Game Project Report

The Cat Dungeon

Submitted by

Team Members	**Student Number**
Lily Chong	69331049
Thomson Lai	64922040
Phoebe Ramsay	69452043
Kenneth Wong	61694048
Effiam Yung	62912043

EECE 478 Computer Graphics
The University of British Columbia
April 10, 2008

Abstract

The goal of our game project is to design a 3-D graphical computer game using OpenGL. For our project, we decided to design a 3-D maze game where the objective of the game is to find a way to escape out of the maze. The user, played as a mouse, has to find a way to get out of the cat's dungeon without being caught. The game is designed in a Windows environment and written in C/C++ OpenGL with the use of OpenAL. For our project, we have implemented several GLUT programming functions as taught in EECE 478. As a result, we have created a 3-D maze game that is fun and enjoyable.

Table of Contents

1.0 Introduction

This report describes the process involved in making a 3-D maze graphical computer game with OpenGL. Using OpenAL together with OpenGL, we created a 3-D game where the objective for the user is to escape from traps and rivalries to find their way out of a maze. This report discusses the game overview, including the description and game play, and then focuses on the game design, describing how the game is implemented and the programming functions and libraries used in the design.

The individual project logs of each of the members are also included as an appendix in this report.

2.0 Game Overview

The primary objective of the game is for the user to search for a way out of a 3-D cat dungeon. Throughout the maze, there are various items placed on different locations where the user can pick up and use to aid their way out of the maze. However, there are also several obstacles that the user must avoid in order to exit the maze successfully. This section elaborates on the description of the game play and user controls.

2.1 Game Description

In the game, the player is played as a mouse that has to find its exit through a cat dungeon within a specified time. The dungeon is designed as a two-storey high maze. To escape from the maze, the player must avoid getting caught by cats and falling into traps located in different areas in the maze. Throughout the maze, the player can find items, which includes a cheese that can speed up their walk and torch, to help them to get out of the maze as soon as possible. Additionally, the player can advance to a different storey in the maze via a teleport pad.

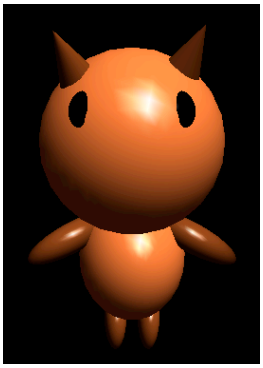
2.1.1 Characters

The Mouse

The major character of the game that represents the player. The Mouse would not be visible when player is playing the game because The Cat Dungeon uses a first person view. However, the Mouse is animated on the

game start screen, the victory screen and the game over screen.

The Cat



The villain of the game. There are multiple cats in the maze and meow as they walk. Player should avoid the cat if they hear the meow sound.

2.1.2 Items

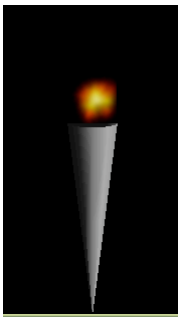
Cheese

Speed increase item. Increase the Mouse's moving speed once taken.

Mouse Trap

Speed decrease item. Decrease the Mouse's moving speed once taken.

Torch



Light increase item. Brighten the surrounding for a better visibility.

2.1.3 Environment

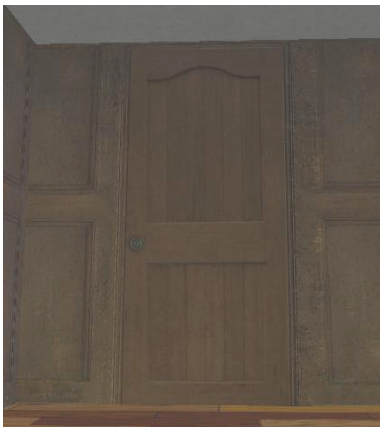
Telepad

Teleport the mouse to the storey above or below.

Entrance

Maze entrance. Game starts once going through the door.

Exit



Maze exit. Player clears game level by going through the exit.

2.2 User Manual

The player controls the game play using particular keys on the keyboard. The different movements of the mouse and camera controls available for the player are listed below.

Direction Keys:

- Key ‘w’: Move the mouse forward
- Key ‘s’: Move the mouse backward
- Key ‘a’: Turn left
- Key ‘d’: Turn right

Other:

- Key ‘ESC’: Terminate the game
- Mouse: Changes the view angle of the scene

3.0 Game Design

In creating the game, many different GLUT programming techniques are used for drawing and viewing objects and adding special effects. This section describes the methods of how the game is implemented.

3.1 3-D Viewing and Objects

The mouse, cats and other items in the game are drawn with different shapes and lines using various GLUT programming functions, including functions that create 3-D shapes and triangles such as glSolidSphere, glSolidCone and GL_TRIANGLES. These basic shapes are then altered into a desired form using transformation functions including glRotate and glTranslate. For example, the following code shows how the cat's body can be drawn with basic shapes and transformation.

```
glPushMatrix();
    glColor3f(1.0000,   0.4900,   0.2500);
    glScaled(0.80, 1.0, 0.80);
    glTranslated(0.0,-5.50,0.0);
    glutSolidSphere(3.0, 25, 25);
glPopMatrix();
```

3.2 Camera

The camera in the game is defined with the gluLookAt function. Throughout the main part of the game the player explores the maze in the first-person-view. He is able to look around the maze from his current position by using the mouse as well. This was done with glutPassiveMotionFunc which allows the game to trace where the mouse is located inside the game window.

3.3 Lighting and Material Variation

Whenever the game is displayed in first person view, light sources, namely, specular, diffuse and ambient, are set. The light sources settings are initialized with the function glLightfv. These light sources are created with the code below.

```
light_pos[0]=position[1];
light_pos[1]=position[2]+MOUSEHEIGHT;
light_pos[2]=position[0];

glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_DIFFUSE,  light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_AMBIENT,  light_ambient);
```

The position of the light is defined by the current position of the player. The diffuse light is used to define the bright areas of the object and the ambient light defines the color of the object.

3.4 Texture Mapping

In designing the game, we used the technique of texture mapping in order to provide a professional touch to our graphics. Texture mapping is the process of coloring pixels based on an image. In our game, we used 24-bit bitmaps as our textures and mapped them to different objects in the game.

To use texture mapping, we first need to locate the computer directory of where the desired bitmap image file is stored and load the bitmap into our programming code. This process was done by using an available code from [ref1]. The data of the bitmap file is then read and created into a texture that is ready to map to our object. Using the GLUT function call glTexCoord2f, the texture is mapped to the surface of the object. The figure below illustrates one of the texture mapped objects in the game.

Figure: Textured Maze Wall and Ground

3.5 Design of the Maze

The maze is the environment in which the objects in our game interact. It is essentially the world of our game. To add to the replayability of our game, a random maze generator was implemented for the generation of the dungeon.

3.5.1 Maze Structure

The physical structure of the maze is a 3-dimensional rectangular grid of cells. It consists of a number of floors and each floor is made up of a number of rows and columns. Each cell consists of 4 walls which can either be a permanent wall, interior wall, or removed. Permanent walls are walls that will not be removed by the maze generator. These walls surround the outer edge of the maze and around the teleports between floors. Visually, there is no difference

between a permanent wall and an interior wall; it is only there to keep boundaries on the maze generator. Walls that are removed represent the path that the mouse can travel to get from cell to cell. The complete maze is stored as a 3-dimensional array of these cells.

3.5.2 Random Maze Generator

Two different algorithms were explored in the development of the maze generator. The first algorithm that was attempted was based on the depth-first search tree traversal algorithm. This method tended to create a simple, long, winding path with very short side paths. The second algorithm was based on Prim's algorithm for building a tree from a weighted graph. Prim's algorithm created more complex mazes than the depth-first search algorithm and so we decided use Prim's algorithm as the main maze generation algorithm.

The maze generation begins by selecting an entrance to the maze. The entrance is selected as a random cell in the first row of the first floor the maze. Next, a location for the connection between each floor and the floor above it is selected for every floor except the last one. There exists one of these for each floor and it can be located anywhere that is not on the outside edge of the maze. Following that, the actual maze is generated using the maze generator. Lastly, we select an exit point for the maze. This is located in the last row of the last floor of the maze.

The depth-first search algorithm was implemented as a recursive function. First, the current cell is marked as visited. Then, the adjacent cells are checked to see which cells have not yet been visited. If all cells have been visited then the function exits; otherwise, the function is called on a random neighbouring cell that has not been visited and the wall between the two cells is

removed.

In Prim's algorithm, vertices from a graph are added to a tree one by one based on the weight of each link. For our Prim's algorithm implementation, we use unweighted links so cells are selected randomly. At the beginning, we have one connected cell in our maze. For each iteration of the function's loop, a random cell that still has unvisited neighbouring cells from our group of connected cells is selected and the wall between that cell and the neighbour cell is removed. The neighbour cell is now considered to be connected to the maze. This loop continues until no more cells have unvisited neighbour cells.

3.6 Collision Detection

Since there are various objects located in the maze, the implementation of collision detection is required to detect when the player hits the wall of the maze or other items in the maze. There are two types of collision detection that we are concerned with in the game: collision between two objects and collision between the mouse and a wall. For the first type, each object is bounded by a rectangular box. A collision is detected when the volumes of the two objects' boxes intercept one another. The condition for a collision is: for each dimension (i.e. x,y,z), there is an overlap between the space occupied by object 1 and object 2 in that dimension. The second type of collision, between the mouse and a wall, is slightly different due to the fact that the maze is built up by numerous walls. To simplify this check, we isolate the mouse to a single cell. Each cell can only have up to four walls and so only collisions these four walls will need to be checks. A collision occurs when the mouse's volume extends across the border of the cell and the wall on the corresponding side of the cell has not been removed.

3.7 Sound Effects

OpenAL was used for the sound playback during the game play. The sound clips are in wav format which are loaded and stored in an array for easy access. Three of the sounds are used as background music for the main menu, game play and ending scene. The rest are sound effects trigger when certain criteria are met. Some criteria include: obtaining items, moving near the cats, and traveling to different floors. In addition, the sound effects play differently with respect to the distance and position of the objects with the player.

3.8 AI of the Cats

We attempted to keep the AI of the cats as simple as possible while still being effective. The cats only travel along the lines from the center of a cell to the center of the next cell. This is to remove the need to check for cat collisions with the maze walls. The cat checks for which direction to move whenever it reaches the center of its destination cell. When it enters a cell, it will randomly decide on a direction to continue moving based on what directions are available. The cat will only turn around if it reaches a dead end. The cat's memory is limited to one cell in the past and so it may randomly repeat some paths numerous times before moving on.

In addition to its normal maze roaming state, the cat will attempt to chase down a mouse that is in its direct line of sight. The conditions for this are that the cat must be on the same floor as the mouse and the cat shares a row or column with the mouse with no walls in between. In this state, the cat will attempt to move to the cell that the mouse is currently in.

4.0 Conclusion

In the game The Cat Dungeon, we have implemented a game environment that includes 3D viewing and objects with a moving camera, lighting and material variations, and texture mapping. To enhance the game environment we have also implemented advanced features such as collision detection, AI and sound effects using different OpenGL libraries available. The key feature of our randomly generated maze makes our game original and the variety of game features such as help and hindering items and background music makes the game more challenging but yet interesting and enjoyable to play with.

APPENDIX

APPENDIX A: Individual Reports

Lily Chong
69331049

Individual Project Log

My first task of the project was to implement the module of the cat. I used the glut primitives to design the cats. The module was broken down into different body parts and the parts are drawn separately using glutSolidSphere and glutSolidCone and other primitives. Details and proper proportion were given by using glScaled and glRotated. Then I give the module a walking animation by rotating its joints and tail at a limited angle with the using of a moving direction flag.

After implementing the cat module, I was responsible to implement the Game Starting screen, the Victory screen and the Game over screen animations. These scenes each display a loop animation that involves the two main characters, the mouse and the cat. For example, the game starting screen would show a mouse being chased by a cat around whereas the game over screen shows a dead mouse with its soul leaving the corpse and the victory screen is a mouse that moves around happily. Background music are also embedding in these three scenes by using the openAl library.

These three scenes are coded as independent functions and can be called by the main code. For example, when the mouse is caught by the cat, the death() is called and this function first clear the display window and display the game over scene and show the mouse dying animation and the game over title.

APPENDIX A: Individual Reports

Thomson Lai
64922040

Individual Project Log

Initially, I implemented a 2D version of the maze that always guarantees a path from the entrance to the exit. This version, however, contains many open spaces which sometimes lead to a straight path. Ken later on improved the maze and made it into multiple floors. The maze was then made into 3D textures were applied to the walls to create an indoor looking environment. These textures were taken from here:
http://www.garagegames.com/mg/snapshot/view.php?qid=1517

I also implemented the initial item system and collision detection for the game. Each of the items holds the value of its position, type and radius. The position and type were randomly generated so that they are scatter around the maze. The radius was set according to the type of item and was used for collision detection. At first all of the objects in the game were bounded by a sphere. Collision occurs when the player and the objects' distance is smaller or equal to the sum of their radius.

I applied OpenAL into the game so that the game will have sound playback. I visited various site and used some of the wav files in the game (see list below). The sound files are loaded into the game sound buffer for easy access. Two public function calls were created to allow simple playback. The functions allow the sound to be played with variation depending on the distance between the player and source.

http://blue-bomber.jvmwriter.org/index.php?title=Music_Archive
http://www.talkingwav.com/nintendo_wav_sounds.html
http://www.kessels.com/CatSounds/

APPENDIX A: Individual Reports

Phoebe Ramsay
69452043

Individual Project Log

In the project, my primary task was to implement the graphical items. At first I was involved in drawing the graphical objects with simple GLUT lines and shape primitives. Later on, I was involved in designing a fire torch. The base of the torch was drawn with the glutSolidCone function with the use of different material settings. Next, to give a realistic look to the fire, a particle generation system was used. The particle system is implemented based on a simple particle system found in an online OpenGL tutorial. Using the particle system, the torch fire is simulated to give a real-time fire effect by creating new particles starting with a single fire particle texture that is generated at a different random position and velocity in different times.

http://www.codeproject.com/KB/openGL/ParticleEngine.aspx

http://www.gameprogrammer.org/main.php?page=tutorial

http://www.gamedev.net/reference/list.asp?categoryid=72

APPENDIX A: Individual Reports

Kenneth Wong
61694048

Individual Project Log

My first task in this project was to randomly generate a maze. At first, I was just assisting Thomson in debugging his maze, but eventually I went on to find and implement a brand new algorithm. A few different algorithms were tested and these methods were compared based on average distance to the exit and general appearance of the maze. In the end, I decided to stick with the Prim's algorithm for generating maze. The information I used for learning how to build mazes was found at the website: http://www.mazeworks.com/mazegen/mazetut/index.htm.

After the maze was completed, Lily and Effiam sent me their cat and mouse models and I worked on integrating them into the maze. First, they had to be resized to fit into the maze, which was done by testing different glScalef values. Then, they were just randomly placed somewhere in the maze.

Next, I began to work on the cat's movement from cell to cell and subsequently, the cat's AI. For the cat AI, we did not want the cat to know the exact whereabouts of the mouse or the exit so we set the cats to randomly roam the maze. The algorithm for this was simple. A cell has a total of four walls and the cat enters the cell from one of the walls; if one of the three remaining ones is opened then the cat will randomly select an open wall and move to that cell, otherwise the cat turns around and returns to its previous cell. Due to the fact that the cat only keeps track of the previous cell it traveled in, it is possible that the cat will only roam a small portion of the maze.

At this point, we now had a completed maze and randomly roaming cats. The camera, however, was set in place in the sky giving a bird's eye view of the maze. The next step was to set up the moving camera in the inside of the maze. The camera's location was set to the location of the mouse giving a first-person view. The direction of the camera's view was divided into two angles, the horizontal and vertical angles. The vector for the camera's angle is given by:

[sin(h_ang), tan(v_ang), cos(h_ang)]

where the y axis is vertical axis. The camera was then set in place with the gluLookAt function.

The next thing that I worked on was detecting collisions. Now that the mouse was able to move, it was necessary to prevent it from moving through walls. The method used to check this was to determine which cell the mouse was in before he moves. After his movement, we check to see that the mouse has not crossed any walls that have not been removed. If the mouse has crossed a wall then it is moved back to its previous position.

Next, I worked on placing the items into the maze and implementing the powerups for when the player picks up the item. I made a few more changes to Thomson's collision detection.

Finally, all the different scenes of the game (starting, game, death, victory) were put together.

APPENDIX A: Individual Reports

Effiam Yung
62912043

Individual Project Log

My work was mostly related to the implementation of the graphical objects. The mouse, teleportation device, and the various items were drawn using glut primitives such as glutSolidSphere and others (lines, circles…etc.). Initially the objects were mapped to textures using the glaux library, however since the library has become outdated and obsolete we prompted for a better method. Obviously some objects require additional functions to be implemented in order for them to look the way I had envisioned. One such example is the mouse trap; I used quadrics (quadric = gluNewQuadric();) in order to be able to draw cylinders. Also I used alpha blending so the teleportation device will have a translucent effect to it.

In addition to the graphics, I implemented the initial first person camera view and mouse controls system. Basically the controls work by taking the difference in the x and y positions of the moving mouse, and then translating the camera according to it. When the mouse hits the edge of the window, glutWarpPointer(width /2, height / 2) is called to reposition the mouse back to the center of the screen.

I helped a bit with the opening scene by applying a fog effect to it and adding a ground to it. Also I helped with the death scene by drawing and animation the angelic object that rises from the corpse of the mouse when it gets caught.