2D();
                }

                else
                {
                        darkNow = false;
                        timeSet = false;
                }
        }
}

void HeadsUpDisplay::drawConsole()
{
        double colors[4] = { .1, .1, .1, .9 };
        double vertices[12] =
        {
                SCREENLEFT, SCREENTOP, -1,
                SCREENLEFT, SCREENBOTTOM / 5, -1,
                SCREENRIGHT, SCREENBOTTOM / 5, -1,
                SCREENRIGHT, SCREENTOP, -1
        };

        Plane console_tab{ vertices, colors };
        console_tab.Display2D();

        if (currentInput != "")
        {
                dev.activate(currentInput, currentText);
                currentInput.clear();
        }

        else
        {
                dev.activate(currentText);
        }
}

void HeadsUpDisplay::drawInfoBox()
{
        double colors[4] = { 0, 1, 1, .5 };
        double vertices[12] =
        {
                SCREENLEFT, SCREENTOP, -1,
                SCREENLEFT, SCREENBOTTOM / 10, -1,
                SCREENRIGHT / 10, SCREENBOTTOM / 10, -1,
                SCREENRIGHT / 10, SCREENTOP, -1
        };

        Plane info{ vertices, colors };
        info.Display2D();
}
```

```
void HeadsUpDisplay::displayInfo(string tag)
{
        helmet.openFile(SCREENLEFT, SCREENTOP +  20, 1, 1, 1,
                "suitAlerts.log", currentStatus);
}

void HeadsUpDisplay::goDim(int time)
{
        dimTime = time;
        dimNow = true;
}

void HeadsUpDisplay::goDark(int time)
{
        darkTime = time;
        darkNow = true;
}

void HeadsUpDisplay::goFade(int time)
{
        fadeTime = time;
        fadeNow = true;
}

void HeadsUpDisplay::displayWarning(std::string warning)
{
        currentAlert = warning;
}

void HeadsUpDisplay::setStatus(std::string status)
{
        currentStatus = status;
}

void HeadsUpDisplay::printToConsole(std::string text)
{
        currentText = text;
}

void HeadsUpDisplay::inputString(std::string text)
{
        currentInput = text;
}

void HeadsUpDisplay::toggleConsole()
{
        devConsole = !devConsole;
}

void HeadsUpDisplay::drawHUD()
{
        drawHelmetBounds();

        if (dimNow)
        {
```

```
                dim();
        }

        else if (fadeNow)
        {
                fade();
        }

        // Not else if due to fade -> dark transition
        if (darkNow)
        {
                dark();
        }

        drawInfoBox();
        displayInfo(currentStatus);

        if (devConsole)
        {
                drawConsole();
        }

        if (currentAlert != "")
        {
                DisplayAlerts();
        }
}

string HeadsUpDisplay::getHist(int count)
{
        return dev.getHist(count);
}

int HeadsUpDisplay::getHistNum()
{
        return dev.getHistNum();
}

void HeadsUpDisplay::DisplayHUD()
{
        prepare2D();

        drawHUD();

        prepare3D();
}
```

6.1.19 Keyboard.h

```
/*************************************************************\
 * Keyboard.h                                                *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the Keyboard class, *
 * which logs keypresses from the user and determines,       *
```

```
 * depending on the context, what action to take such.       *
\*************************************************************/

#ifndef KEYBOARD_H
#define KEYBOARD_H

// std::string
#include <string>

class Keyboard
{
private:
        // Signals to recieve a part of the console's history
        bool getPrev, getNext;

public:
        // Normal keys
        void normal(unsigned char key, int x, int y);
        // To read console input
        void inputConsole(unsigned char key, int x, int y);
        // To read terminal input
        void inputTerminal(unsigned char key, int x, int y);
        // To interact with the world
        void interact(unsigned char key, int x, int y);
        // If a key is released
        void key_up(unsigned char key, int x, int y);
        // Special keys (functions, arrows, ect.)
        void special(int key, int x, int y);
        // Manages interactivity
        void interact();
};

#endif
```

6.1.20 Keyboard.cpp

```
/*************************************************************\
 * Keyboard.cpp                                              *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the defintion of the Keyboard class.   *
 * for more information, see Keyboard.h                      *
\*************************************************************/

// Class decleration
#include "Keyboard.h"

// std::string
#include <string>

// glut really wants cstdlib here
#include <cstdlib>

// OpenGL API
#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 (!isInMain)
51         {
52                 interact(key, x, y);
53         }
54 }
55
56 void Keyboard::inputConsole(unsigned char key, int x, int y)
57 {
58         // User string input
59         static string input;
60         // Number in console history
61         static int count = 0;
62
63         // Up arrow, recieves the next older entry in the console's history
64         if (getPrev)
65         {
66                 input = HUD.getHist(count);
67
68                 if (count < HUD.getHistNum() - 1)
69                 {
70                         count++;
71                 }
72
73                 getPrev = false;
74         }
75
76         // Down arrow, recieves the next newer entry in the console's history
77         else if (getNext)
```

```
        {
                input = HUD.getHist(count);

                if (count > 0)
                {
                        count--;
                }

                getNext = false;
        }

        // Enter key, process and clear input
        else if (key == 13)
        {
                HUD.inputString(input);
                input.clear();
                count = 0;
        }

        // Tilda, close the console
        else if (key == '~' || isInConsole == false)
        {
                input.clear();
                isInConsole = false;
                HUD.toggleConsole();
                count = 0;
        }

        // Backspace. Self explanatory
        else if (key == 8 && !input.empty())
        {
                input.pop_back();
        }

        // Otherwise, type normally
        else
        {
                input += key;
        }

        // Print what's been typed so far
        HUD.printToConsole(input);
}

// Pretty much a copy pasta of inputConsole because I'm a terrible programmer
// I'll try to combine em in the future, I swear
// Just adjust all of these to do terminally stuff I guess
void Keyboard::inputTerminal(unsigned char key, int x, int y)
{
        // TODO: Fix terminal input with active Terminal hijibis

        // User string input
        static string input;
        // Number in console history
        static int count = 0;

```

```
        // Up arrow, recieves the next older entry in the console's history
        if (getPrev)
        {
                input = activeTerminal->getHist(count);

                if (count < activeTerminal->getHistNum() - 1)
                {
                        count++;
                }

                getPrev = false;
        }

        // Down arrow, recieves the next newer entry in the console's history
        else if (getNext)
        {
                input = activeTerminal->getHist(count);

                if (count > 0)
                {
                        count--;
                }

                getNext = false;
        }

        // Enter key, process and clear input
        else if (key == 13)
        {
                activeTerminal->getInput(input);
                input.clear();
                count = 0;
        }

        // Backspace. Self explanatory
        else if (key == 8 && !input.empty())
        {
                input.pop_back();
        }

        // Otherwise, type normally
        else
        {
                input += key;
        }

        // Print what's been typed so far
        activeTerminal->getText(input); // Drawing handled elsewhere?
}

void Keyboard::interact(unsigned char key, int x, int y)
{
        CollisionEngine col;
        // Speed at which the player moves
        int speedMod = 1;

```

```
        int modKey = glutGetModifiers();

        if (modKey == GLUT_ACTIVE_SHIFT)
        {
                speedMod = 2;
        }

        else
        {
                speedMod = 1;
        }

        switch (key)
        {
        case 'w':
        case 'W':
                Cam.moveForward(speedMod);
                if (col.collide())
                {
                        Cam.moveBackward(speedMod);
                }
                break;
        case 'a':
        case 'A':
                Cam.strafeRight();
                if (col.collide())
                {
                        Cam.strafeLeft();
                }
                break;
        case 's':
        case 'S':
                Cam.moveBackward(speedMod);
                if (col.collide())
                {
                        Cam.moveForward(speedMod);
                }
                break;
        case 'd':
        case 'D':
                Cam.strafeLeft();
                if (col.collide())
                {
                        Cam.strafeRight();
                }
                break;
        case 'e':
        case 'E':
                interact();
                break;
        case '~':
                isInConsole = true;
                HUD.toggleConsole();
                break;

                // Enter
```

```
        case 13:
                //goDim = true;
                break;

                // Escape
        case 27:
                isInMain = true;
                songNum = 0;
                changeSong = true;
                break;
        }
}

void Keyboard::key_up(unsigned char key, int x, int y)
{
        // I'm sure I'll do something smart here
}

void Keyboard::special(int key, int x, int y)
{
        Logger log;
        // We start in fullscreen
        static bool fullScreen = true;
        switch (key)
        {
        case GLUT_KEY_F1:
                fullScreen = !fullScreen;
                break;

        case GLUT_KEY_F2:
                // Only way to exit main loop.
                log.logLine("Exiting via F2");
                exit(EXIT_OK);
                break;

        case GLUT_KEY_F3:
                Cam.resetCam();
                break;

        case GLUT_KEY_F4:
                isInMain = !isInMain;
                break;

        case GLUT_KEY_F5:
                log.logCamCoords();
                break;

        case GLUT_KEY_UP:
                if (isInConsole || isInTerminal)
                {
                        getPrev = true;
                        getNext = false;

                        // To ensure that the input is updated BEFORE next key
                            press
                        normal(0, 0, 0);
```

```
                }
                break;

        case GLUT_KEY_DOWN:
                if (isInConsole || isInTerminal)
                {
                        getNext = true;
                        getPrev = false;

                        // To ensure that the input is updated BEFORE next key
                            press
                        normal(0, 0, 0);
                }
                break;
        }

        if (fullScreen)
        {
                glutFullScreen();
        }

        else
        {
                glutReshapeWindow(1367, 767);
                glutPositionWindow(50, 50);
        }
}

void Keyboard::interact()
{
        // Only do things if we actually can
        if (interactivity)
        {
                if (activeSwitch != NULL)
                {
                        activeSwitch->toggleTarget();

                        for (unsigned int i = 0; i < triggers.size(); i++)
                        {
                                triggers[i].tryToTrigger(activeSwitch, T_SWITCH);
                        }
                }

                else if (activeTerminal != NULL)
                {
                        isInTerminal = true;

                        for (unsigned int i = 0; i < triggers.size(); i++)
                        {
                                triggers[i].tryToTrigger(activeTerminal,
                                    T_TERMINAL);
                        }
                }
        }
}
```

6.1.21 Level.h

```
/*************************************************************\
 * Level.h                                                   *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the Level class     *
 * Which loads all level assets from a sqlite database       *
 * (data.db)                                                 *
\*************************************************************/

#ifndef LEVEL_H
#define LEVEL_H

// std;:string
#include <string>
// std::vector
#include <vector>
// Planes for walls/doors/such else
#include "Plane.h"

// SQLite API
#include "sqlite3.h"

// Glut API
#include <GL\glut.h>

class Level
{
private:
        // Used to load cylinders
        GLUquadricObj *quadratic;
        // The current level being loaded
        std::string currLevel;

        // Look, the names are self-explanatory
        void loadWalls(sqlite3 *db);
        void loadDoors(sqlite3 *db);
        void loadCylinders(sqlite3 *db);
        void loadSwitches(sqlite3 *db);
        void loadTerminals(sqlite3 *db);
        void loadTriggers(sqlite3 *db);

        // Binds the triggering object and target object to a single trigger
        bool bindTrigger(std::string id, std::string trigger, std::string
            triggerType);
        bool bindTarget(std::string id, std::string target, std::string targetType
            );

public:
        // Manages the loading of the level
        void loadLevel(std::string levelName);
        // Draws the level
        void displayLevel();
```

```
};

#endif
```

6.1.22 Level.cpp

```
/*************************************************************\
 * Level.cpp                                                 *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the defintion of the Level class.      *
 * for more information, see Level.h                         *
\*************************************************************/

// Class declaration
#include "Level.h"
// To use Planes
#include "Plane.h"
// Vectors to plop stuff in
#include "Globals.h"
// Return codes
#include "Return.h"
// System log
#include "Logger.h"
// Oject Types
#include "GCTypes.h"

#include <iostream>

using namespace std;

void Level::loadWalls(sqlite3 *db)
{
        walls.clear();
        // Prepared Statement
        sqlite3_stmt *stm;
        // SQL command
        string cmd;
        // Connection Error Test
        int err;
        cmd = "SELECT * FROM walls WHERE LEVEL = \"" + currLevel + "\"";

        err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);

        if (err != SQLITE_OK)
        {
                Logger log;
                vector<string> output = { "FATAL ERROR: failed to load walls from
                    ", currLevel };
                log.logLine(output);
                exit(STATEMENT_ERROR);
        }

        // While we still get rows of output
        while (sqlite3_step(stm) == SQLITE_ROW)
```

```
        {
                double x1, x2, x3, x4,
                        y1, y2, y3, y4,
                        z1, z2, z3, z4,
                        r, g, b, a;
                string axis;

                x1 = sqlite3_column_double(stm, 2);
                x2 = sqlite3_column_double(stm, 3);
                x3 = sqlite3_column_double(stm, 4);
                x4 = sqlite3_column_double(stm, 5);

                y1 = sqlite3_column_double(stm, 6);
                y2 = sqlite3_column_double(stm, 7);
                y3 = sqlite3_column_double(stm, 8);
                y4 = sqlite3_column_double(stm, 9);

                z1 = sqlite3_column_double(stm, 10);
                z2 = sqlite3_column_double(stm, 11);
                z3 = sqlite3_column_double(stm, 12);
                z4 = sqlite3_column_double(stm, 13);

                r = sqlite3_column_double(stm, 14);
                g = sqlite3_column_double(stm, 15);
                b = sqlite3_column_double(stm, 16);
                a = sqlite3_column_double(stm, 17);

                axis = reinterpret_cast<const char*>(sqlite3_column_text(stm, 18))
                    ;

                char ax;
                if (axis == "x") ax = 'x';
                else if (axis == "y") ax = 'y';
                else if (axis == "z") ax = 'z';
                else ax = 0;

                double verts[12] =
                {
                        x1, y1, z1,
                        x2, y2, z2,
                        x3, y3, z3,
                        x4, y4, z4
                };
                double colors[4] = { r, g, b, a };

                Plane rect(verts, colors, ax);

                walls.push_back(rect);
        }

        /*
        Logger log;
        vector<string> output = { "Loaded walls on", currLevel };
        log.logLine(output);
        */

```

```
106         // Deconstructs the statement
107         sqlite3_finalize(stm);
108 }
109
110 void Level::loadDoors(sqlite3 *db)
111 {
112         doors.clear();
113         // Prepared Statement
114         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
                        loading", currLevel };
127                 log.logLine(output);
128
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);
```

```
                a = sqlite3_column_double(stm, 17);

                a = sqlite3_column_double(stm, 17);

                axis = reinterpret_cast<const char*>(sqlite3_column_text(stm, 18));

                char ax;
                if (axis == "x") ax = 'x';
                else if (axis == "y") ax = 'y';
                else if (axis == "z") ax = 'z';
                else ax = 0;

                double verts[12] =
                {
                        x1, y1, z1,
                        x2, y2, z2,
                        x3, y3, z3,
                        x4, y4, z4
                };
                double colors[4] = { r, g, b, a };

                Plane rect(verts, colors, ax);

                doors.push_back(Door(rect, id));
        }

        Logger log;
        vector<string> output = { "Loaded doors on", currLevel };
        log.logLine(output);

        // Deconstructs the statement
        sqlite3_finalize(stm);
}

void Level::loadCylinders(sqlite3 *db)
{
        cylinders.clear();
        // Prepared Statement
        sqlite3_stmt *stm;
        // SQL command
        string cmd;
        // Connection Error Test
        int err;
        cmd = "SELECT * FROM cylinders WHERE LEVEL = \"" + currLevel + "\"";

        err = sqlite3_prepare(db, cmd.c_str(), -1, &stm, 0);

        if (err != SQLITE_OK)
        {
                Logger log;
                vector<string> output = { "FATAL ERROR: Can't load cylinders while loading", currLevel };
                log.logLine(output);

                exit(STATEMENT_ERROR);
```

```
        }

        // While we still get rows of output
        while (sqlite3_step(stm) == SQLITE_ROW)
        {
                double xt, yt, zt,
                        xr, yr, zr,
                        r, g, b, a,
                        baseRadius, topRadius, height;
                int stacks, slices;


                xt = sqlite3_column_double(stm, 1);
                yt = sqlite3_column_double(stm, 2);
                zt = sqlite3_column_double(stm, 3);

                xr = sqlite3_column_double(stm, 4);
                yr = sqlite3_column_double(stm, 5);
                zr = sqlite3_column_double(stm, 6);

                baseRadius = sqlite3_column_double(stm, 7);
                topRadius = sqlite3_column_double(stm, 8);
                height = sqlite3_column_double(stm, 9);

                stacks = sqlite3_column_int(stm, 10);
                slices = sqlite3_column_int(stm, 11);

                r = sqlite3_column_double(stm, 12);
                g = sqlite3_column_double(stm, 13);
                b = sqlite3_column_double(stm, 14);
                a = sqlite3_column_double(stm, 15);


                double translate[3] = { xt, yt, zt };
                double rotate[3] = { xr, yr, zr };
                double colors[4] = { r, g, b, a };

                cylinders.push_back(Cylinder(baseRadius, topRadius, height, stacks
                    , slices, translate, rotate, colors));
        }

        Logger log;
        vector<string> output = { "Loaded cylinders on", currLevel };
        log.logLine(output);

        // Deconstructs the statement
        sqlite3_finalize(stm);
}


void Level::loadSwitches(sqlite3 *db)
{
        switches.clear();
        // Prepared Statement
        sqlite3_stmt *stm;
        // 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));
                targetType = reinterpret_cast<const char*>(sqlite3_column_text(stm, 5));

                if (triggerType == "SWITCH")
                        i_triggerType = T_SWITCH;
                else if (triggerType == "TERMINAL")
                        i_triggerType = T_TERMINAL;
                else
                {
                        Logger log;
                        vector<string> output = { "Failed to evaluate string trigger type entry: ", triggerType, "for trigger ", id };
                        log.logLine(output);

                        exit(DATA_ENTRY_ERROR);
                }

                if (targetType == "SWITCH")
                        i_targetType = T_SWITCH;
                else if (targetType == "TERMINAL")
                        i_targetType = T_TERMINAL;
                else
                {
                        Logger log;
                        vector<string> output = { "Failed to evaluate string trigger type entry: ", targetType, "for trigger ", id };
                        log.logLine(output);

                        exit(DATA_ENTRY_ERROR);
                }

                triggers.push_back(Trigger(i_triggerType, i_targetType));

                bool assigned = bindTrigger(id, trigger, triggerType) && bindTarget(id, target, targetType);

                if (!assigned)
                {
                        Logger log;
                        vector<string> output = { "Failed to bind trigger ", id };
                        log.logLine(output);

                        exit(BINDING_ERROR);
                }
        }

        Logger log;
        vector<string> output = { "Loaded trigger on", currLevel };
        log.logLine(output);

        // Deconstructs the statement
        sqlite3_finalize(stm);
}
```

```
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,
                                        " to trigger-switch", trigger };
540                                 log.logLine(output);
541
542                                 triggers[triggers.size() - 1].bindTrigger(&(
                                        switches[i]));
543
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,
                                        " to trigger-terminal", trigger };
557                                 log.logLine(output);
558
559                                 triggers[triggers.size() - 1].bindTrigger(&(
                                        terminals[i]));
560
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,
                                        " to target-switch", target };
```

```
580                                 log.logLine(output);
581
582                                 triggers[triggers.size() - 1].bindTarget(&(
                                        switches[i]));
583
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 };
597                                 log.logLine(output);
598
599                                 triggers[triggers.size() - 1].bindTarget(&(
                                        terminals[i]));
600
601                                 return true;
602                         }
603                 }
604         }
605
606         return false;
607 }
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         interactivity = false;
655
656         // Get out of wall
657         for (unsigned int i = 0; i < 10; i++)
658         {
659                 Cam.moveForward(1);
660         }
661
662         // Go dim for 5 seconds
663         HUD.goDim(5);
664 }
665
666 void Level::displayLevel()
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
688         for (auto i : walls)
```

```
        {
                i.Display();
        }
}
```

6.1.23 Logger.h

```
/*************************************************************\
 * Logger.h                                                  *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the Logger class    *
 * Which writes messages to output.log because it's more     *
 * Reliable than stdout
\*************************************************************/

#ifndef LOGGER_H
#define LOGGER_H

// To help find the user's document folder
#include <shlobj.h>

// std::vector
#include <vector>
// std::string
#include <string>

class Logger
{
private:
        // Path to the log file
        char CHAR_PATH[MAX_PATH];
        std::string LOG_PATH;

public:
        Logger();
        // Erases the log file, called at the beggining of the program
        void nuke();
        // Writes to the log, either multiple lines or one line
        void logLine(std::vector<std::string> input);
        void logLine(std::string input);
        // Writes the Camera Coordinates to the log file
        void logCamCoords();

};

#endif
```

6.1.24 Logger.cpp

```
/*************************************************************\
 * Logger.cpp                                                *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
```

```
 *                                                          *
 * This file contains the defintion of the Logger class.     *
 * for more information, see Logger.h                        *
\*********************************************************/

// Class declaration
#include "Logger.h"
// For Cam coords
#include "Globals.h"
// File I/O
#include <fstream>

#include <iostream>

using namespace std;

Logger::Logger()
{
        HRESULT ret = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL,
            SHGFP_TYPE_CURRENT, CHAR_PATH);
        LOG_PATH = CHAR_PATH;
        LOG_PATH += "\\The God Core\\output.log";
}

void Logger::nuke()
{
        ofstream outfile(LOG_PATH); // Nukes everything within
}

void Logger::logLine(vector<string> input)
{
        ofstream outfile(LOG_PATH, ios::app);

        string output;

        for (auto i : input)
        {
                output += i;
                output += " ";
        }
        outfile << output << std::endl;
}

void Logger::logLine(string input)
{
        ofstream outfile(LOG_PATH, ios::app);

        outfile << input << std::endl;
}

void Logger::logCamCoords()
{
        ofstream outfile(LOG_PATH, ios::app);

        outfile << "Player Coordinates:\n";
        outfile << "X: " << -Cam.x << endl;
```

```
        outfile << "Y: " << -Cam.y << endl;
        outfile << "Z: " << -Cam.z << endl;
}
```

6.1.25 MainMenu.h

```
/*********************************************************\
 * MainMenu.h                                            *
 * This file was created by Jeremy Greenburg             *
 * As part of The God Core game for the University of    *
 * Tennessee at Martin's University Scholars Organization *
 *                                                       *
 * This file contains the decleration of the MainMenu class *
 * Which uses the Simple OpenGL Interface Library to load a *
 * png picture of the main menu, as well as provide button *
 * Interactivity                                         *
\*********************************************************/

#ifndef MAIN_MENU_H
#define MAIN_MENU_H

// For loading pictures
#include <SOIL.h>
// Inherit 2D functionality
#include "TwoD.h"

// Make OpenGL happy
#include <cstdlib>
// openGL API
#include <GL\glut.h>

class MainMenu : public TwoD
{
public:
        // Loads the picture up in memory
        MainMenu();
        // Handles drawing to the screen
        void display();
        // Handles and processes mouse clicks
        void getClick(double x, double y);

private:
        // Draws the main picture
        void drawMainPic();
        // DEBUG: draws boxes around all buttons
        void drawClickBoxes();
        // What the picture is bound to
        GLint texture;
};

#endif
```

6.1.26 MainMenu.cpp

```
/*********************************************************\
 * MainMenu.cpp                                          *
 * This file was created by Jeremy Greenburg             *
```

```
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the defintion of the MainMenu class.   *
 * for more information, see MainMenu.h                      *
\*********************************************************/

// Class declaration
#include "MainMenu.h"
// isInMain
#include "Globals.h"
// Return codes
#include "Return.h"
// System log
#include "Logger.h"

#include "SaveManager.h"

using namespace std;

MainMenu::MainMenu()
{
        texture = SOIL_load_OGL_texture
                        (
                                "Resources\\Images\\Main.png",   // Image to load
                                SOIL_LOAD_AUTO,                                  //
                                     ???
                                SOIL_CREATE_NEW_ID,
                                SOIL_FLAG_MIPMAPS | SOIL_FLAG_NTSC_SAFE_RGB |
                                    SOIL_FLAG_COMPRESS_TO_DXT // !?!?!?!
                        );

        if (texture == 0)
        {
                Logger log;
                vector<string> output = {"FATAL ERROR: SOIL cannot load image",
                    SOIL_last_result()};
                log.logLine(output);
                exit(SOIL_ERROR);
        }
}

void MainMenu::drawMainPic()
{
        glEnable(GL_TEXTURE_2D);

        glBindTexture(GL_TEXTURE_2D, texture); // Prepares the texture for usage

        glColor3d(1, 1, 1);
        glBegin(GL_QUADS);
        glTexCoord2d(0, 0);     glVertex2d(SCREENLEFT, SCREENTOP);
        glTexCoord2d(0, 1); glVertex2d(SCREENLEFT, SCREENBOTTOM);
        glTexCoord2d(1, 1); glVertex2d(SCREENRIGHT, SCREENBOTTOM);
        glTexCoord2d(1, 0);     glVertex2d(SCREENRIGHT, SCREENTOP);

        glEnd();
```

```
57
58          glDisable(GL_TEXTURE_2D);
59
60  }
61
62  void MainMenu::drawClickBoxes()
63  {
64          glColor3d(1, 0, 0);
65
66          // Start a new game
67          glBegin(GL_LINE_LOOP);
68          glVertex2d(SCREENRIGHT / 20.0, SCREENBOTTOM / 2.2);
69          glVertex2d(SCREENRIGHT / 20.0, SCREENBOTTOM / 1.9);
70          glVertex2d(SCREENRIGHT / 3.0, SCREENBOTTOM / 1.9);
71          glVertex2d(SCREENRIGHT / 3.0, SCREENBOTTOM / 2.2);
72          glEnd();
73
74          // Load game
75          glBegin(GL_LINE_LOOP);
76          glVertex2d(SCREENRIGHT / 10.0, SCREENBOTTOM / 1.57);
77          glVertex2d(SCREENRIGHT / 10.0, SCREENBOTTOM / 1.75);
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 / 6.5, SCREENBOTTOM / 1.16);
94          glVertex2d(SCREENRIGHT / 6.5, SCREENBOTTOM / 1.25);
95          glVertex2d(SCREENRIGHT / 4.5, SCREENBOTTOM / 1.25);
96          glVertex2d(SCREENRIGHT / 4.5, SCREENBOTTOM / 1.16);
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 = 1;
109                         changeSong = true;
110                         curr_level = "LEVELZERO";
111                         loading = true;
112                 }
```

```
113         }
114
115         // Load Game
116         if (x >= SCREENRIGHT / 10.0 && x <= SCREENRIGHT / 3.5)
117         {
118                 if (y >= SCREENBOTTOM / 1.75 && y <= SCREENBOTTOM / 1.57)
119                 {
120                         SaveManager Jesus; // Jesus Saves
121                         if (!Jesus.loadGame()); // null
122                         else isInMain = false;
123
124                 }
125         }
126
127         // Options
128         if (x >= SCREENRIGHT / 8.5 && x <= SCREENRIGHT / 3.9)
129         {
130                 if (y >= SCREENBOTTOM / 1.45 && y <= SCREENBOTTOM / 1.35)
131                 {
132                         // Jokes on me I never did get any options up
133                 }
134         }
135
136         // Exit
137         if (x >= SCREENRIGHT / 6.5 && x <= SCREENRIGHT / 4.5)
138         {
139                 if (y >= SCREENBOTTOM / 1.25 && y <= SCREENBOTTOM / 1.16)
140                 {
141                         exit(EXIT_OK);
142                 }
143         }
144 }
145
146 void MainMenu::display()
147 {
148         prepare2D();
149
150         drawMainPic();
151
152         // Disable once finished
153         //drawClickBoxes();
154
155         glEnd();
156
157         prepare3D();
158 }
```

6.1.27 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    *
```

```
 * Memory, play them when called, and release the memory     *
 * When the song is no longer needed.                        *
\*************************************************************/

#ifndef MUSICMANAGER_H
#define MUSICMANAGER_H

// FMOD API
#include <fmod.hpp>

// Creates new type for ease of use
typedef FMOD::Sound* SoundClass;

class MusicManager
{
private:
        // Pointer to dynamic system memory to load music
        FMOD::System *m_pSystem;

        // The path to the music folder
        static const char* MUSIC_PATH;

public:
        // Loads the song in memory
        void makeSound(SoundClass *psound, const char *song);
        // Plays the song (Always loops)
        void playSound(SoundClass pSound, bool bLoop = true);
        // Releases the song
        void releaseSound(SoundClass psound);
        // Initializes FMOD
        MusicManager();
};

#endif
```

6.1.28 MusicManager.cpp

```
/*************************************************************\
 * MusicManager.cpp                                          *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the definition of the MusicManager     *
 * Class. For more information, see MusicManager.h           *
\*************************************************************/

// Class definition
#include "MusicManager.h"

// Because concatenating char*'s are really hard
#include <string>

// Return codes
#include "Return.h"

// 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)
```

```
{
        pSound->release();
}
```

6.1.29 Plane.h

```
/*************************************************************\
 * Plane.h                                                   *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the Plane class     *
 * Which is used to hold the details of a 2D Plane and       *
 * draw  it to the screen                                    *
\*************************************************************/

#ifndef Plane_H
#define Plane_H

class Plane
{
private:
        // Arrays containing the color and vertices of the Plane
        double color[4];
        // What axis is it aligned on (x y z)
        char axis;
        // The vertices of the corners
        double vertices[12];
public:

        // Paramaterized constructor, as there cannot be a Plane without vertices
        // Can take an axis or can ignore exis
        Plane(const double(&new_vertices)[12], const double(&new_color)[4], char
            _axis);
        Plane(const double(&new_vertices)[12], const double(&new_color)[4]);

        // Part of the plane equation, calculated in constructor
        double a, b, c, d;

        // Determines if the player is in the bounds of the Plane (based on axis)
        bool isInBounds();

        // Returns the plane norm (Perpindicular line)
        double getNorm();

        // Mutate's the rectangles coordinates for the end of the game
        void mutate();

        // Print a Plane in 3D
        void Display();
        // Print a Plane in 2D
        void Display2D();
};

#endif
```

6.1.30 Plane.cpp

```
/*********************************************************\
 * Plane.cpp                                             *
 * This file was created by Jeremy Greenburg             *
 * As part of The God Core game for the University of    *
 * Tennessee at Martin's University Scholars Organization *
 *                                                       *
 * This file contains the definition of the Plane class  *
 * For more information, see Plane.h                     *
\*********************************************************/

#include "Plane.h"

// For std::copy
#include <iterator>
#include <utility>

// max and min
#include <algorithm>

// OpenGL API
#include <GL\glut.h>

// For Cam coords
#include "Globals.h"

using namespace std;

Plane::Plane(const double (&new_vertices)[12], const double (&new_color)[4], char
    _axis)
{
        // Copies the color
        copy(begin(new_color), end(new_color), color);

        // Copies the vertices
        copy(begin(new_vertices), end(new_vertices), vertices);


        // Somedays I wonder what I'm even doing \\
        // When I forget what all this means: http://keisan.casio.com/exec/system
            /1223596129 \\

        // Calculate vector equation ax + by + cz + d = 0
        // Get two vectors from three of the corners
        double AB[] = { vertices[3] - vertices[0], vertices[4] - vertices[1],
            vertices[5] - vertices[2] };
        double AC[] = { vertices[6] - vertices[0], vertices[7] - vertices[1],
            vertices[8] - vertices[2] };
        // Cross Product of AB and AC
        a = (AB[1] * AC[2]) - (AB[2] * AC[1]);
        b = (AB[2] * AC[0]) - (AB[0] * AC[2]);
        c = (AB[0] * AC[1]) - (AB[1] * AC[0]);
        d = (a * vertices[0] + b * vertices[1] + c * vertices[2]);

        axis = _axis;
```

```
}

Plane::Plane(const double(&new_vertices)[12], const double(&new_color)[4])
{
        // Copies the color
        copy(begin(new_color), end(new_color), color);

        // Copies the vertices
        copy(begin(new_vertices), end(new_vertices), vertices);


        // Somedays I wonder what I'm even doing \\
                // When I forget what all this means: http://keisan.casio.com/exec/system/1223596129 \\

// Calculate vector equation ax + by + cz + d = 0
// Get two vectors from three of the corners
        double AB[] = { vertices[3] - vertices[0], vertices[4] - vertices[1], vertices[5] - vertices[2] };
        double AC[] = { vertices[6] - vertices[0], vertices[7] - vertices[1], vertices[8] - vertices[2] };
        // Cross Product of AB and AC
        a = (AB[1] * AC[2]) - (AB[2] * AC[1]);
        b = (AB[2] * AC[0]) - (AB[0] * AC[2]);
        c = (AB[0] * AC[1]) - (AB[1] * AC[0]);
        d = (a * vertices[0] + b * vertices[1] + c * vertices[2]);

        axis = 0;
}

void Plane::Display()
{
        // Set's OpenGL's color to the color of the Plane
        glColor4f(color[0], color[1], color[2], color[3]);

        glBegin(GL_QUADS);
        glVertex3d(vertices[0], vertices[1], vertices[2]);
        glVertex3d(vertices[3], vertices[4], vertices[5]);
        glVertex3d(vertices[6], vertices[7], vertices[8]);
        glVertex3d(vertices[9], vertices[10], vertices[11]);
        glEnd();
}

void Plane::Display2D()
{
        glColor4f(color[0], color[1], color[2], color[3]);

        glBegin(GL_QUADS);
        glVertex2d(vertices[0], vertices[1]);
        glVertex2d(vertices[3], vertices[4]);
        glVertex2d(vertices[6], vertices[7]);
        glVertex2d(vertices[9], vertices[10]);
        glEnd();
}

bool Plane::isInBounds()
```

```
{
        if (axis == 'x')
        {
                vector<double> X = { vertices[0], vertices[3], vertices[6], vertices[9] };
                double maxX = *max_element(X.begin(), X.end());
                double minX = *min_element(X.begin(), X.end());

                return (-Cam.x <= maxX && -Cam.x >= minX);

        }

        else if (axis == 'y')
        {
                vector<double> Y = { vertices[1], vertices[4], vertices[7], vertices[10] };
                double maxY = *max_element(Y.begin(), Y.end());
                double minY = *min_element(Y.begin(), Y.end());

                return (-Cam.y <= maxY && -Cam.x >= minY);
        }

        else if (axis == 'z')
        {
                vector<double> Z = { vertices[2], vertices[5], vertices[8], vertices[11] };
                double maxZ = *max_element(Z.begin(), Z.end());
                double minZ = *min_element(Z.begin(), Z.end());

                return (-Cam.z <= maxZ && -Cam.z >= minZ);
        }
        else return false;
}

double Plane::getNorm()
{
        return sqrt(a * a + b * b + c * c);
}

void Plane::mutate()
{
        // We're gonna mess stuff up, disable the axis so nothing funky happens with collision
        axis = ' ';

        for (unsigned int i = 0; i < 12; i++)
        {
                // 0 <= mutator <= 200
                double mutator = rand() % 201;
                // -100 <= mutator <= 100
                mutator -= 100;
                // -.01 <= mutator <= .01
                mutator /= 10000;

                vertices[i] += mutator;
        }
```

```
}
```

6.1.31 Return.h

```
/*************************************************************\
 * Return.h                                                  *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains varius return codes for when things    *
 * Go horribly wrong (and they do)                           *
 * (just hopefully not during my senior defense)             *
\*************************************************************/

#ifndef RETURN_H
#define RETURN_H

#define EXIT_OK 0 // Indicates an intended exit
#define EXIT_EARLY 1 // If we exit OpenGL main loop early
#define FMOD_ERROR 2 // Fmod can't load sound
#define DATABASE_ERROR 3 // sqlite can't load database
#define STATEMENT_ERROR 4 // sqlite statement fails to execute
#define SOIL_ERROR 5 // SOIl fails to load image
#define DATA_ENTRY_ERROR 6 // Indicates an internal error in a database entry
#define BINDING_ERROR 7 // Error binding a trigger
#define FILE_NOT_FOUND 8 // A file is not found

#endif
```

6.1.32 Resource.h

```
/*************************************************************\
 * Return.h                                                  *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains varius return codes for when things    *
 * Go horribly wrong (and they do)                           *
 * (just hopefully not during my senior defense)             *
\*************************************************************/

#ifndef RETURN_H
#define RETURN_H

#define EXIT_OK 0 // Indicates an intended exit
#define EXIT_EARLY 1 // If we exit OpenGL main loop early
#define FMOD_ERROR 2 // Fmod can't load sound
#define DATABASE_ERROR 3 // sqlite can't load database
#define STATEMENT_ERROR 4 // sqlite statement fails to execute
#define SOIL_ERROR 5 // SOIl fails to load image
#define DATA_ENTRY_ERROR 6 // Indicates an internal error in a database entry
#define BINDING_ERROR 7 // Error binding a trigger
#define FILE_NOT_FOUND 8 // A file is not found

#endif
```

6.1.33 SaveManager.h

```
/*************************************************************\
 * SaveManager.h                                             *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the SaveManager     *
 * Class, which saves data by encrypting an array of strings *
 * And writing them to core.sav, or by reading in an array of*
 * Strings from core.sav and decrypting them                 *
\*************************************************************/

#ifndef SAVEMANAGER_H
#define SAVEMANAGER_H

// Windows API
#include <shlobj.h>

// Because concatenating char*'s is really hard
#include <string>

class SaveManager
{
private:
        // The path to core.sav
        char CHAR_PATH[MAX_PATH];
        std::string SAVE_PATH;

        // Takes an unencrypted string and returns an encrypted string
        std::string encrytData(std::string data);
        // Takes an encrypted string and returns a decrypted string
        std::string decryptData(std::string data);
public:
        SaveManager();
        // Writes the array of encrypted strings to core.sav
        void saveLevel();
        // Sets global variables to load game
        bool loadGame();
        // Returns the decrypted string in core.sav
        std::string readSave();
        // Returns true if core.save exists
        bool checkSave();
};

#endif
```

6.1.34 SaveManager.cpp

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

```
 * This file contains the definition of the SaveManager class*
 * For more information, see SaveManager.h                   *
\*************************************************************/

// Class definition
#include "SaveManager.h"

// File I/O
#include <fstream>

#include "Globals.h"

#include "Logger.h"

using namespace std;

SaveManager::SaveManager()
{
        HRESULT ret = SHGetFolderPath(NULL, CSIDL_PERSONAL, NULL,
            SHGFP_TYPE_CURRENT, CHAR_PATH);
        SAVE_PATH = CHAR_PATH;
        SAVE_PATH += "\\The God Core\\core.sav";
}

string SaveManager::encrytData(string data)
{
        string ret_str;
        for (unsigned int i = 0; i < data.length()*3; i+=3)
        {
                ret_str += data[i/3] + 48;
                ret_str += data[i/3] - 48;
                ret_str += data[i/3] + 53;
        }
        return ret_str;
}

string SaveManager::decryptData(string data)
{
        string ret_str;
        for (unsigned int i = 0; i < data.length(); i+=3)
        {
                ret_str += data[i] - 48;
        }

        return ret_str;
}

string SaveManager::readSave()
{
        Logger log;

        ifstream save(SAVE_PATH);
        log.logLine("Checking Save integrity.");

        string enc_data; // Encrypted Data
        string dcr_data; // Decrypted Data
```

```
        save >> enc_data;// Read encrypted data from file
        dcr_data = decryptData(enc_data); // Decrypt data

        vector<string> output{ "Decrypted Data: ", dcr_data };
        log.logLine(output);

        save.close();

        return dcr_data;
}

void SaveManager::saveLevel()
{
        ofstream save(SAVE_PATH);

        string input = curr_level + " " + to_string(songNum);

        string encr_str = encrytData(input);

        save << encr_str;

        save.close();
}

bool SaveManager::loadGame()
{
        // might change to vector<string> later
        string data = readSave();
        size_t pos = data.find(' ');

        if (pos == string::npos) return false;
        string savedLevel = data.substr(0, pos);
        int savedSong = stoi(data.substr(pos + 1));

        int temp_levelNum = getLevelNum(savedLevel);

        if (temp_levelNum == -1) return false;

        levelNum = temp_levelNum;
        curr_level = getLevelString(levelNum);
        songNum = savedSong;

        loading = true;
        changeSong = true;

        return true;
}

bool SaveManager::checkSave()
{
        ifstream save(SAVE_PATH);

        if (save)
        {
                return true;
        }
```

```

        else
        {
                return false;
        }
}
```

6.1.35 Switch.h

```
/*********************************************************\
 * Switch.h                                              *
 * This file was created by Jeremy Greenburg             *
 * As part of The God Core game for the University of    *
 * Tennessee at Martin's University Scholars Organization *
 *                                                       *
 * This file contains the declaration of the Switch class *
 * Which is bound to a Door via pointer and can open and *
 * Close the door at will                                *
\*********************************************************/

#ifndef SWITCH_H
#define SWITCH_H

// Door class
#include "Door.h"
#include "PoweredObject.h"
// Terminal Class
#include "Terminal.h"

// Types
#include "GCTypes.h"

class Switch  : public PoweredObject
{
private:
        void* target; // The door that this switch activates
        // Translation and rotation coordinates
        double translate[3], rotate[3];

        // One of the predefined types
        GCtype targetType;

        // Unique ID
        std::string id;

public:
        // Initializes the translation and rotation matrices
        Switch(const double(&_translate)[3], const double(&_rotate)[3], GCtype
            _type, std::string _id, bool _isOn);
        // Binds the target pointer to an object
        void assign(void* _target);
        // Opens/Closes the door
        void toggleTarget();
        // Actually draws the switch
        void Display();

        // Get's the switch's ID
```

```
        std::string getID();

        // Gets the translation coordinates
        double getX();
        double getY();
        double getZ();
};

#endif
```

6.1.36 Switch.cpp

```
/*********************************************************\
 * Switch.cpp                                            *
 * This file was created by Jeremy Greenburg             *
 * As part of The God Core game for the University of    *
 * Tennessee at Martin's University Scholars Organization *
 *                                                       *
 * This file contains the definition of the Switch class *
 * For more information, see Switch.h                    *
\*********************************************************/

// Class decleration
#include "Switch.h"

// Allows copying arrays
#include <iterator>
#include <utility>
#include <algorithm>

#include "Globals.h"

// OpenGL API
#include <GL\glut.h>

using namespace std;

Switch::Switch(const double(&_translate)[3], const double(&_rotate)[3], GCtype
    _type, string _id, bool _isOn)
{
        // Copies the color
        copy(begin(_translate), end(_translate), translate);

        // Copies the vertices
        copy(begin(_rotate), end(_rotate), rotate);

        targetType = _type;

        target = NULL;

        id = _id;

        if (_isOn) activate();
        else deactivate();

}

```

```
void Switch::assign(void* _target)
{
        target = _target;
}

void Switch::toggleTarget()
{
        switch (targetType)
        {
                case T_DOOR:
                {
                        Door* t = (Door*)target;
                        t->isOpen = !t->isOpen;
                        break;
                }
                case T_TERMINAL:
                {
                        Terminal* t = (Terminal*)target;
                        t->toggle();
                        break;
                }
                case T_LEVEL_END:
                {
                        levelNum++;
                        curr_level = getLevelString(levelNum);
                        loading = true;

                        // TEMP
                        advanceMusic();
                        changeSong = true;
                }
        }
}

void Switch::Display()
{
        glPushMatrix();
        glTranslated(translate[0], translate[1], translate[2]);
        glRotated(rotate[0], 1, 0, 0);
        glRotated(rotate[1], 0, 1, 0);
        glRotated(rotate[2], 0, 0, 1);

        glColor3d(0.9, 0.9, 0.9);
        glutSolidCube(.1);

        switch (targetType)
        {
        case T_DOOR:
                glColor3d(0, 1, 0);
                break;
        case T_TERMINAL:
                glColor3d(1, 0, 0);
                break;
        default:
                glColor3d(0, 0, 1);
        }
```

```
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 }
```

6.1.37 Terminal.h

```
1  /************************************************************\
2   * Terminal.h                                               *
3   * This file was created by Jeremy Greenburg                *
4   * As part of The God Core game for the University of       *
5   * Tennessee at Martin's University Scholars Organization   *
6   *                                                          *
7   * This file contains the declaration of the Terminal class *
8   * Which draws and manages ingame computer terminals        *
9   * And has nothing to do with terminal illness I swear      *
10  \************************************************************/
11
12 #ifndef TERMINAL_H
13 #define TERMINAL_H
14
15 #include "TwoD.h" // To inherit 2D class
16 #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
37         // Where to print each item
38         const double INPUT_LINE = SCREENBOTTOM / 7.0;
39         const double ERROR_LINE = INPUT_LINE - 30;
40         const double PROMPT_START = INPUT_LINE + 30;
41         const double CONTENT_START = PROMPT_START + 150;
42
43         // The banner texture
44         GLint bTexture;
45
46         // The user inputed number
47         int num;
48
49         // Print our text
50         TextEngine text;
51
52         // Translation and rotation matrices
53         double translate[3], rotate[3];
54
55         // Draws the actual terminal
56         void draw();
57
58         // Draws a standing terminal
59         void drawStanding();
60
61         // Draws a wall mounter terminal
62         void drawWallMounted();
63
64         // Processes the user input
65         void processInput();
66
67         // Parse the terminal file
68         void parseFile();
69
70         // The path to the Terminal Files
71         static const char* TERM_PATH;
72
73 public:
74         // Draws the 3D object in the world
75         void Display();
76         // Draws the 2D Terminal screen
77         void DisplayScreen();
78         // Shows the currently typed string
79         void getText(std::string text);
80         // Signifies a completed string to process
```

```
        void getInput(std::string text);
        // Returns an item in the terminal's log
        std::string getHist(int count);
        // Returns the number of items in the terminal's log
        int getHistNum();

        // Gets the translation coordinates
        double getX();
        double getY();
        double getZ();

        // Get the terminal's ID
        std::string getID();

        // To construct and initialize the terminal
        Terminal(const double(&_translate)[3], const double(&_rotate)[3], std::
            string _file, std::string _id);

};

#endif
```

6.1.38 Terminal.cpp

```
/*************************************************************\
 * Terminal.cpp                                              *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the definition of the Terminal class   *
 * For more information, see Terminal.h                      *
\*************************************************************/

//
// Class declaration
#include "Terminal.h"

// Planes
#include "Plane.h"

// For system logging
#include "Logger.h"

// Return codes
#include "Return.h"

// Global variables
#include "Globals.h"

// Logger
#include "Logger.h"

// File I/O
#include <fstream>

using namespace std;
```

```

const char* Terminal::TERM_PATH = "Resources\\Text\\";

void Terminal::getText(std::string text)
{
        currentText = text;
}

void Terminal::getInput(std::string text)
{
        currentInput = text;
}

string Terminal::getHist(int count)
{
        int size = history.size();
        if (history.empty())
        {
                return "";
        }

        // If, somehow, a fool manages to get a variable that is out of bounds

        else if (count >= size)
        {
                return history.back();
        }

        else if (count < 0)
        {
                return history.front();
        }

        else
        {
                return history[size - count - 1];
        }
}

int Terminal::getHistNum()
{
        return history.size();
}

void Terminal::draw()
{
        // Completely black background
        double colors[4] = { 0, 0, 0, 1 };
        double vertices[12] =
        {
                SCREENLEFT, SCREENTOP, -1,
                SCREENLEFT, SCREENBOTTOM, -1,
                SCREENRIGHT, SCREENBOTTOM, -1,
                SCREENRIGHT, SCREENTOP, -1
        };

```

```
        Plane background{ vertices, colors};
        background.Display2D();


        // Gotta do the banner manually
        glEnable(GL_TEXTURE_2D);

        glBindTexture(GL_TEXTURE_2D, bTexture); // Prepares the texture for usage

        glColor3d(1, 1, 1);
        glBegin(GL_QUADS);
        glTexCoord2d(0, 0);     glVertex2d(SCREENLEFT, SCREENTOP);
        glTexCoord2d(0, 1); glVertex2d(SCREENLEFT, SCREENBOTTOM / 9.0);
        glTexCoord2d(1, 1); glVertex2d(SCREENRIGHT, SCREENBOTTOM / 9.0);
        glTexCoord2d(1, 0);     glVertex2d(SCREENRIGHT, SCREENTOP);

        glEnd();

        glDisable(GL_TEXTURE_2D);
}

void Terminal::DisplayScreen()
{
        prepare2D();

        draw();

        // If we need to proces a command
        if (currentInput != "")
        {
                processInput();

                history.push_back(currentInput);

                currentInput.clear();
        }

        else
        {
                // Print all prompts
                for (unsigned int i = 0; i < prompts.size(); i++)
                {
                        text.printString(SCREENLEFT, PROMPT_START + 20 * i, 0, 1,
                            0, prompts[i]);
                }

                // Print an error
                text.printString(SCREENLEFT, ERROR_LINE, 1, 0, 0, error);
                // Echo user text
                text.printString(SCREENLEFT, INPUT_LINE, 0, 1, 0, ":> " +
                    currentText);

                // If needed, print content
                if (num != -1 && num < (signed int)content.size())
                {
                        text.openFile(SCREENLEFT, CONTENT_START, 0, 1, 0, file,
```

```
                                content[num]);
                }
        }

        prepare3D();
}

void Terminal::processInput()
{
        error = "";
        if (currentInput == "exit" || currentInput == "Exit")
        {
                isInTerminal = false;
                history.clear();
        }

        else if (currentInput == "clear" || currentInput == "Clear")
        {
                num = -1;
        }

        else if (currentInput == "help" || currentInput == "Help")
        {
                num = 0;
        }

        else
        {
                string first, last;
                size_t pos = currentInput.find(" ");

                first = currentInput.substr(0, pos); // First half of string
                last = currentInput.substr(pos + 1); // Second half of string

                if (first == "read" || first == "Read")
                {
                        num = atoi(last.c_str());
                        if (num <= 0 || num >= (signed int)prompts.size())
                        {
                                error = "ERROR: Invalid file number";
                                num = -1;
                        }
                }

                else
                {
                        error = "ERROR: Invalid Command: " + currentInput;
                        num = -1;
                }
        }
}

void Terminal::Display()
{
        // Add two styles - Standing and wall mounted
        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                 log.logLine(output);
283                 exit(FILE_NOT_FOUND);
284         }
285
286         content.push_back("HELP"); // Help text is always the 0th tag in the
                terminals
287
288         getline(infile, buff);
289         prompts.push_back(buff); // Push back the file tag
290         getline(infile, buff);
291
292         while (buff != "<TAGS>")
293         {
294                 size_t pos = buff.find("--");
295                 if (pos != string::npos)
296                 {
297                         prompts.push_back(buff.substr(0, pos));
298                         content.push_back(buff.substr(pos + 3));
299                 }
300                 getline(infile, buff);
301         }
302
303 }
304
305 string Terminal::getID()
306 {
307         return id;
308 }
```

```

Terminal::Terminal(const double(&_translate)[3], const double(&_rotate)[3], string _file, string _id)
{
        // Copies the color
        copy(begin(_translate), end(_translate), translate);

        // Copies the vertices
        copy(begin(_rotate), end(_rotate), rotate);

        bTexture = SOIL_load_OGL_texture
                (
                        "Resources\\Images\\banner.png",   // Image to load
                        SOIL_LOAD_AUTO,                                // ???
                        SOIL_CREATE_NEW_ID,
                        SOIL_FLAG_MIPMAPS | SOIL_FLAG_COMPRESS_TO_DXT // !?!?!?!
                        );

        if (bTexture == 0)
        {
                Logger log;
                vector<string> output = { "FATAL ERROR: SOIL cannot load terminal banner", SOIL_last_result() };
                log.logLine(output);
                exit(SOIL_ERROR);
        }

        file = _file;

        id = _id;

        num = 0;

        parseFile();
}
```

6.1.39 TextEngine.h

```
/*************************************************************\
 * TextEngine.h                                              *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the TextEngine class*
 * Which uses glutBitmapCharacter to print strings into the  *
 * OpenGL window.                                            *
\*************************************************************/

#ifndef TEXTENGINE_H
#define TEXTENGINE_H

// For string lengths in displaying text
#include <string>

// For multiple lines of text
#include <vector>
```

```

class TextEngine
{
private:
        // The path to the game's text files (.log's)
        static const char* TEXT_PATH;
        // The offset between lines of characters
        static const double LINE_OFFSET;

        void displayText(
                // 2d start location of the text
                double x, double y,
                // rgb color of text
                double r, double g, double b,
                // glut font and text to be displayed
                void* font,
                std::vector<std::string> text);

        // Searches a text file for text related to the tag, and returns all text within the tag
        std::vector<std::string> findText(std::string fileName, std::string tag);

public:
        // Takes the location to display the text, color of the text,
        // The file to read from, and a tag to search for
        void openFile(double x, double y, double r, double g, double b,
                std::string fileName, std::string tag);

        // Takes in a string to display
        void printString(double x, double y, double r, double g, double b,
                std::string text);

        // Returns text from fileName specified by tag
        std::vector<std::string> getText(std::string fileName, std::string tag);
};

#endif
```

6.1.40 TextEngine.cpp

```
/*************************************************************\
 * TextEngine.cpp                                            *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the definition of the TextEngine class *
 * For more information, see TextEngine.h                    *
\*************************************************************/

// TextEngine declaration and std::string
#include "TextEngine.h"

// std::ifstream
#include <fstream>

// Standard I/O for debugging
```

```
#include <iostream>

// OpenGL API
#include <gl\glut.h>

using namespace std;

// Initializing the constants
const char* TextEngine::TEXT_PATH = "Resources\\Text\\";
const double TextEngine::LINE_OFFSET = 20;

void TextEngine::displayText(double x, double y,
        double r, double g, double b,
        void* font, vector<string> text)
{
        vector<string>::iterator it;

        // Iterates throguh the text vector and prints it to the screen
        for (it = text.begin(); it != text.end(); it++)
        {
                glColor3d(r, g, b);
                glRasterPos2d(x, y);

                for (unsigned int i = 0; i < it->length(); i++)
                {
                        glutBitmapCharacter(font, (*it)[i]);
                }

                // Because glut does not print newlines
                y += LINE_OFFSET;
        }
}

vector<string> TextEngine::findText(string fileName, string tag)
{
        // The tags are listed between dollar signs
        string fullTag = '$' + tag + '$';

        string fullPath = TEXT_PATH + fileName;

        ifstream infile(fullPath);

        // Buffer to read in data
        string buff;
        // Array to store strings
        vector<string> data;

        // Find the string(s) to read in
        getline(infile, buff);
        while (infile && buff != fullTag)
        {
                getline(infile, buff);
        }

        // Store the string(s)
        getline(infile, buff);
```

```
        while (infile && buff != "$END$")
        {
                data.push_back(buff);
                getline(infile, buff);
        }

        infile.close();

        return data;
}

void TextEngine::openFile(double x, double y,
        double r, double g, double b,
        string fileName, string tag)
{
        vector<string> input = findText(fileName, tag);

        displayText(x, y, r, g, b,
                GLUT_BITMAP_HELVETICA_18,
                input);
}

vector<string> TextEngine::getText(string fileName, string tag)
{
        vector<string> input = findText(fileName, tag);

        return input;
}

void TextEngine::printString(double x, double y, double r, double g, double b,
        string text)
{
        glColor3d(r, g, b);
        glRasterPos2d(x, y);

        for (unsigned int i = 0; i < text.length(); i++)
        {
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);
        }

        // Vertical spacing
        y += LINE_OFFSET;
}
```

6.1.41 Triangle.h

```
/*************************************************************\
 * Triangle.h                                                *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the Triangle class  *
 * Which is used to hold the details of a 2D Triangle and    *
 * draw  it to the screen                                    *
\*************************************************************/

```

```
#ifndef TRIANGLE_H
#define TRIANGLE_H

class Triangle
{
private:
        // Arrays containing the colors and the xyz vertices of the triangles
        double color[4], vertices[9];
public:
        // Takes in the vertices and color of the triangle
        Triangle(const double(&new_vertices)[9], const double(&new_color)[4]);
        // Print the triangle in 3D
        void Display();
        // Print the triangle in 2D
        void Display2D();
};

#endif
```

6.1.42 Triangle.cpp

```
/*************************************************************\
 * Triangle.h                                                *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the definition of the triangle class   *
 * For more information, see Triangle.h                      *
\*************************************************************/

// Class declaration
#include "Triangle.h"

// For std::copy
#include <iterator>
#include <utility>

// OpenGL API
#include <GL\glut.h>

using namespace std;


Triangle::Triangle(const double(&new_vertices)[9], const double(&new_color)[4])
{
        // Copies the color entry
        copy(begin(new_color), end(new_color), color);

        // Copies the vertices
        copy(begin(new_vertices), end(new_vertices), vertices);
}

void Triangle::Display()
{
        // Sets OpenGL's color to the triangle's color
        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  }
```

6.1.43 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();
```

```
        // Attempts to trigger the target
        bool tryToTrigger(void* input, GCtype type);
        // Binds the triggering object
        void bindTrigger(void* _trigger);
        // Binds the target object
        void bindTarget(void* _target);
        // Constructor takes in trigger type and target type
        Trigger(GCtype _triggerType, GCtype _targetType);

};

#endif
```

6.1.44 Trigger.cpp

```
/*********************************************************\
 * Trigger.cpp                                            *
 * This file was created by Jeremy Greenburg              *
 * As part of The God Core game for the University of     *
 * Tennessee at Martin's University Scholars Organization *
 *                                                        *
 * This file contains the definition of the Trigger class *
 * For more information, see Trigger.h                    *
\*********************************************************/

#include <cstdlib>
#include "Trigger.h"

int Trigger::getTriggerType()
{
        return triggerType;
}

void Trigger::activateTarget()
{
        switch (targetType)
        {
                case T_TERMINAL:
                {
                        Terminal* t = (Terminal*)target;
                        t->activate();
                        break;
                }
                case T_SWITCH:
                {
                        Switch* s = (Switch*)target;
                        s->activate();
                        break;
                }
                default:
                {
                        break;
                }
        }
}

bool Trigger::tryToTrigger(void* input, GCtype type)
```

```
{
        // If this trigger is the correct type
        if (triggerType != type) return false;

        // If this trigger is the correct object
        if (trigger != input) return false;

        activateTarget();

        return true;
}

void Trigger::bindTrigger(void* _trigger)
{
        trigger = _trigger;
}

void Trigger::bindTarget(void* _target)
{
        target = _target;
}

Trigger::Trigger(GCtype _triggerType, GCtype _targetType)
{
        trigger = NULL;
        target = NULL;
        triggerType = _triggerType;
        targetType = _targetType;
}
```

6.1.45 Triple.h

```
/*************************************************************\
 * Triple.h                                                  *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the Triple class    *
 * Which is just a simple 3-tuple really                     *
\*************************************************************/

#ifndef TRIPLE_H
#define TRIPLE_H

class Triple
{
public:
        double a, b, c;
};

// For converting to a triple
Triple makeTrip(double _a, double _b, double _c);

#endif
```

6.1.46 Triple.cpp

```
/*************************************************************\
 * Triple.cpp                                                *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the definition of the TwoD class       *
 * For more information, see Triple.h                        *
\*************************************************************/

#include "Triple.h"

Triple makeTrip(double _a, double _b, double _c)
{
        Triple ret;
        ret.a = _a;
        ret.b = _b;
        ret.c = _c;

        return ret;
}
```

6.1.47 TwoD.h

```
/*************************************************************\
 * TwoD.h                                                    *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the declaration of the TwoD class      *
 * Which is used to hold the data and functionality for      *
 * Drawing in 2D with OpenGL                                 *
\*************************************************************/

#ifndef TWOD
#define TWOD

class TwoD
{
protected:
        // The pixel boundaries of the screen
        const double SCREENTOP = 0, SCREENBOTTOM = 1080,
                SCREENLEFT = 0, SCREENRIGHT = 1920;

        // Prepares OpenGL draw in 2D
        void prepare2D();

        // "Resets" OpenGL to draw in 3D
        void prepare3D();

};

#endif
```

6.1.48 TwoD.cpp

```
/*************************************************************\
 * TwoD.cpp                                                  *
 * This file was created by Jeremy Greenburg                 *
 * As part of The God Core game for the University of        *
 * Tennessee at Martin's University Scholars Organization    *
 *                                                           *
 * This file contains the definition of the TwoD class       *
 * For more information, see TwoD.h                          *
\*************************************************************/

#include "TwoD.h"

// OpenGL API
#include <gl\glut.h>

void TwoD::prepare2D()
{
        // Disable depth testing
        glDisable(GL_DEPTH_TEST);
        // Disable writing to the z buffer
        glDepthMask(GL_FALSE);
        // Disables lighting
        glDisable(GL_LIGHTING);

        // Create an orthogonal matrix to write on
        glMatrixMode(GL_PROJECTION);
        glPushMatrix();
        glLoadIdentity();
        glOrtho(SCREENLEFT, SCREENRIGHT, SCREENBOTTOM, SCREENTOP, -1, 1);
        glMatrixMode(GL_MODELVIEW);
        glPushMatrix();
        glLoadIdentity();
}

void TwoD::prepare3D()
{
        // Discards the orthogonal matrices
        glMatrixMode(GL_PROJECTION);
        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();

        // Enable depth testing
        glEnable(GL_DEPTH_TEST);
        // Enables writing to the z buffer
        glDepthMask(GL_TRUE);
        // Renable lighting
        glEnable(GL_LIGHTING);
}
```

6.2 Database

6.2.1 Walls

ID	LEVEL	X1	X2	X3	X4	Y1	Y2	Y3	Y4	Z1	Z2	Z3	Z4	R	G	B	A	Axis

ID	LEVEL	X1	X2	X3	X4	Y1	Y2	Y3	Y4	Z1	Z2	Z3	Z4	R	G	B	A	Axis
lvlceiling	LEVELZERO	-5	-5	8	8	1	1	1	1	-4	1	1	-4	0.70	0.70	0.70	1	0
lvlfloor	LEVELZERO	-5	-5	8	8	-1	-1	-1	-1	-4	1	1	-4	0.70	0.70	0.70	1	0
room0frntlftwall	LEVELZERO	5	5	5	5	-1	1	1	-1	-4	-4	-2.5	-2.5	0.29	0.29	0.29	1	z
room0frntrghtwall	LEVELZERO	5	5	5	5	-1	1	1	-1	-0.5	-0.5	1	1	0.29	0.29	0.29	1	z
room0backwall	LEVELZERO	-5	-5	-5	-5	-1	1	1	-1	-4	-4	1	1	0.29	0.29	0.29	1	z
room0rghtwall	LEVELZERO	-5	-5	5	5	-1	1	1	-1	1	1	1	1	0.29	0.29	0.29	1	x
room0frnttopwall	LEVELZERO	5	5	5	5	0.5	1	1	0.5	-2.5	-2.5	-0.5	-0.5	0.29	0.29	0.29	1	z
room1lftwall	LEVELZERO	5	5	8	8	-1	1	1	-1	-4	-4	-4	-4	0.29	0.29	0.29	1	x
room1rghtwall	LEVELZERO	5	5	8	8	-1	1	1	-1	1	1	1	1	0.29	0.29	0.29	1	x
room1frntbotwall	LEVELZERO	8	8	8	8	-1	-0.5	-0.5	-1	-4	-4	1	1	0.29	0.29	0.29	1	z
room1frnttopwall	LEVELZERO	8	8	8	8	0.5	1	1	0.5	-4	-4	1	1	0.29	0.29	0.29	1	z
room0lftlftwall	LEVELZERO	-5	-5	-1.5	-1.5	-1	1	1	-1	-4	-4	-4	-4	0.29	0.29	0.29	1	x
room0lftrghtwall	LEVELZERO	1.5	1.5	5	5	-1	1	1	-1	-4	-4	-4	-4	0.29	0.29	0.29	1	x
room0lfttopwall	LEVELZERO	-1.5	-1.5	1.5	1.5	0.70	1	1	0.70	-4	-4	-4	-4	0.29	0.29	0.29	1	x
room0frntmidwall	LEVELZERO	8	8	8	8	0.5	-0.5	-0.5	0.5	-4	-4	1	1	0.12	0.56	1	0.60	z
lvl1Floor	LEVELONE	30	30	-30	-30	-1	-1	-1	-1	40	-5	-5	40	0.70	0.70	0.70	1	0
lv1HangarCeiling	LEVELONE	15	15	-15	-15	5	5	5	5	5	-5	-5	5	0.70	0.70	0.70	1	0
lv1HangerFrntLeftWall	LEVELONE	15	15	1	1	-1	5	5	-1	5	5	5	5	0.80	0	0	1	x
lv1HangerFrntRghtWall	LEVELONE	-1	-1	-15	-15	-1	5	5	-1	5	5	5	5	0.80	0	0	1	x
lvl1HangerFrntTopWall	LEVELONE	1	1	-1	-1	1	5	5	1	5	5	5	5	0.80	0	0	1	x
lvl1HangerRghtWall	LEVELONE	-15	-15	-15	-15	-1	5	5	-1	5	5	-5	-5	0.80	0	0	1	z
lvl1HangerLeftWall	LEVELONE	15	15	15	15	-1	5	5	-1	5	5	-5	-5	0.80	0	0	1	z
lvl1HangerBckWall	LEVELONE	15	15	-15	-15	-1	5	5	-1	-5	-5	-5	-5	0.12	0.56	1	0.402	x
lvl1HallLftWall1	LEVELONE	3	3	3	3	-1	1	1	-1	5	5	11	11	0.80	0	0	1	z
lvl1HallRghtWall1	LEVELONE	-3	-3	-3	-3	-1	1	1	-1	5	5	8	8	0.80	0	0	1	z
lvl1HallLftWall2	LEVELONE	3	3	3	3	-1	1	1	-1	13	13	19	19	0.80	0	0	1	z
lvl1HallRghtWall2	LEVELONE	-3	-3	-3	-3	-1	1	1	-1	10	10	15	15	0.80	0	0	1	z
lvl1HallRghtWall3	LEVELONE	-3	-3	-3	-3	-1	1	1	-1	17	17	19	19	0.80	0	0	1	z
lvl1HallCeiling	LEVELONE	15	15	-15	-15	1	1	1	1	5	29	29	5	0.70	0.70	0.70	1	0
lvl1HallLftEnd	LEVELONE	7.5	7.5	1	1	-1	1	1	-1	19	19	19	19	0.80	0	0	1	x
lvl1HallRghtEnd	LEVELONE	-3	-3	-1	-1	-1	1	1	-1	19	19	19	19	0.80	0	0	1	x
lvl1LftRoom1LeftWall	LEVELONE	-3	-3	-15	-15	-1	1	1	-1	19	19	19	19	0.80	0	0	1	x
lvl1LftRoom1RghtWall	LEVELONE	-3	-3	-15	-15	-1	1	1	-1	12.5	12.5	12.5	12.5	0.80	0	0	1	x
lvl1LftRoomsBckWall	LEVELONE	-15	-15	-15	-15	-1	1	1	-1	19	19	5	5	0.80	0	0	1	z
lvl1Room4LeftWall	LEVELONE	7.5	7.5	7.5	7.5	-1	1	1	-1	19	19	29	29	0.80	0	0	1	z
lvl1Room4RghtWall	LEVELONE	-7.5	-7.5	-7.5	-7.5	-1	1	1	-1	19	19	29	29	0.80	0	0	1	z
lvl1Room3LeftWall	LEVELONE	20	20	20	20	-1	1	1	-1	10	10	-3	-3	0.80	0	0	1	z
lvl1Room3BckWall	LEVELONE	20	20	15	15	-1	1	1	-1	-3	-3	-3	-3	0.80	0	0	1	x
lvl1Room3FrntWall1	LEVELONE	20	20	6	6	-1	1	1	-1	10	10	10	10	0.80	0	0	1	x
lvl1Room3FrntWall2	LEVELONE	4	4	3	3	-1	1	1	-1	10	10	10	10	0.80	0	0	1	x
lvl1Room2LftWall	LEVELONE	7.5	7.5	7.5	7.5	-1	1	1	-1	19	19	10	10	0.80	0	0	1	z
lvl1Room3Ceiling	LEVELONE	20	20	15	15	1	1	1	1	-3	10	10	-3	0.70	0.70	0.70	1	0
lvl1Room4BckWallLft	LEVELONE	7.5	7.5	1	1	-1	1	1	-1	29	29	29	29	0.80	0	0	1	x
lvl1Room4BckWallRght	LEVELONE	-1	-1	-7.5	-7.5	-1	1	1	-1	29	29	29	29	0.80	0	0	1	x
lvl2Room0BckWallLft	LEVELTWO	5	5	1	1	-1	1	1	-1	2	2	2	2	0	0.80	0	1	x
lvl2Room0BckWallRght	LEVELTWO	-5	-5	-1	-1	-1	1	1	-1	2	2	2	2	0	0.80	0	1	x
lvl2Room0LftWallLft	LEVELTWO	-5	-5	-5	-5	-1	1	1	-1	2	2	1	1	0	0.80	0	1	z
lvl2Room0LftWallRght	LEVELTWO	-5	-5	-5	-5	-1	1	1	-1	-1	-1	-2	-2	0	0.80	0	1	z
Lvl2Room0FrtnWall	LEVELTWO	-5	-5	5	5	-1	1	1	-1	-2	-2	-2	-2	0	0.80	0	1	x
Lvl2Room0RghtwallLft	LEVELTWO	5	5	5	5	-1	1	1	-1	-2	-2	-1	-1	0	0.80	0	1	z
Lvl2Room0RghtWallRght	LEVELTWO	5	5	5	5	-1	1	1	-1	1	1	2	2	0	0.80	0	1	z
Lvl2Room0FakeDoor	LEVELTWO	-1	-1	1	1	-1	1	1	-1	2	2	2	2	0.29	0.29	0.29	1	x
Lvl2LftHallBckWall	LEVELTWO	-5	-5	-18	-18	-1	1	1	-1	1	1	1	1	0	0.80	0	1	x
lvl2RghtHallBckWall	LEVELTWO	5	5	18	18	1	-1	-1	1	1	1	1	1	0	0.80	0	1	x
Lvl2LftHallLftWall	LEVELTWO	-18	-18	-18	-18	1	-1	-1	1	1	1	-15	-15	0	0.80	0	1	z
lvl2RghtHallRghtWall	LEVELTWO	18	18	18	18	1	-1	-1	1	1	1	-15	-15	0	0.80	0	1	z
lvl2LftHallFrntWallRght	LEVELTWO	-5	-5	-9	-9	-0.5	0.5	0.5	-0.5	-1	-1	-1	-1	0	0.80	0	1	x
lvl2LftHallLftWall	LEVELTWO	-11	-11	-15	-15	-0.5	0.5	0.5	-0.5	-1	-1	-1	-1	0	0.80	0	1	x
lvl2LftHallTopStrip1	LEVELTWO	-5	-5	-15	-15	1	0.5	0.5	1	-1	-1	-1	-1	0	0.80	0	1	x
lvl2LftHallBotStrip1	LEVELTWO	-5	-5	-15	-15	-0.5	-1	-1	-0.5	-1	-1	-1	-1	0	0.80	0	1	x
lvl2RghtHallFrntWallLft	LEVELTWO	5	5	9	9	-0.5	0.5	0.5	-0.5	-1	-1	-1	-1	0	0.80	0	1	x
lvl2RghtHallFrntWallRght	LEVELTWO	11	11	15	15	-0.5	0.5	0.5	-0.5	-1	-1	-1	-1	0	0.80	0	1	x
lvl2RghtHallTopStrip1	LEVELTWO	5	5	15	15	0.5	1	1	0.5	-1	-1	-1	-1	0	0.80	0	1	x
lvl2RghtHallBotStrip1	LEVELTWO	5	5	15	15	-0.5	-1	-1	-0.5	-1	-1	-1	-1	0	0.80	0	1	x
lvl2LftRoomsRghtWall	LEVELTWO	-5	-5	-5	-5	1	-1	-1	1	-2	-2	-12	-12	0	0.80	0	1	z
lvl2RghtRoomsLftWall	LEVELTWO	5	5	5	5	1	-1	-1	1	-2	-2	-12	-12	0	0.80	0	1	z
lvl2LftHallBckWall2	LEVELTWO	-18	-18	-1	-1	1	-1	-1	1	-15	-15	-15	-15	0	0.80	0	1	x
lvl2RghtHallBckWall2	LEVELTWO	18	18	1	1	1	-1	-1	1	-15	-15	-15	-15	0	0.80	0	1	x
lvl2BckWall	LEVELTWO	-9	-9	9	9	1	-1	-1	1	-12	-12	-12	-12	0	0.80	0	1	x
lvl2LftDividerLft	LEVELTWO	-15	-15	-11	-11	1	-1	-1	1	-5.5	-5.5	-5.5	-5.5	0	0.80	0	1	x
lvl2LftDividerRght	LEVELTWO	-9	-9	-5	-5	1	-1	-1	1	-5.5	-5.5	-5.5	-5.5	0	0.80	0	1	x
lvl2LftDividerTop	LEVELTWO	-11	-11	-9	-9	0.5	1	1	0.5	-5.5	-5.5	-5.5	-5.5	0	0.80	0	1	x
lvl2LftDividerTBot	LEVELTWO	-11	-11	-9	-9	-0.5	-1	-1	-0.5	-5.5	-5.5	-5.5	-5.5	0	0.80	0	1	x
lvl2RghtDividerLft	LEVELTWO	15	15	11	11	1	-1	-1	1	-5.5	-5.5	-5.5	-5.5	0	0.80	0	1	x
lvl2RghtDividerRght	LEVELTWO	9	9	5	5	1	-1	-1	1	-5.5	-5.5	-5.5	-5.5	0	0.80	0	1	x
lvl2RghtDividerTop	LEVELTWO	11	11	9	9	0.5	1	1	0.5	-5.5	-5.5	-5.5	-5.5	0	0.80	0	1	x
lvl2RghtDividerTBot	LEVELTWO	11	11	9	9	-0.5	-1	-1	-0.5	-5.5	-5.5	-5.5	-5.5	0	0.80	0	1	x
lvl2Ceiling	LEVELTWO	-18	-18	18	18	1	1	1	1	2	-20	-20	2	0.70	0.70	0.70	1	0
lvl2Floor	LEVELTWO	-18	-18	18	18	-1	-1	-1	-1	2	-20	-20	2	0.70	0.70	0.70	1	0
lvl2LftInnerWallLft	LEVELTWO	-15	-15	-15	-15	-1	1	1	-1	-1	-1	-2	-2	0	0.80	0	1	z
lvl2LftInnerWallCtr	LEVELTWO	-15	-15	-15	-15	-1	1	1	-1	-4	-4	-8	-8	0	0.80	0	1	z
lvl2LftInnerWalLRght	LEVELTWO	-15	-15	-15	-15	-1	1	1	-1	-10	-10	-12	-12	0	0.80	0	1	z
lvl2RghtInnerWallLft	LEVELTWO	15	15	15	15	-1	1	1	-1	-1	-1	-2	-2	0	0.80	0	1	z
lvl2RghtInnerWallCtr	LEVELTWO	15	15	15	15	-1	1	1	-1	-4	-4	-8	-8	0	0.80	0	1	z
lvl2RghtInnerWalLRght	LEVELTWO	15	15	15	15	-1	1	1	-1	-10	-10	-12	-12	0	0.80	0	1	z
lvl2EndHallLft	LEVELTWO	-1	-1	-1	-1	-1	1	1	-1	-15	-15	-20	-20	0	0.80	0	1	z
lvl2EndHallRght	LEVELTWO	1	1	1	1	-1	1	1	-1	-15	-15	-20	-20	0	0.80	0	1	z
lvl3StrtWallLft	LEVELTHREE	-1	-1	-13	-13	-1	1	1	-1	1	1	1	1	0	0	0.80	1	x
lvl3StrtWallRght	LEVELTHREE	1	1	13	13	-1	1	1	-1	1	1	1	1	0	0	0.80	1	x
lvl3Room0RghtWall	LEVELTHREE	-8	-8	-8	-8	-1	1	1	-1	1	1	-1	-1	0	0	0.80	1	z
lvl3Room0/1RghtWall	LEVELTHREE	-8	-8	-8	-8	-1	1	1	-1	-3	-3	-7	-7	0	0	0.80	1	z
lvl3Room1/2RghtWall	LEVELTHREE	-8	-8	-8	-8	-1	1	1	-1	-9	-9	-13	-13	0	0	0.80	1	z
lvl3Room2/3RghtWall	LEVELTHREE	-8	-8	-8	-8	-1	1	1	-1	-15	-15	-19	-19	0	0	0.80	1	z
lvl3Room3RghtWall	LEVELTHREE	-8	-8	-8	-8	-1	1	1	-1	-21	-21	-23	-23	0	0	0.80	1	z
lvl3Room7LftWall	LEVELTHREE	8	8	8	8	-1	1	1	-1	1	1	-1	-1	0	0	0.80	1	z
lvl3Room7/6LftWall	LEVELTHREE	8	8	8	8	-1	1	1	-1	-3	-3	-7	-7	0	0	0.80	1	z
lvl3Room6/5LftWall	LEVELTHREE	8	8	8	8	-1	1	1	-1	-9	-9	-13	-13	0	0	0.80	1	z
lvl3Room5/4LftWall	LEVELTHREE	8	8	8	8	-1	1	1	-1	-15	-15	-19	-19	0	0	0.80	1	z
lvl3Room4LftWall	LEVELTHREE	8	8	8	8	-1	1	1	-1	-21	-21	-23	-23	0	0	0.80	1	z

ID	LEVEL	X1	X2	X3	X4	Y1	Y2	Y3	Y4	Z1	Z2	Z3	Z4	R	G	B	A	Axis
lvl3Ceiling	LEVELTHREE	-13	-13	13	13	1	1	1	1	1	-23	-23	1	0.70	0.70	0.70	1	0
lvl3Floor	LEVELTHREE	-13	-13	13	13	-1	-1	-1	-1	1	-23	-23	1	0.70	0.70	0.70	1	0
lvl3Room0/1Divid	LEVELTHREE	-13	-13	-8	-8	-1	1	1	-1	-5	-5	-5	-5	0	0	0.80	1	x
lvl3Room1/2Divid	LEVELTHREE	-13	-13	-8	-8	-1	1	1	-1	-11	-11	-11	-11	0	0	0.80	1	x
lvl3Room2/3Divid	LEVELTHREE	-13	-13	-8	-8	-1	1	1	-1	-17	-17	-17	-17	0	0	0.80	1	x
lvl3Room7/6Divid	LEVELTHREE	13	13	8	8	-1	1	1	-1	-5	-5	-5	-5	0	0	0.80	1	x
lvl3Room6/5Divid	LEVELTHREE	13	13	8	8	-1	1	1	-1	-11	-11	-11	-11	0	0	0.80	1	x
lvl3Room5/4Divid	LEVELTHREE	13	13	8	8	-1	1	1	-1	-17	-17	-17	-17	0	0	0.80	1	x
lvl3FrntWall	LEVELTHREE	-13	-13	13	13	-1	1	1	-1	-23	-23	-23	-23	0	0	0.80	1	x
lvl3InnerLftWall	LEVELTHREE	-3	-3	-3	-3	-1	1	1	-1	-2	-2	-20	-20	0	0	0.80	1	z
lvl3InnerRghtWall	LEVELTHREE	3	3	3	3	-1	1	1	-1	-2	-2	-20	-20	0	0	0.80	1	z
lvl3Room0LftWall	LEVELTHREE	-13	-13	-13	-13	-1	1	1	-1	1	1	-1	-1	0	0	0.80	1	z
lvl3Room0/1LftWall	LEVELTHREE	-13	-13	-13	-13	-1	1	1	-1	-3	-3	-7	-7	0	0	0.80	1	z
lvl3Room1/2LftWall	LEVELTHREE	-13	-13	-13	-13	-1	1	1	-1	-9	-9	-13	-13	0	0	0.80	1	z
lvl3Room2/3LftWall	LEVELTHREE	-13	-13	-13	-13	-1	1	1	-1	-15	-15	-19	-19	0	0	0.80	1	z
lvl3Room3LftWall	LEVELTHREE	-13	-13	-13	-13	-1	1	1	-1	-21	-21	-23	-23	0	0	0.80	1	z
lvl3Room7RghtWall	LEVELTHREE	13	13	13	13	-1	1	1	-1	1	1	-1	-1	0	0	0.80	1	z
lvl3Room7/6RghtWall	LEVELTHREE	13	13	13	13	-1	1	1	-1	-3	-3	-7	-7	0	0	0.80	1	z
lvl3Room6/5RghtWall	LEVELTHREE	13	13	13	13	-1	1	1	-1	-9	-9	-13	-13	0	0	0.80	1	z
lvl3Room5/4RghtWall	LEVELTHREE	13	13	13	13	-1	1	1	-1	-15	-15	-19	-19	0	0	0.80	1	z
lvl3Room4RghtWall	LEVELTHREE	13	13	13	13	-1	1	1	-1	-21	-21	-23	-23	0	0	0.80	1	z
lvl3Entrance	LEVELTHREE	-1	-1	1	1	-1	1	1	-1	1	1	1	1	0.29	0.29	0.29	1	x
lvl3Room0FakeDoor	LEVELTHREE	-13	-13	-13	-13	-1	-1	-1	-1	-1	-1	-3	-3	0.5	0.5	0.5	1	z
lvl3Room1FakeDoor	LEVELTHREE	-13	-13	-13	-13	-1	1	1	-1	-7	-7	-9	-9	0.5	0.5	0.5	1	z
lvl3Room2FakeDoor	LEVELTHREE	-13	-13	-13	-13	-1	1	1	-1	-13	-13	-15	-15	0.5	0.5	0.5	1	z
lvl3Room3FakeDoor	LEVELTHREE	-13	-13	-13	-13	-1	1	1	-1	-19	-19	-21	-21	0.5	0.5	0.5	1	z
lvl3Room7FakeDoor	LEVELTHREE	13	13	13	13	-1	1	1	-1	-1	-1	-3	-3	0.5	0.5	0.5	1	z
lvl3Room6FakeDoor	LEVELTHREE	13	13	13	13	-1	1	1	-1	-7	-7	-9	-9	0.5	0.5	0.5	1	z
lvl3Room5FakeDoor	LEVELTHREE	13	13	13	13	-1	1	1	-1	-13	-13	-15	-15	0.5	0.5	0.5	1	z
lvl3Room4FakeDoor	LEVELTHREE	13	13	13	13	-1	1	1	-1	-19	-19	-21	-21	0.5	0.5	0.5	1	z
lvl3Room8BckWall	LEVELTHREE	-3	-3	3	3	-1	1	1	-1	-12	-12	-12	-12	0	0	0.80	1	x
lvl3Room9BckWall	LEVELTHREE	-3	-3	3	3	-1	1	1	-1	-15	-15	-15	-15	0	0	0.80	1	x
lvl3Room8FrntWallLft	LEVELTHREE	-3	-3	-1	-1	-1	1	1	-1	-2	-2	-2	-2	0	0	0.80	1	x
lvl3Room8FrntWallRght	LEVELTHREE	3	3	1	1	-1	1	1	-1	-2	-2	-2	-2	0	0	0.80	1	x
lvl3Room9FrntWallLft	LEVELTHREE	-3	-3	-1	-1	-1	1	1	-1	-20	-20	-20	-20	0	0	0.80	1	x
lvl3Room9FrntWallRght	LEVELTHREE	3	3	1	1	-1	1	1	-1	-20	-20	-20	-20	0	0	0.80	1	x
lvl4Room0LftWall	LEVELFOUR	-5	-5	-5	-5	-1	1	1	-1	-2	-2	2	2	0.29	0.29	0.29	1	z
lvl4Room0RghtWall	LEVELFOUR	5	5	5	5	-1	1	1	-1	2	2	-2	-2	0.29	0.29	0.29	1	z
lvl4Room0BckLft	LEVELFOUR	-5	-5	-1	-1	-1	1	1	-1	2	2	2	2	0.29	0.29	0.29	1	x
lvl4Room0BckRght	LEVELFOUR	1	1	5	5	-1	1	1	-1	2	2	2	2	0.29	0.29	0.29	1	x
lvl4Room0Ceiling	LEVELFOUR	5	5	-5	-5	1	1	1	1	2	-2	-2	2	0.70	0.70	0.70	1	0
lvl4Room0Floor	LEVELFOUR	5	5	-5	-5	-1	-1	-1	-1	2	-2	-2	2	0.70	0.70	0.70	1	0
lvl4Entrance	LEVELFOUR	-1	-1	1	1	-1	1	1	-1	2	2	2	2	0.5	0.5	0.5	1	x
lvl4Room1LftWall	LEVELFOUR	10	10	10	10	-5	5	5	-5	-2	-2	-12	-12	0.60	0.60	0.60	1	z
lvl4Room1RghtWall	LEVELFOUR	-10	-10	-10	-10	-5	5	5	-5	-2	-2	-12	-12	0.60	0.60	0.60	1	z
lvl4Room1FrntWall	LEVELFOUR	10	10	-10	-10	-5	5	5	-5	-12	-12	-12	-12	0.60	0.60	0.60	1	x
lvl4Room1BckLftWall	LEVELFOUR	-10	-10	-5	-5	-5	5	5	-5	-2	-2	-2	-2	0.60	0.60	0.60	1	x
lvl4Room1BckRghtWall	LEVELFOUR	10	10	5	5	-5	5	5	-5	-2	-2	-2	-2	0.60	0.60	0.60	1	x
lvl4Room1Ceiling	LEVELFOUR	10	10	-10	-10	5	5	5	5	-2	-12	-12	-2	0.70	0.70	0.70	1	0
lvl4Room1Floor	LEVELFOUR	10	10	-10	-10	-5	-5	-5	-5	-2	-12	-12	-2	0.70	0.70	0.70	1	0
lvl4Room0FrntWall	LEVELFOUR	-5	-5	5	5	-1	1	1	-1	-2	-2	-2	-2	0.1	0.1	0.1	0.90	x
lvl2LftBckWindowLft	LEVELTWO	-15	-15	-11	-11	1	-1	-1	1	-12	-12	-12	-12	0	0.80	0	1	x
lvl2BckWindowTop	LEVELTWO	-11	-11	-9	-9	0.5	1	1	0.5	-12	-12	-12	-12	0	0.80	0	1	x
lvl2BckWindowBot	LEVELTWO	-11	-11	-9	-9	-0.5	-1	-1	-0.5	-12	-12	-12	-12	0	0.80	0	1	x
lvl2RghtBckWindowLft	LEVELTWO	15	15	11	11	1	-1	-1	1	-12	-12	-12	-12	0	0.80	0	1	x
lvl2RghtWindowTop	LEVELTWO	11	11	9	9	0.5	1	1	0.5	-12	-12	-12	-12	0	0.80	0	1	x
lvl2RghtWindowBot	LEVELTWO	11	11	9	9	-0.5	-1	-1	-0.5	-12	-12	-12	-12	0	0.80	0	1	x
lvl2lftWindow1	LEVELTWO	-11	-11	-9	-9	-0.5	0.5	0.5	-0.5	-1	-1	-1	-1	0	1	1	0.29	x
lvl2lftWindow2	LEVELTWO	-11	-11	-9	-9	-0.5	0.5	0.5	-0.5	-5.5	-5.5	-5.5	-5.5	0	1	1	0.29	x
lvl2lftWindow3	LEVELTWO	-11	-11	-9	-9	-0.5	0.5	0.5	-0.5	-12	-12	-12	-12	0	1	1	0.29	x
lvl2RghtWindow1	LEVELTWO	11	11	9	9	-0.5	0.5	0.5	-0.5	-1	-1	-1	-1	0	1	1	0.29	x
lvl2RghtWindow2	LEVELTWO	11	11	9	9	-0.5	0.5	0.5	-0.5	-5.5	-5.5	-5.5	-5.5	0	1	1	0.29	x
lvl2RghtWindow3	LEVELTWO	11	11	9	9	-0.5	0.5	0.5	-0.5	-12	-12	-12	-12	0	1	1	0.29	x

6.2.2 Doors

ID	LEVEL	X1	X2	X3	X4	Y1	Y2	Y3	Y4	Z1	Z2	Z3	Z4	R	G	B	A	axis
room0room1Door	LEVELZERO	5	5	5	5	-1	0.5	0.5	-1	-2.5	-2.5	-0.5	-0.5	0.90	0.90	0.90	1	z
lvl0ExitDoor	LEVELZERO	-1.5	-1.5	1.5	1.5	-1	0.70	0.70	-1	-4	-4	-4	-4	0.5	0.5	0.5	1	x
lvl1HangerHallDoor	LEVELONE	1	1	-1	-1	-1	1	1	-1	5	5	5	5	0.90	0.90	0.90	1	x
lvl1Room2Door	LEVELONE	3	3	3	3	-1	1	1	-1	11	11	13	13	0.90	0.90	0.90	1	z
lvl1Room0Door	LEVELONE	-3	-3	-3	-3	-1	1	1	-1	17	17	15	15	0.90	0.90	0.90	1	z
lvl1Room1Door	LEVELONE	-3	-3	-3	-3	-1	1	1	-1	10	10	8	8	0.90	0.90	0.90	1	z
lvl1Room3Door	LEVELONE	6	6	4	4	-1	1	1	-1	10	10	10	10	0.90	0.90	0.90	1	x
lvl1Room4Door	LEVELONE	1	1	-1	-1	-1	1	1	-1	19	19	19	19	0.90	0.90	0.90	1	x
lvl1ExitDoor	LEVELONE	1	1	-1	-1	1	-1	-1	1	29	29	29	29	0.5	0.5	0.5	1	x
lvl2Room0Door	LEVELTWO	-15	-15	-15	-15	-1	1	1	-1	-2	-2	-4	-4	0.90	0.90	0.90	1	z
lvl2Room1Door	LEVELTWO	-15	-15	-15	-15	-1	1	1	-1	-8	-8	-10	-10	0.90	0.90	0.90	1	z
lvl2Room2Door	LEVELTWO	15	15	15	15	-1	1	1	-1	-8	-8	-10	-10	0.90	0.90	0.90	1	z
lvl2Room3Door	LEVELTWO	15	15	15	15	-1	1	1	-1	-2	-2	-4	-4	0.90	0.90	0.90	1	z
lvl2ExitDoor	LEVELTWO	-1	-1	1	1	-1	1	1	-1	-20	-20	-20	-20	0.5	0.5	0.5	1	x
lvl3Room0Door	LEVELTHREE	-8	-8	-8	-8	-1	1	1	-1	-1	-1	-3	-3	0.13	0.5504	0.13	1	z
lvl3Room1Door	LEVELTHREE	-8	-8	-8	-8	-1	1	1	-1	-7	-7	-9	-9	0.13	0.5504	0.13	1	z
lvl3Room2Door	LEVELTHREE	-8	-8	-8	-8	-1	1	1	-1	-13	-13	-15	-15	0.70	0.13	0.13	1	z
lvl3Room3Door	LEVELTHREE	-8	-8	-8	-8	-1	1	1	-1	-19	-19	-21	-21	1	0.5504	0	1	z
lvl3Room7Door	LEVELTHREE	8	8	8	8	-1	1	1	-1	-1	-1	-3	-3	1	0.5504	0	1	z
lvl3Room6Door	LEVELTHREE	8	8	8	8	-1	1	1	-1	-7	-7	-9	-9	1	0.5504	0	1	z
lvl3Room5Door	LEVELTHREE	8	8	8	8	-1	1	1	-1	-13	-13	-15	-15	0.70	0.13	0.13	1	z
lvl3Room4Door	LEVELTHREE	8	8	8	8	-1	1	1	-1	-19	-19	-21	-21	1	0.5504	0	1	z
lvl3Room8Door	LEVELTHREE	-1	-1	1	1	-1	1	1	-1	-2	-2	-2	-2	0.90	0.90	0.90	1	x
lvl3Room9Door	LEVELTHREE	-1	-1	1	1	-1	1	1	-1	-20	-20	-20	-20	0.90	0.90	0.90	1	x

6.2.3 Switches

ID	LEVEL	target	xt	yt	zt	xr	yr	zr	type	startOn
lvl0Door1	LEVELZERO	room0room1Door	5	0	-3	0	90	0	DOOR	1
lvl0Door2	LEVELZERO	t˙lvlzero˙room1	7	0	-4	0	0	0	TERMINAL	1
lvl0END	LEVELZERO	NULL	-2	0	-4	0	0	0	LEVEL˙END	0
lvl1END	LEVELONE	NULL	1.5	0	29	0	0	0	LEVEL˙END	0
lv1Door1	LEVELONE	lvl1HangerHallDoor	1.5	0	5	0	0	0	DOOR	1
lvl1Door2	LEVELONE	lvl1Room2Door	3	0	10	0	90	0	DOOR	1
lvl1Door3	LEVELONE	lvl1Room0Door	-3	0	18	0	90	0	DOOR	1
lvl1Door4	LEVELONE	lvl1Room1Door	-3	0	11	0	90	0	DOOR	1
lvl1Door5	LEVELONE	lvl1Room4Door	1.5	0	19	0	0	0	DOOR	1
lvl1Door6	LEVELONE	lvl1Room3Door	6.5	0	10	0	0	0	DOOR	0
lvl2Door0	LEVELTWO	lvl2Room0Door	-15	0	-5	0	90	0	DOOR	1
lvl2Door1	LEVELTWO	lvl2Room1Door	-15	0	-7	0	90	0	DOOR	0
lvl2Door2	LEVELTWO	lvl2Room2Door	15	0	-7	0	90	0	DOOR	1
lvl2Door3	LEVELTWO	lvl2Room3Door	15	0	-5	0	90	0	DOOR	0
lvl2END	LEVELTWO	NULL	-1	0	-18	0	90	0	LEVEL˙END	0
lvl3Room0	LEVELTHREE	lvl3Room0Door	-8	0	-0.5	0	90	0	DOOR	1
lvl3Room1	LEVELTHREE	lvl3Room1Door	-8	0	-6.5	0	90	0	DOOR	1
lvl3Room2	LEVELTHREE	lvl3Room2Door	-8	0	-12.5	0	90	0	DOOR	0
lvl3Room3	LEVELTHREE	lvl3Room3Door	-8	0	-18.5	0	90	0	DOOR	0
lvl3Room7	LEVELTHREE	lvl3Room7Door	8	0	-0.5	0	90	0	DOOR	0
lvl3Room6	LEVELTHREE	lvl3Room6Door	8	0	-6.5	0	90	0	DOOR	0
lvl3Room5	LEVELTHREE	lvl3Room5Door	8	0	-12.5	0	90	0	DOOR	0
lvl3Room4	LEVELTHREE	lvl3Room4Door	8	0	-18.5	0	90	0	DOOR	0
lvl3Room0END	LEVELTHREE	NULL	-13	0	-0.5	0	90	0	LEVEL˙END	0
lvl3Room1END	LEVELTHREE	NULL	-13	0	-6.5	0	90	0	LEVEL˙END	0
lvl3Room2END	LEVELTHREE	NULL	-13	0	-12.5	0	90	0	LEVEL˙END	0
lvl3Room3END	LEVELTHREE	NULL	-13	0	-18.5	0	90	0	LEVEL˙END	0
lvl3Room7END	LEVELTHREE	NULL	13	0	-0.5	0	90	0	LEVEL˙END	0
lvl3Room6END	LEVELTHREE	NULL	13	0	-6.5	0	90	0	LEVEL˙END	0
lvl3Room5END	LEVELTHREE	NULL	13	0	-12.5	0	90	0	LEVEL˙END	0
lvl3Room4END	LEVELTHREE	NULL	13	0	-18.5	0	90	0	LEVEL˙END	0
lvl3Room8	LEVELTHREE	lvl3Room8Door	-1.5	0	-2	0	0	0	DOOR	0
lvl3Room9	LEVELTHREE	lvl3Room9Door	1.5	0	-20	0	0	0	DOOR	1

6.2.4 Terminals

ID	LEVEL	tag	xt	yt	zt	xr	yr	zr
t˙lvlzero˙room1	LEVELZERO	lv0TM1.tm	7	0	-2	0	0	0
t˙lvl1Room0	LEVELONE	lvl1TM0.tm	-13	0	15	0	180	0
t˙lvl1Room1	LEVELONE	lvl1TM1.tm	-13	0	8	0	180	0
t˙lvl1Room3	LEVELONE	lvl1TM2.tm	17	0	-2	0	90	0
t˙lvl2Room0	LEVELTWO	lvl2TM0.tm	-6	0	-3	0	0	0
t˙lvl2Room1	LEVELTWO	lvl2TM1.tm	-6	0	-9	0	0	0
t˙lvl2Room2	LEVELTWO	lvl2TM2.tm	6	0	-9	0	180	0
t˙lvl2Room3	LEVELTWO	lvl2TM3.tm	6	0	-3	0	180	0
t˙lvl3Room0	LEVELTHREE	lvl3TM0.tm	-10	0	-4	180	-90	0
t˙lvl3Room1	LEVELTHREE	lvl3TM1.tm	-10	0	-10	180	-90	0
t˙lvl3Room2	LEVELTHREE	lvl3TM2.tm	-10	0	-16	180	-90	0
t˙lvl3Room3	LEVELTHREE	lvl3TM3.tm	-10	0	-22	180	-90	0
t˙lvl3Room4	LEVELTHREE	lvl3TM4.tm	10	0	-22	180	-90	0
t˙lvl3Room5	LEVELTHREE	lvl3TM5.tm	10	0	-16	180	-90	0
t˙lvl3Room6	LEVELTHREE	lvl3TM6.tm	10	0	-10	180	-90	0
t˙lvl3Room7	LEVELTHREE	lvl3TM7.tm	10	0	-4	180	-90	0
t˙lvl3Room8	LEVELTHREE	lvl3TM8.tm	0	0	-11	0	90	0
t˙lvl3Room9	LEVELTHREE	lvl3TM9.tm	0	0	-16	0	-90	0

6.2.5 Triggers

ID	LEVEL	Trigger	Target	TriggerType	TargetType
tr˙lvl0End	LEVELZERO	t˙lvlzero˙room1	lvl0END	TERMINAL	SWITCH
tr˙lvl1Switch	LEVELONE	t˙lvl1Room0	lvl1Door6	TERMINAL	SWITCH
tr˙lvl1End	LEVELONE	t˙lvl1Room3	lvl1END	TERMINAL	SWITCH
tr˙lvl2Door1	LEVELTWO	t˙lvl2Room0	lvl2Door3	TERMINAL	SWITCH
tr˙lvl2Door2	LEVELTWO	t˙lvl2Room3	lvl2Door1	TERMINAL	SWITCH
tr˙lvl2End	LEVELTWO	t˙lvl2Room1	lvl2END	TERMINAL	SWITCH
tr˙lvl3Sec1	LEVELTHREE	t˙lvl3Room9	lvl3Room3	TERMINAL	SWITCH
tr˙lvl3Sec2	LEVELTHREE	t˙lvl3Room9	lvl3Room4	TERMINAL	SWITCH
tr˙lvl3Sec3	LEVELTHREE	t˙lvl3Room9	lvl3Room6	TERMINAL	SWITCH
tr˙lvl3Sec4	LEVELTHREE	t˙lvl3Room9	lvl3Room8	TERMINAL	SWITCH
tr˙lvl3Sec5	LEVELTHREE	t˙lvl3Room8	lvl3Room5	TERMINAL	SWITCH
tr˙lvl3Sec6	LEVELTHREE	t˙lvl3Room8	lvl3Room2	TERMINAL	SWITCH
tr˙lvl3End	LEVELTHREE	t˙lvl3Room2	lvl3Room2END	TERMINAL	SWITCH
tr˙lvl3Sec7	LEVELTHREE	t˙lvl3Room8	lvl3Room7	TERMINAL	SWITCH

6.2.6 Cylinders

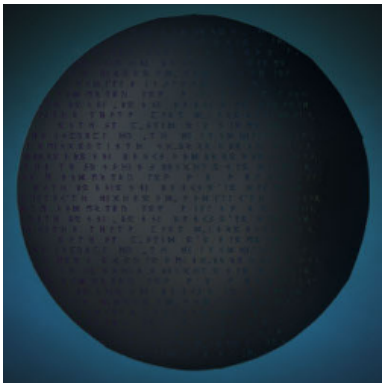
ID	XT	YT	ZT	XR	YR	ZR	base Radius	top Radius	height	stacks	slices	R	G	B	A	LEVEL
lifePodCenter	-4	1	-1.5	90	0	0	0.5	0.5	2	50	50	0	0.80	0.8195	1	LEVELZERO
lifePodBot	-4	0	-1.5	90	0	0	0.0503	1	1	50	50	0.5	0.5	0.5	1	LEVELZERO
lifePodTop	-4	1	-1.5	90	0	0	1	0.5	0.5	50	50	0.5	0.5	0.5	1	LEVELZERO
kiosk1	5	1	25	90	0	0	0.5	0.5	2	50	50	0	0.75	1	0.598	LEVELONE
kiosk2	-5	1	25	90	0	0	0.5	0.5	2	50	50	0	0.75	1	0.598	LEVELONE
power1	-2	1	-5	90	0	0	0.5	0.5	2	50	50	1	0.8397	0	1	LEVELTHREE
power2	2	1	-5	90	0	0	0.5	0.5	2	50	50	1	0.8397	0	1	LEVELTHREE
power3	-2	1	-18	90	0	0	0.5	0.5	2	50	50	1	0.8397	0	1	LEVELTHREE
power4	2	1	-18	90	0	0	0.5	0.5	2	50	50	1	0.8397	0	1	LEVELTHREE
power1BaseBot	-2	0	-5	90	0	0	0.70	0.70	1	50	50	0.201	0.201	0.201	1	LEVELTHREE
power1BaseTop	-2	1	-5	90	0	0	0.70	0.70	0.5	50	50	0.201	0.201	0.201	1	LEVELTHREE
power2BaseBot	2	0	-5	90	0	0	0.70	0.70	1	50	50	0.201	0.201	0.201	1	LEVELTHREE
power2BaseTop	2	1	-5	90	0	0	0.70	0.70	0.5	50	50	0.201	0.201	0.201	1	LEVELTHREE
power3BaseBot	-2	0	-18	90	0	0	0.70	0.70	1	50	50	0.201	0.201	0.201	1	LEVELTHREE
power3BaseTop	-2	1	-18	90	0	0	0.70	0.70	0.5	50	50	0.201	0.201	0.201	1	LEVELTHREE
power4BaseBot	2	0	-18	90	0	0	0.70	0.70	1	50	50	0.201	0.201	0.201	1	LEVELTHREE
power4BaseTop	2	1	-18	90	0	0	0.70	0.70	0.5	50	50	0.201	0.201	0.201	1	LEVELTHREE

6.3 Images

6.3.1 Main Menu

6.3.2 Terminal Banner

6.3.3 Game Icon



6.4 Music

1. Dark Fog—Kevin MacLeod
2. Mismer—Devin Powers
3. One Sly Move—Kevin MacLeod
4. Hypnothis—Kevin MacLeod
5. Cold Hope—Arseniy Shkljaev
6. Spacial Harvest—Kevin MacLeod

References

[1] Firelight Technologies *FMOD Studio API*
http://www.fmod.org/documentation/

[2] Jonathan Dummer *Simple OpenGL Image Library* July 7, 2008
http://www.lonesock.net/soil.html

[3] Khronos Group *OpenGL API Documentation Overview*
https://www.opengl.org/documentation/

[4] Maplesoft *Equation of a Plane - 3 Points*

[5] Microsoft *SHGetFolderPath function*
https://msdn.microsoft.com/en-us/library/windows/desktop/bb762181(v=vs.85).aspx

[6] Robert Deyoso *Personal Interview* February, 2015

[7] SQLite Consortium *An Introduction To The SQLite C/C++ Interface*
https://www.sqlite.org/cintro.html