Utsav Pandey  
Final Project  
Computer Graphics  
December 8, 2008

**3D Maze**

**Gameplay and Objective**

This is a 3D Maze game that gives the user the ability to start from a NXN maze. The objective of this game is to collect a certain number of spheres in a given amount of time to proceed to the next level. There are a total of 18 Levels. If the user fails to collect the required number of spheres in the allotted time, the level simply restarts. The user can try as many times as he/she wants.

The game uses texture mapping to create a dungeon-like environment. The total time spent in a level is made to represent an entire day. Hence, there is a sun, which is associated to the countdown timer. The sun rises as the countdown timer begins and sets as the counter ends.

**Startup Options**

The user will chose a NXN maze to begin the game from a menu as soon as the program starts. The screen will also have a ‘Navigation On/Off’ option, which will either turn on or turn off a small navigation map on the bottom left corner of the playing screen. The map will have a green arrow representing the user and white dots representing the spheres to be collected.

**Navigation**

The user will use the keyboard (arrow keys Up, Down, Left and Right) to navigate. The game does not have complex movements, only simple movements made smoother using timer function and linear interpolation. Hence, user can only move backward, forward or turn 90 degrees in either direction. For instance, pressing the Up and Left keys simultaneously will not make the user move forward at 45 degrees as that would be a complex movement.

The user can also use the Page Up and Page Down keys along with Left and Right keys to get and overview of the maze.

**On-screen Display**

The user will have a number of displays on-screen. Like mentioned earlier, if the navigation map is turned on, it will be displayed at the bottom left corner of the screen. The top left corner will have the score tally i.e. the number of spheres collected and the number of spheres left to be collected. The top center of the screen will have the countdown timer, which is an example of OpenGL animation. While in the game, the user can also right click at anytime to display a menu, which has the Restart and the Exit options.

**Functions**

1. **A 3D perspective view**

The 3D view has been created using the gluPerspective and gluLookAt functions utilizing the GL\_PROJECTION and GL\_MODELVIEW matrices.

---------------------------------------------------------------------------------------------------------------------------------

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(120, 1, 0.25, 100);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

.

.

.

//Linearly interpolate eye coordinates from current cell to next cell during viewer movement

float eyex = int\_pol(progress, current\_x, next\_x);

float eyey = current\_y;

float eyez = int\_pol(progress, current\_z, next\_z);

//Linearly interpolate view vector from current cell to next cell during viewer rotation

float viewx = int\_pol(progress, dx[current\_dir], dx[next\_dir]);

float viewy = 0;

float viewz = int\_pol(progress, dz[current\_dir], dz[next\_dir]);

.

.

.

//gluLookAt for inside maze

gluLookAt(eyex, eyey, eyez, eyex + viewx, eyey + viewy, eyez + viewz, 0, 1, 0);

1. **User interaction involving both the keyboard and the mouse.**

//Keyboard Function

void keyboardfunc(unsigned char key, int x, int y)

{

if(key == 27)

exit(0);

glutPostRedisplay();

}

//Special Keyboard function for handling keyboard arrow keys

void specialfunc(int key, int x, int y)

{

//Prevent keyboard handling during movement/rotation animation

if(moving\_state) return;

//Rotate the viewer 90 degrees to the left

if(key == GLUT\_KEY\_LEFT) {

if(current\_dir == ZPOS) next\_dir = XPOS;

if(current\_dir == XNEG) next\_dir = ZPOS;

if(current\_dir == ZNEG) next\_dir = XNEG;

if(current\_dir == XPOS) next\_dir = ZNEG;

start\_movement();

}

//Rotates the viewer 90 degrees to the right

if(key == GLUT\_KEY\_RIGHT) {

if(current\_dir == ZPOS) next\_dir = XNEG;

if(current\_dir == XNEG) next\_dir = ZNEG;

if(current\_dir == ZNEG) next\_dir = XPOS;

if(current\_dir == XPOS) next\_dir = ZPOS;

start\_movement();

}

if(key == GLUT\_KEY\_UP || key == GLUT\_KEY\_DOWN) {

//Current position in the maze

int x0 = current\_x;

int z0 = current\_z;

//Forward/backward direction of movement

int moving\_dir = current\_dir;

if(key == GLUT\_KEY\_DOWN)

moving\_dir = opposite[current\_dir];

//Position to move to

int x1 = current\_x + dx[moving\_dir];

int z1 = current\_z + dz[moving\_dir];

//Flag indicating if a movement is allowed

int allowed = 1;

//Check if the viewer is inside the maze

if(x0 >= 0 && x0 < maze\_xsize && z0 >= 0 && z0 < maze\_zsize)

{

//Check if a wall is on the way

if(cells[z0][x0].walls[moving\_dir])

allowed = 0;

}

//Check if the viewer is entering the maze

if(x1 >= 0 && x1 < maze\_xsize && z1 >= 0 && z1 < maze\_zsize)

{

//Check if a wall is on the way

if(cells[z1][x1].walls[opposite[moving\_dir]])

allowed = 0;

}

//Check if the viewer is going too far from the maze

if(x1 < -1 || x1 > maze\_xsize || z1 < -1 || z1 > maze\_zsize)

{

allowed = 0;

}

//If movement allowed, start\_movement from current position to position if(allowed && collected\_spheres < max\_spheres)

{

next\_x = x1;

next\_z = z1;

start\_movement();

}

//Check if a sphere is navigated

if (current\_x == sphere\_x && current\_z == sphere\_z)

{

collected\_spheres++;//Update the number of collected spheres

//Check if the viewer has collected all spheres on the level

if (collected\_spheres >= max\_spheres)

{

//Shift to the next level

collected\_spheres = 0;//Reset the number of collected spheres to zero

//Check for the maze size limit

if (maze\_xsize < 20 && maze\_zsize < 20)

{

max\_spheres++;//More spheres for next level

maze\_xsize++;//Larger maze for next level

maze\_zsize++; //Larger maze for next level

}

build\_maze();//Rebuild the maze for next level

}

place\_random\_sphere();

}

//Check if maze is completed

if (current\_x==complete\_x && current\_z+1==complete\_z)

{

check\_exit=1;

}

}

//Go to angle zoom mode

if(key == GLUT\_KEY\_PAGE\_UP)

{

current\_y++;

}

//Return from angle zoom mode

if(key == GLUT\_KEY\_PAGE\_DOWN)

{

if(current\_y > 0)

current\_y--;

}

glutPostRedisplay();

}  
.  
.  
.

//Menu function

void main\_menu(int index)

{

switch(index)

{

case(1):

{

collected\_spheres = 0;

build\_maze();

glutReshapeFunc(reshapefunc\_maze);

glutDisplayFunc(displayfunc\_maze);

glutMouseFunc(NULL);

break;

}

case(2):

{

exit(0);

break;

}

}

}

.

.

.

glutCreateMenu(main\_menu);

glutAddMenuEntry("Restart", 1);

glutAddMenuEntry("Exit", 2);

glutReshapeFunc(reshapefunc\_entry);

glutDisplayFunc(displayfunc\_entry);

glutKeyboardFunc(keyboardfunc);

glutMouseFunc(mousefunc\_entry);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

1. **An animated object using double buffering.**

//Draws timer countdown at the top of the screen

void draw\_countdown()

{

glPushMatrix();

glLoadIdentity();

glTranslatef(0, 0.45, -0.35);//Top of the screen

glScalef(0.0005, 0.0005, 0.2);

glColor3f(1, 0, 0);//Use red color

//Calculate digits of countdown value

int digit1 = countdown / 100 % 10;//First digit of countdown value

int digit2 = countdown / 10 % 10;//Second digit of countdown value

int digit3 = countdown / 1 % 10;//Third digit of countdown value

//Draw first digit if it's not leading zero

if (countdown >= 100)

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, '0' + digit1);

//Draw second digit if it's not leading zero

if (countdown >= 10)

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, '0' + digit2);

//Draw third digit always

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, '0' + digit3);

glPopMatrix();

}  
.  
.  
.  
//Timer function callback

void timerfunc(int ignored)

{

if (countdown > 0)

{

countdown--;

//Check if countdown reaches 0

if (countdown <= 0)

{

//Restart level

collected\_spheres = 0;// reset collected spheres to zero

build\_maze();// rebuild the maze

}

//Call this function again in 1 second

glutTimerFunc(1000, timerfunc, 0);

}

glutPostRedisplay();

}

1. **A texture-mapped object.**

//Loads bitmaps for texture mapping

int LoadBitmap(char \*filename)

{

int i, j=0; //Index variables

FILE \*l\_file; //File pointer

unsigned char \*l\_texture; //The pointer to the memory zone in which we will load the texture

// windows.h gives us these types to work with the Bitmap files

BITMAPFILEHEADER fileheader;

BITMAPINFOHEADER infoheader;

RGBTRIPLE rgb;

num\_texture++; // The counter of the current texture is increased

if( (l\_file = fopen(filename, "rb"))==NULL) return (-1); // Open the file for reading

fread(&fileheader, sizeof(fileheader), 1, l\_file); // Read the fileheader

fseek(l\_file, sizeof(fileheader), SEEK\_SET); // Jump the fileheader

fread(&infoheader, sizeof(infoheader), 1, l\_file); // and read the infoheader

// Now we need to allocate the memory for our image (width \* height \* color deep)

l\_texture = (byte \*) malloc(infoheader.biWidth \* infoheader.biHeight \* 4);

// And fill it with zeros

memset(l\_texture, 0, infoheader.biWidth \* infoheader.biHeight \* 4);

// At this point we can read every pixel of the image

for (i=0; i < infoheader.biWidth\*infoheader.biHeight; i++)

{

// We load an RGB value from the file

fread(&rgb, sizeof(rgb), 1, l\_file);

// And store it

l\_texture[j+0] = rgb.rgbtRed; // Red component

l\_texture[j+1] = rgb.rgbtGreen; // Green component

l\_texture[j+2] = rgb.rgbtBlue; // Blue component

l\_texture[j+3] = 255; // Alpha value

j += 4; // Go to the next position

}

fclose(l\_file); // Closes the file stream

glBindTexture(GL\_TEXTURE\_2D, num\_texture); // Bind the ID texture specified by the 2nd parameter

// The next commands sets the texture parameters

// If the u,v coordinates overflow the range 0,1 the image is repeated

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR); // The magnification function ("linear" produces better results)

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_NEAREST); //The minifying function

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE); // We don't combine the color with the original surface color, use only the texture map.

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

// Finally we define the 2d texture

glTexImage2D(GL\_TEXTURE\_2D, 0, 4, infoheader.biWidth, infoheader.biHeight, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, l\_texture);

// And create 2d mipmaps for the minifying function

gluBuild2DMipmaps(GL\_TEXTURE\_2D, 4, infoheader.biWidth, infoheader.biHeight, GL\_RGBA, GL\_UNSIGNED\_BYTE, l\_texture);

free(l\_texture); // Free the memory we used to load the texture

return (num\_texture); // Returns the current texture OpenGL ID

}  
.  
.  
.  
//Draws maze walls and floors

void draw\_maze()

{

glMaterialfv(GL\_FRONT, GL\_SPECULAR, specular\_color);

glMaterialf(GL\_FRONT, GL\_SHININESS, shine);

glEnable(GL\_LIGHTING);

int x = 0, z = 0;

//Draw cell walls

for (z = 0; z < maze\_zsize; z++)

{

for (x = 0; x < maze\_xsize; x++)

{

glEnable(GL\_TEXTURE\_2D);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, floor\_color);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ZERO);

glBindTexture(GL\_TEXTURE\_2D, wallTex);

glPushMatrix();

glTranslatef(x, 0, z);

//Draw back wall if present

if (cells[z][x].walls[ZPOS])

{

glPushMatrix();

glRotatef(180, 0, 1, 0);

glTranslatef(0, 0, -0.499);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, cells[z][x].color);

glNormal3f(0, 0, 1);

draw\_square();

//If at border of the maze, also draw exterior-facing wall

if (z == maze\_zsize - 1)

{

glTranslatef(0, 0, -0.002);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, exterior\_color);

glNormal3f(0, 0, -1);

draw\_square();

}

glPopMatrix();

}

//Draw left wall if present

if (cells[z][x].walls[XNEG])

{

glPushMatrix();

glRotatef(90, 0, 1, 0);

glTranslatef(0, 0, -0.499);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, cells[z][x].color);

glNormal3f(0, 0, 1);

draw\_square();

//If at border of the maze, also draw exterior-facing wall

if (x == 0)

{

glTranslatef(0, 0, -0.002);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, exterior\_color);

glNormal3f(0, 0, -1);

draw\_square();

}

glPopMatrix();

}

//Draw front wall if present

if (cells[z][x].walls[ZNEG])

{

glPushMatrix();

glTranslatef(0, 0, -0.499);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, cells[z][x].color);

glNormal3f(0, 0, 1);

draw\_square();

//If at border of the maze, also draw exterior-facing wall

if (z == 0)

{

glTranslatef(0, 0, -0.002);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, exterior\_color);

glNormal3f(0, 0, -1);

draw\_square();

}

glPopMatrix();

}

//Draw right wall if present

if (cells[z][x].walls[XPOS])

{

glPushMatrix();

glRotatef(-90, 0, 1, 0);

glTranslatef(0, 0, -0.499);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, cells[z][x].color);

glNormal3f(0, 0, 1);

draw\_square();

//If at border of the maze, also draw exterior-facing wall

if (x == maze\_xsize - 1)

{

glTranslatef(0, 0, -0.002);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, exterior\_color);

glNormal3f(0, 0, -1);

draw\_square();

}

glPopMatrix();

}

glPopMatrix();

}

}

//Draw cell floors

glEnable(GL\_TEXTURE\_2D);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, floor\_color);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ZERO);

glBindTexture(GL\_TEXTURE\_2D, floorTex);

for (z = 0 - 1; z < maze\_zsize + 1; z++)

{

for (x = 0 - 1; x < maze\_xsize + 1; x++)

{

glPushMatrix();

glTranslatef(x, -0.5, z);

glRotatef(90, -1, 0, 0);

glNormal3f(0, 0, 1);

draw\_square();

glPopMatrix();

}

}

glDisable(GL\_TEXTURE\_2D);

glDisable(GL\_LIGHTING);

}  
.  
.  
.  
//Quadratic object for texture mapping of the sun

GLUquadric \*qobj = gluNewQuadric();

gluQuadricTexture(qobj,GL\_TRUE);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D,sunTex);

glPushMatrix();

glTranslatef(sun\_pos[0], sun\_pos[1], sun\_pos[2]);

gluSphere(qobj,1.0,16,16);

glPopMatrix();

gluDeleteQuadric(qobj);

glDisable(GL\_TEXTURE\_2D);

1. **A semi-transparent object created by using alpha blending.**

//Draws sphere

void draw\_sphere()

{

glEnable(GL\_LIGHTING);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, sphere\_color);

glMaterialfv(GL\_RIGHT, GL\_SPECULAR, specular\_color);

glMaterialf(GL\_RIGHT, GL\_SHININESS, shine);

//Quadratic Object for texture mapping into sphere

GLUquadric \*qobj = gluNewQuadric();

gluQuadricTexture(qobj,GL\_TRUE);

glEnable(GL\_TEXTURE\_2D);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE);

glBindTexture(GL\_TEXTURE\_2D,sphereTex);

glTranslatef(sphere\_x, -0.25, sphere\_z);

gluSphere(qobj,0.25,16,16);

gluDeleteQuadric(qobj);

glDisable(GL\_BLEND);

glDisable(GL\_TEXTURE\_2D);

glDisable(GL\_LIGHTING);

}

1. **Realistic coloring that utilizes lighting and shading.**

//Flashlight properties

GLfloat flashlight\_color[] = {1, 1, 1, 1};

GLfloat flashlight\_pos[] = {0, 0, 0, 1};

//Sun properties

GLfloat sun\_color[] = {0.7, 0.7, 0.7, 1};

GLfloat sun\_pos[] = {0, 0, 0, 1};

//Material properties

GLfloat floor\_color[] = {0.8, 0.8, 0.8, 0.8};

GLfloat exterior\_color[] = {1, 1, 1, 1};

GLfloat sphere\_color[] = {0.8, 0.8, 0.8, 1};

GLfloat specular\_color[] = {0.6, 0.6, 0.6, 1};

GLfloat shine = 10;  
.  
.  
.  
//Enable flashlight as a light source

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, flashlight\_color);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, flashlight\_color);

glLightfv(GL\_LIGHT0, GL\_POSITION, flashlight\_pos);

glLightf(GL\_LIGHT0, GL\_SPOT\_CUTOFF, 45);

glLightf(GL\_LIGHT0, GL\_SPOT\_EXPONENT, 90);

glLightf(GL\_LIGHT0, GL\_CONSTANT\_ATTENUATION, 0.07);

glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTENUATION, 0.07);  
.  
.  
.  
//Draws maze walls and floors

void draw\_maze()

{

glMaterialfv(GL\_FRONT, GL\_SPECULAR, specular\_color);

glMaterialf(GL\_FRONT, GL\_SHININESS, shine);

glEnable(GL\_LIGHTING);

int x = 0, z = 0;

//Draw cell walls

for (z = 0; z < maze\_zsize; z++)

{

for (x = 0; x < maze\_xsize; x++)

{

glEnable(GL\_TEXTURE\_2D);

glMaterialfv(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE, floor\_color);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ZERO);

glBindTexture(GL\_TEXTURE\_2D, wallTex);

glPushMatrix();

glTranslatef(x, 0, z);

//Draw back wall if present

if (cells[z][x].walls[ZPOS])

{

glPushMatrix();

glRotatef(180, 0, 1, 0);

glTranslatef(0, 0, -0.499);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, cells[z][x].color);

glNormal3f(0, 0, 1);

draw\_square();

//If at border of the maze, also draw exterior-facing wall

if (z == maze\_zsize - 1)

{

glTranslatef(0, 0, -0.002);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, exterior\_color);

glNormal3f(0, 0, -1);

draw\_square();

}

glPopMatrix();

}

//Draw left wall if present

if (cells[z][x].walls[XNEG])

{

glPushMatrix();

glRotatef(90, 0, 1, 0);

glTranslatef(0, 0, -0.499);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, cells[z][x].color);

glNormal3f(0, 0, 1);

draw\_square();

//If at border of the maze, also draw exterior-facing wall

if (x == 0)

{

glTranslatef(0, 0, -0.002);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, exterior\_color);

glNormal3f(0, 0, -1);

draw\_square();

}

glPopMatrix();

}

//Draw front wall if present

if (cells[z][x].walls[ZNEG])

{

glPushMatrix();

glTranslatef(0, 0, -0.499);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, cells[z][x].color);

glNormal3f(0, 0, 1);

draw\_square();

//If at border of the maze, also draw exterior-facing wall

if (z == 0)

{

glTranslatef(0, 0, -0.002);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, exterior\_color);

glNormal3f(0, 0, -1);

draw\_square();

}

glPopMatrix();

}

//Draw right wall if present

if (cells[z][x].walls[XPOS])

{

glPushMatrix();

glRotatef(-90, 0, 1, 0);

glTranslatef(0, 0, -0.499);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, cells[z][x].color);

glNormal3f(0, 0, 1);

draw\_square();

//If at border of the maze, also draw exterior-facing wall

if (x == maze\_xsize - 1)

{

glTranslatef(0, 0, -0.002);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, exterior\_color);

glNormal3f(0, 0, -1);

draw\_square();

}

glPopMatrix();

}

glPopMatrix();

}

}

//Draw cell floors

glEnable(GL\_TEXTURE\_2D);

glMaterialfv(GL\_RIGHT, GL\_AMBIENT\_AND\_DIFFUSE, floor\_color);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ZERO);

glBindTexture(GL\_TEXTURE\_2D, floorTex);

for (z = 0 - 1; z < maze\_zsize + 1; z++)

{

for (x = 0 - 1; x < maze\_xsize + 1; x++)

{

glPushMatrix();

glTranslatef(x, -0.5, z);

glRotatef(90, -1, 0, 0);

glNormal3f(0, 0, 1);

draw\_square();

glPopMatrix();

}

}

glDisable(GL\_TEXTURE\_2D);

glDisable(GL\_LIGHTING);

}

1. **Start the program with a two-dimensional menu giving the user options to customize your program.**

//Display function callback for entry mode

void displayfunc\_entry()

{

glDisable(GL\_DEPTH\_TEST);

glClearColor(0.5, 0.6, 0.6, 1);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0, 0, 0);

//Draw "X Size" text

glPushMatrix();

glTranslatef(60, 54 - 1 \* 20, 0);

draw\_text("X Size");

glPopMatrix();

//Draw "Z Size" text

glPushMatrix();

glTranslatef(220, 54 - 1 \* 20, 0);

draw\_text("Z Size");

glPopMatrix();

//Draw possible maze sizes

for(int i = 1; i < 18; i++)

{

for(int j = 0; j < 2; j++)

{

int x0 = 60 + j \* 160;

int x1 = 145 + j \* 160;

int y0 = 40 + i \* 20;

int y1 = 56 + i \* 20;

glColor3f(0.8, 0.8, 0.8);

if(j == 0 && maze\_xsize == i + 1)

glColor3f(0.8, 0.6, 0.6);

if(j == 1 && maze\_zsize == i + 1)

glColor3f(0.8, 0.6, 0.6);

glRecti(x0, y0, x1, y1);

glPushMatrix();

glTranslatef(x0 + 40, 54 + i \* 20, 0);

glColor3f(0, 0, 0);

draw\_number(i + 1);

glPopMatrix();

}

}

//Draw OK button

int x0 = 400;

int x1 = 480;

int y0 = 440;

int y1 = 480;

glColor3f(0.8, 0.8, 0.8);

glRecti(x0+45, y0+8, x1, y1-8);

//Draw "OK" text

glColor3f(0, 0, 0);

glPushMatrix();

glTranslatef(x0 + 45, y0 + 30, 0);

draw\_text("OK");

glPopMatrix();

//Draw EXIT button

glColor3f(0.8, 0.8, 0.8);

glRecti(x0-20, y0+8, x1-50, y1-8);

//Draw EXIT text

glColor3f(0, 0, 0);

glPushMatrix();

glTranslatef(x0 - 20, y0 + 30, 0);

draw\_text("EXIT");

glPopMatrix();

//Draw Navigation Prompt

glColor3f(0, 0, 0);

glPushMatrix();

glTranslatef(x0 -350, y0 + 30, 0);

draw\_text("Navigation Map");

glPopMatrix();

//Draw Nav On button

glColor3f(0.8, 0.8, 0.8);

if(map==1)

glColor3f(0.8, 0.6, 0.6);

glRecti(x0-150, y0+8, x1-195, y1-8);

//Draw Navigation On Prompt

glColor3f(0, 0, 0);

glPushMatrix();

glTranslatef(x0 -150, y0 + 30, 0);

draw\_text("ON");

glPopMatrix();

//Draw Nav Off button

glColor3f(0.8, 0.8, 0.8);

if(map==0)

glColor3f(0.8, 0.6, 0.6);

glRecti(x0-100, y0+8, x1-130, y1-8);

//Draw Navigation Off Prompt

glColor3f(0, 0, 0);

glPushMatrix();

glTranslatef(x0 -100, y0 + 30, 0);

draw\_text("OFF");

glPopMatrix();

glColor3f(1, 1, 1);

//Swap back and front buffers for double buffering to work

glutSwapBuffers();

}  
.  
.  
.  
//Mouse function callback for entry mode

void mousefunc\_entry(int button, int state, int x, int y)

{

if(button == GLUT\_LEFT\_BUTTON)

{

//Check if number buttons are clicked

for(int i = 0; i < 20; i++) {

for(int j = 0; j < 2; j++) {

int x0 = 60 + j \* 160;

int x1 = 145 + j \* 160;

int y0 = 40 + i \* 20;

int y1 = 56 + i \* 20;

if(x >= x0 && x < x1 && y >= y0 && y < y1) {

if(j == 0)

{

maze\_xsize = i + 1;

}

else

{

maze\_zsize = i + 1;

}

}

}

}

//Check for Navigation Map Options

if(x >= 250 && x < 285 && y >= 440+8 && y < 480-8)

map=1; //Map on

if(x >= 300 && x < 350 && y >= 440+8 && y < 480-8)

map=0; //Map Off

//Check if OK button is clicked

if(x >= 400+45 && x < 480 && y >= 440+8 && y < 480-8)

{

build\_maze();

place\_random\_sphere();

//Switch to maze mode

glutDisplayFunc(displayfunc\_maze);

glutMouseFunc(mousefunc\_maze);

glutSpecialFunc(specialfunc);

//Start countdown

glutTimerFunc(1000, timerfunc, 0);

}

//Check if EXIT button is clicked

if(x >= 400-20 && x < 480-50 && y >= 440+8 && y < 480-8)

exit(0);

}

glutPostRedisplay();

}

1. **Allow the user to choose from multiple viewing options while in the program.**

//Go to angle zoom mode

if(key == GLUT\_KEY\_PAGE\_UP)

{

current\_y++;

}

//Return from angle zoom mode

if(key == GLUT\_KEY\_PAGE\_DOWN)

{

if(current\_y > 0)

current\_y--;

}