Student Name: Hongyuan Jin 靳鸿媛
Student ID: 1409853G-I011-0046

3D interaction maze implementation

CS104 Fundamentals of Computer Graphics

1. Design outline

This simple Maze game is designed according to the guidance provided in the project description. There are four major parts in the development of this maze game including: 1) general environment settings, 2) the construction of the basic scene set, 3) the drawing of the player and 4) collision detection. In the construction of the basic scene set, ground, walls and the exit door are drawn according to the positions provided in the maze map and textures are added respectively. The figure of the player object is adopted from a pink mosaic pig. Fog effect and a blue sky are added and collision detection and player replacing are implemented. The general design style is inspired by a famous game called *Minecraft.*

1.1 General environment settings

In the `init()` function, several texture objects are introduced using the OpenGL texture interface. In the `display()` function, apart from the projection, scene setting codes provided, depth test is enabled and shade model is set to `GL_SMOOTH`. Fog effect is also created there with the sky color (background clear color) set to sky blue. After `DrawGround()` and `DrawWalls()` are called, lighting is enabled to avoid unnecessary lighting impact on the scene building.

1.2 Construction of basic scene set

After iterating the map data from `readmap()`, the initial block of the player as well as the exit is memorized. Utilizing `doorI` and `doorJ` enables me to draw the exit door together with the ground for once. In the `DrawGround()`, the current object coordinate is moved to the remembered position of the door and door texture is mapped to the quadric.

Drawing walls is the challenging part. In the `DrawWalls()` function, I iterate through the `_map` array and check whether the value of the current item is 1 or not. If it’s 1, draw four walls for a wall block in the following order:

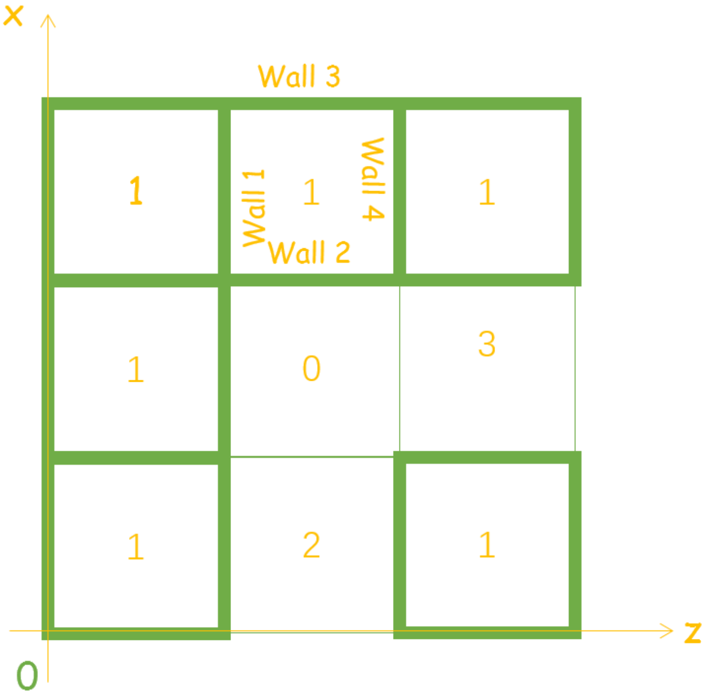


Figure. 1 Wall block drawing order

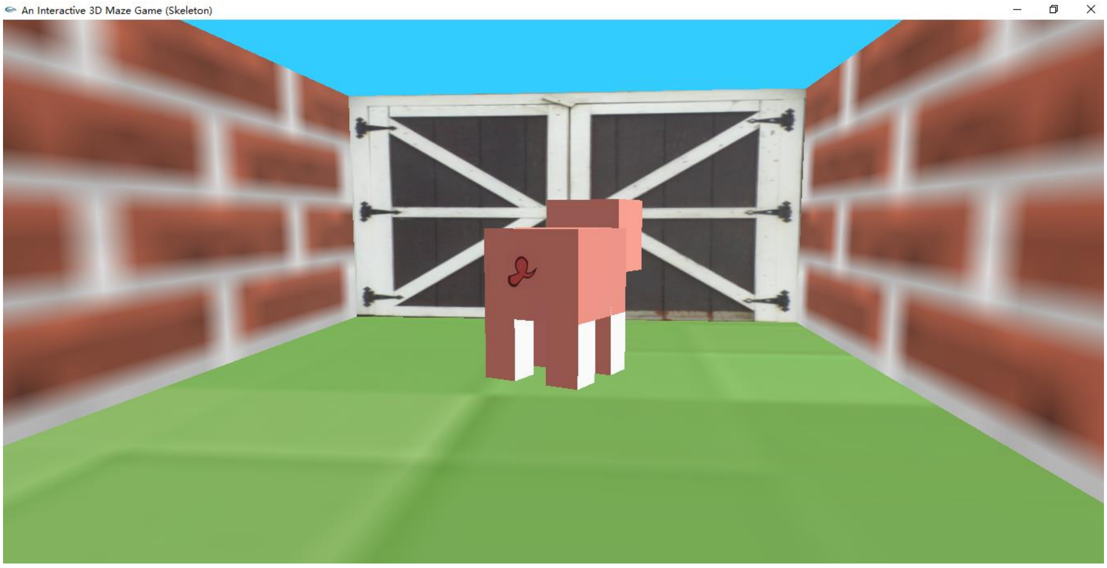


Figure. 2 The effect of the walls, the ground and the exit door.

1.3 The drawing of the player

I adopted the figure of a pink pig from the game Minecraft. The player is composed of 7 parts: head, body, four limbs and a tail. All of them are in a shape of solid cube. Its material includes a pink diffuse and dark pink ambient, and the light applied is a parallel light with the same diffuse and ambient.

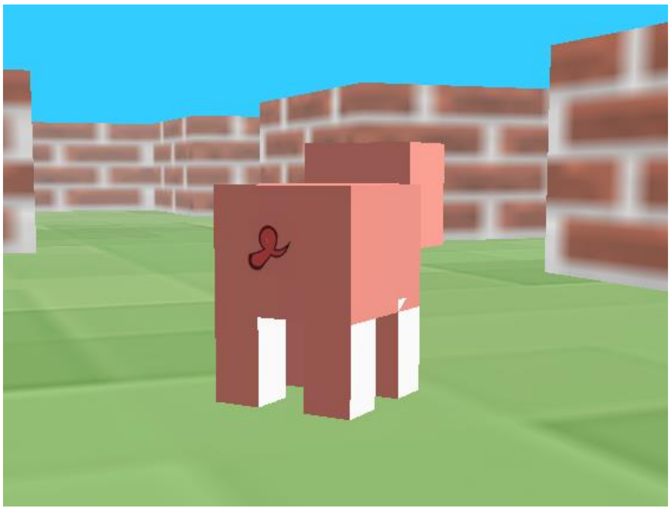


Figure. 3 The effect of the player

One thing worth mentioning is that I fixated the angle of the player so that its position remains forward no matter how we spin the maze set.

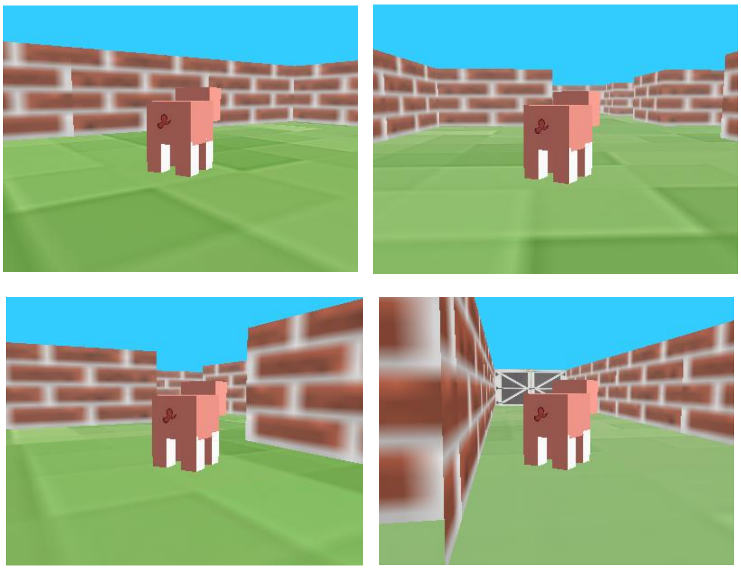


Figure. 4 The player remains a forward position no matter how we rotate the set.

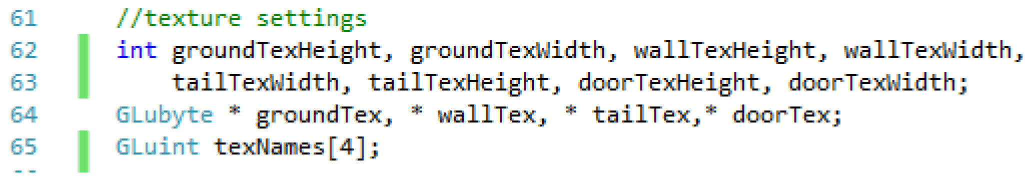
1.4 Collision detection

I implemented collision detection in a dead simple by high-performant way. Although it would not consider the case that the edge of the player is overlapped with the walls, it provides us a simple mechanism to return to the previous valid position when player's intrusion to the wall block is detected.

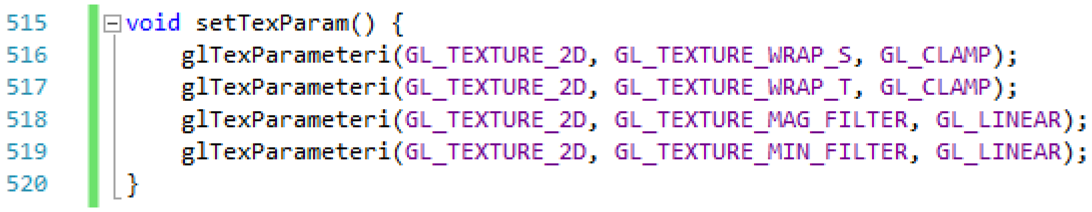
When `_player.forward` is set not equal to zero and `checkcollide()` function is called, I would calculate the current block that the player's supposed to go. After acquiring the block number curI and curJ and check whether the corresponding block in _map array is equal to 1 or not, the movement of the player is decided. If it’s not equal to 1, that means collision does not happen and the player is allowed to move forward. In the meantime this block would be stored as preValidI and preValidJ; otherwise, collision happens and the player is reset to the center position of the pre-valid block defined by preValidI and preValidJ. Please check out more details in the code fragments.

2. Key code fragments or algorithms of your program

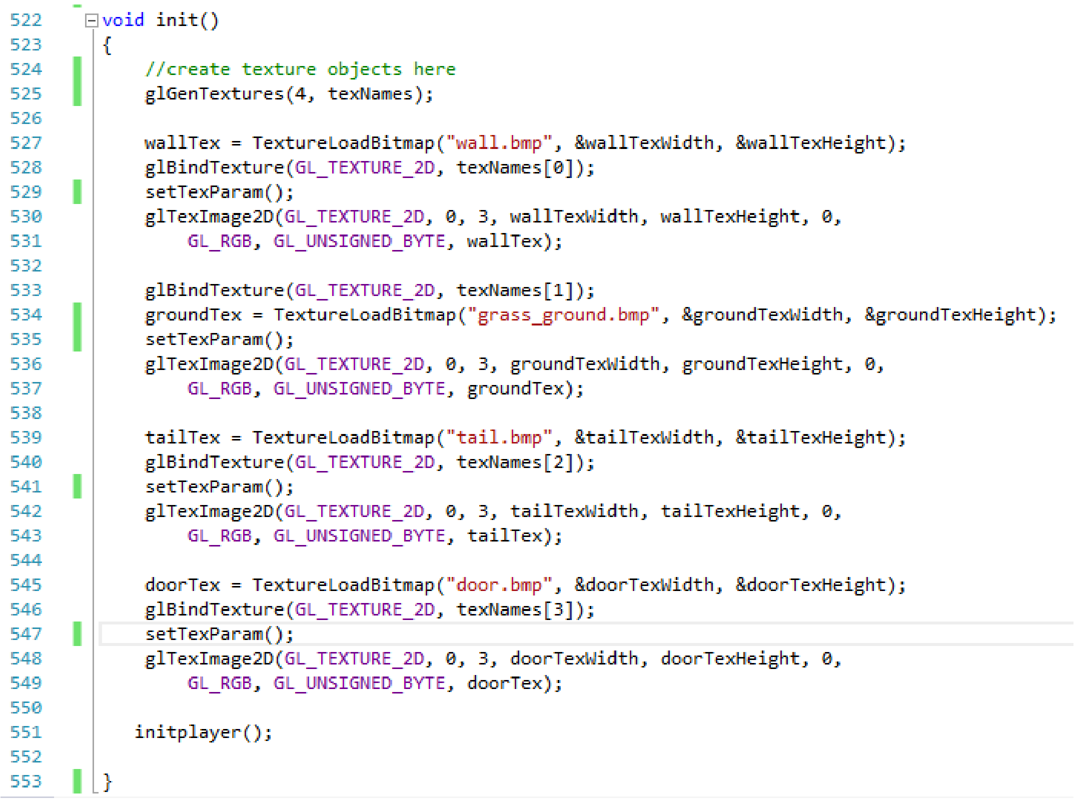
Texture object creation



```
//texture settings
int groundTexHeight, groundTexWidth, wallTexHeight, wallTexWidth,
    tailTexWidth, tailTexHeight, doorTexHeight, doorTexWidth;
GLubyte * groundTex, * wallTex, * tailTex,* doorTex;
GLuint texNames[4];
```

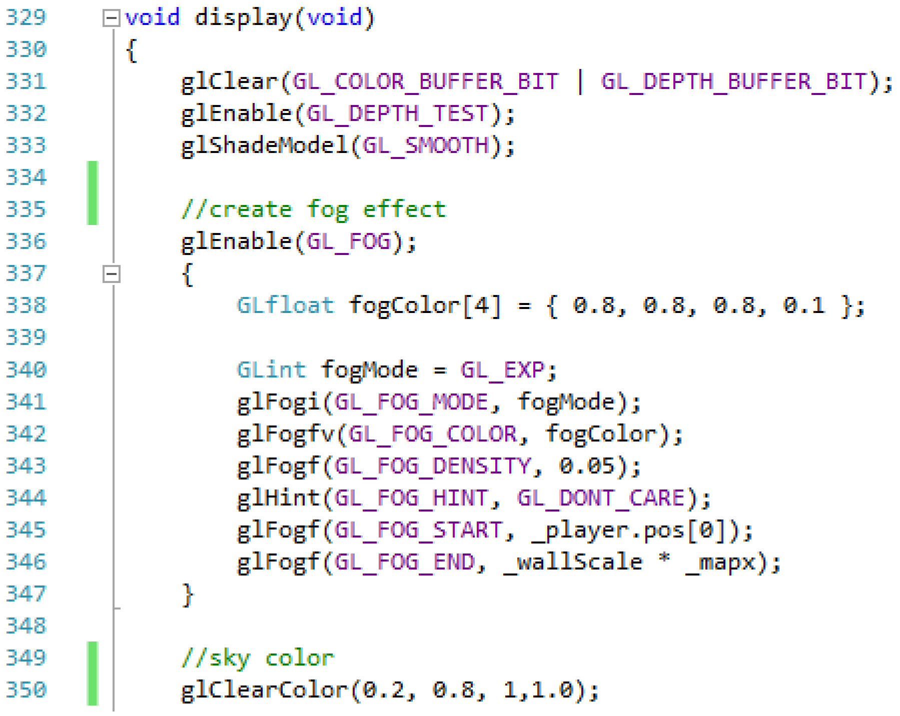


```
void setTexParam() {
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
}
```

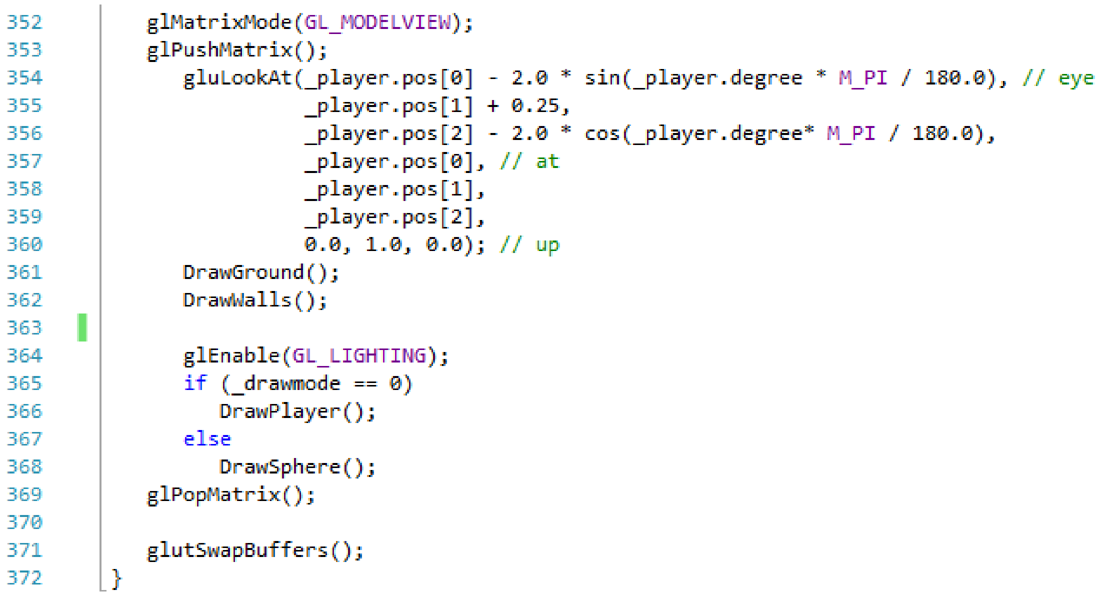


```
void init()
{
    //create texture objects here
    glGenTextures(4, texNames);

    wallTex = TextureLoadBitmap("wall.bmp", &wallTexWidth, &wallTexHeight);
    glBindTexture(GL_TEXTURE_2D, texNames[0]);
    setTexParam();
    glTexImage2D(GL_TEXTURE_2D, 0, 3, wallTexWidth, wallTexHeight, 0,
        GL_RGB, GL_UNSIGNED_BYTE, wallTex);

    glBindTexture(GL_TEXTURE_2D, texNames[1]);
    groundTex = TextureLoadBitmap("grass_ground.bmp", &groundTexWidth, &groundTexHeight);
    setTexParam();
    glTexImage2D(GL_TEXTURE_2D, 0, 3, groundTexWidth, groundTexHeight, 0,
        GL_RGB, GL_UNSIGNED_BYTE, groundTex);

    tailTex = TextureLoadBitmap("tail.bmp", &tailTexWidth, &tailTexHeight);
    glBindTexture(GL_TEXTURE_2D, texNames[2]);
    setTexParam();
    glTexImage2D(GL_TEXTURE_2D, 0, 3, tailTexWidth, tailTexHeight, 0,
        GL_RGB, GL_UNSIGNED_BYTE, tailTex);

    doorTex = TextureLoadBitmap("door.bmp", &doorTexWidth, &doorTexHeight);
    glBindTexture(GL_TEXTURE_2D, texNames[3]);
    setTexParam();
    glTexImage2D(GL_TEXTURE_2D, 0, 3, doorTexWidth, doorTexHeight, 0,
        GL_RGB, GL_UNSIGNED_BYTE, doorTex);

   initplayer();

}
```

General environment settings

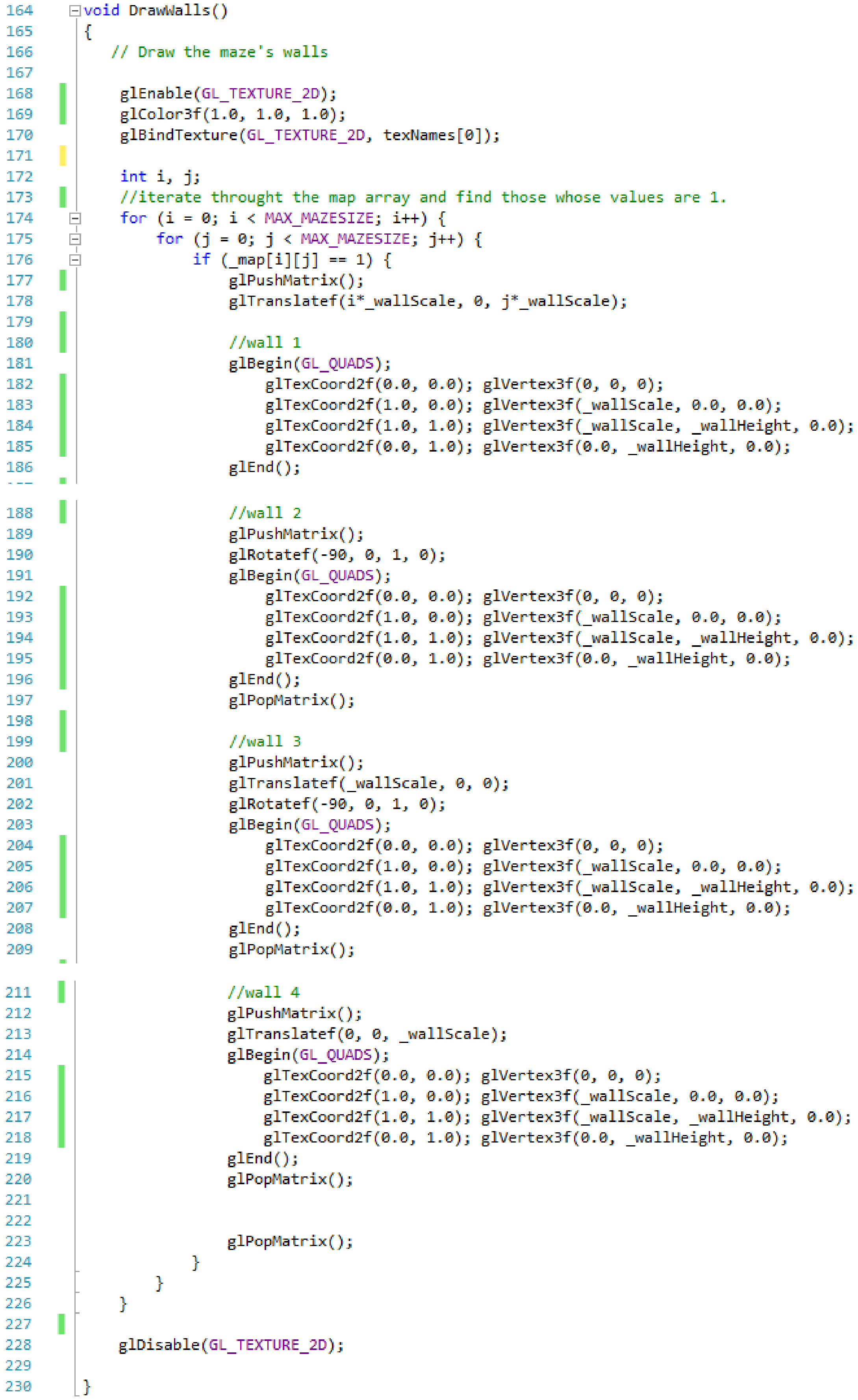


```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);

    //create fog effect
    glEnable(GL_FOG);
    {
        GLfloat fogColor[4] = { 0.8, 0.8, 0.8, 0.1 };

        GLint fogMode = GL_EXP;
        glFogi(GL_FOG_MODE, fogMode);
        glFogfv(GL_FOG_COLOR, fogColor);
        glFogf(GL_FOG_DENSITY, 0.05);
        glHint(GL_FOG_HINT, GL_DONT_CARE);
        glFogf(GL_FOG_START, _player.pos[0]);
        glFogf(GL_FOG_END, _wallScale * _mapx);
    }

    //sky color
    glClearColor(0.2, 0.8, 1,1.0);
```



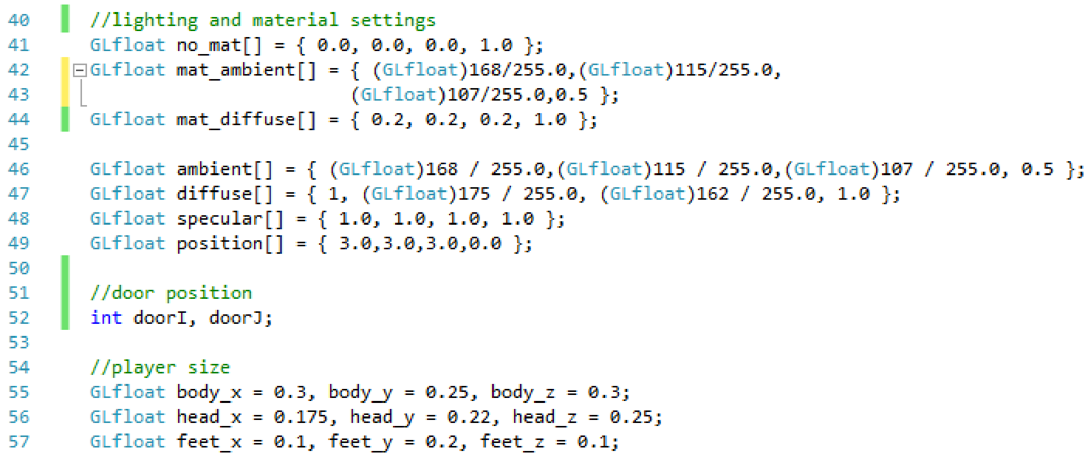
```
   glMatrixMode(GL_MODELVIEW);
   glPushMatrix();
      gluLookAt(_player.pos[0] - 2.0 * sin(_player.degree * M_PI / 180.0), // eye
                _player.pos[1] + 0.25,
                _player.pos[2] - 2.0 * cos(_player.degree* M_PI / 180.0),
                _player.pos[0], // at
                _player.pos[1],
                _player.pos[2],
                0.0, 1.0, 0.0); // up
      DrawGround();
      DrawWalls();

      glEnable(GL_LIGHTING);
      if (_drawmode == 0)
         DrawPlayer();
      else
         DrawSphere();
   glPopMatrix();

   glutSwapBuffers();
}
```

Drawing walls

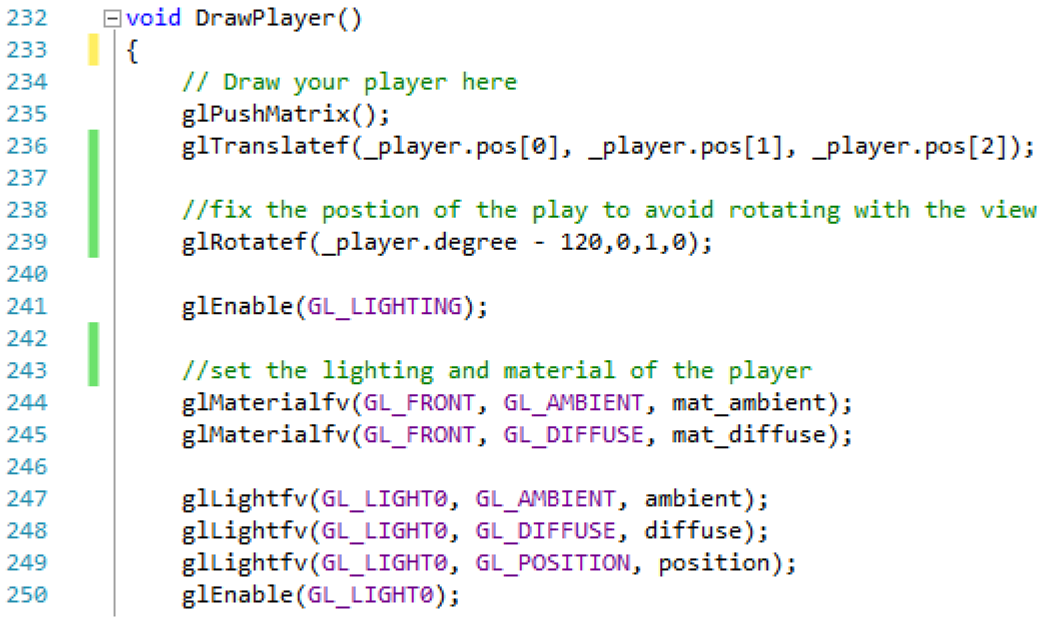


```
void DrawWalls()
{
   // Draw the maze's walls

    glEnable(GL_TEXTURE_2D);
    glColor3f(1.0, 1.0, 1.0);
    glBindTexture(GL_TEXTURE_2D, texNames[0]);

    int i, j;
    //iterate throught the map array and find those whose values are 1.
    for (i = 0; i < MAX_MAZESIZE; i++) {
        for (j = 0; j < MAX_MAZESIZE; j++) {
            if (_map[i][j] == 1) {
                glPushMatrix();
                glTranslatef(i*_wallScale, 0, j*_wallScale);

                //wall 1
                glBegin(GL_QUADS);
                    glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
                    glTexCoord2f(1.0, 0.0); glVertex3f(_wallScale, 0.0, 0.0);
                    glTexCoord2f(1.0, 1.0); glVertex3f(_wallScale, _wallHeight, 0.0);
                    glTexCoord2f(0.0, 1.0); glVertex3f(0.0, _wallHeight, 0.0);
                glEnd();

                //wall 2
                glPushMatrix();
                glRotatef(-90, 0, 1, 0);
                glBegin(GL_QUADS);
                    glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
                    glTexCoord2f(1.0, 0.0); glVertex3f(_wallScale, 0.0, 0.0);
                    glTexCoord2f(1.0, 1.0); glVertex3f(_wallScale, _wallHeight, 0.0);
                    glTexCoord2f(0.0, 1.0); glVertex3f(0.0, _wallHeight, 0.0);
                glEnd();
                glPopMatrix();

                //wall 3
                glPushMatrix();
                glTranslatef(_wallScale, 0, 0);
                glRotatef(-90, 0, 1, 0);
                glBegin(GL_QUADS);
                    glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
                    glTexCoord2f(1.0, 0.0); glVertex3f(_wallScale, 0.0, 0.0);
                    glTexCoord2f(1.0, 1.0); glVertex3f(_wallScale, _wallHeight, 0.0);
                    glTexCoord2f(0.0, 1.0); glVertex3f(0.0, _wallHeight, 0.0);
                glEnd();
                glPopMatrix();

                //wall 4
                glPushMatrix();
                glTranslatef(0, 0, _wallScale);
                glBegin(GL_QUADS);
                    glTexCoord2f(0.0, 0.0); glVertex3f(0, 0, 0);
                    glTexCoord2f(1.0, 0.0); glVertex3f(_wallScale, 0.0, 0.0);
                    glTexCoord2f(1.0, 1.0); glVertex3f(_wallScale, _wallHeight, 0.0);
                    glTexCoord2f(0.0, 1.0); glVertex3f(0.0, _wallHeight, 0.0);
                glEnd();
                glPopMatrix();


                glPopMatrix();
            }
        }
    }

    glDisable(GL_TEXTURE_2D);

}
```

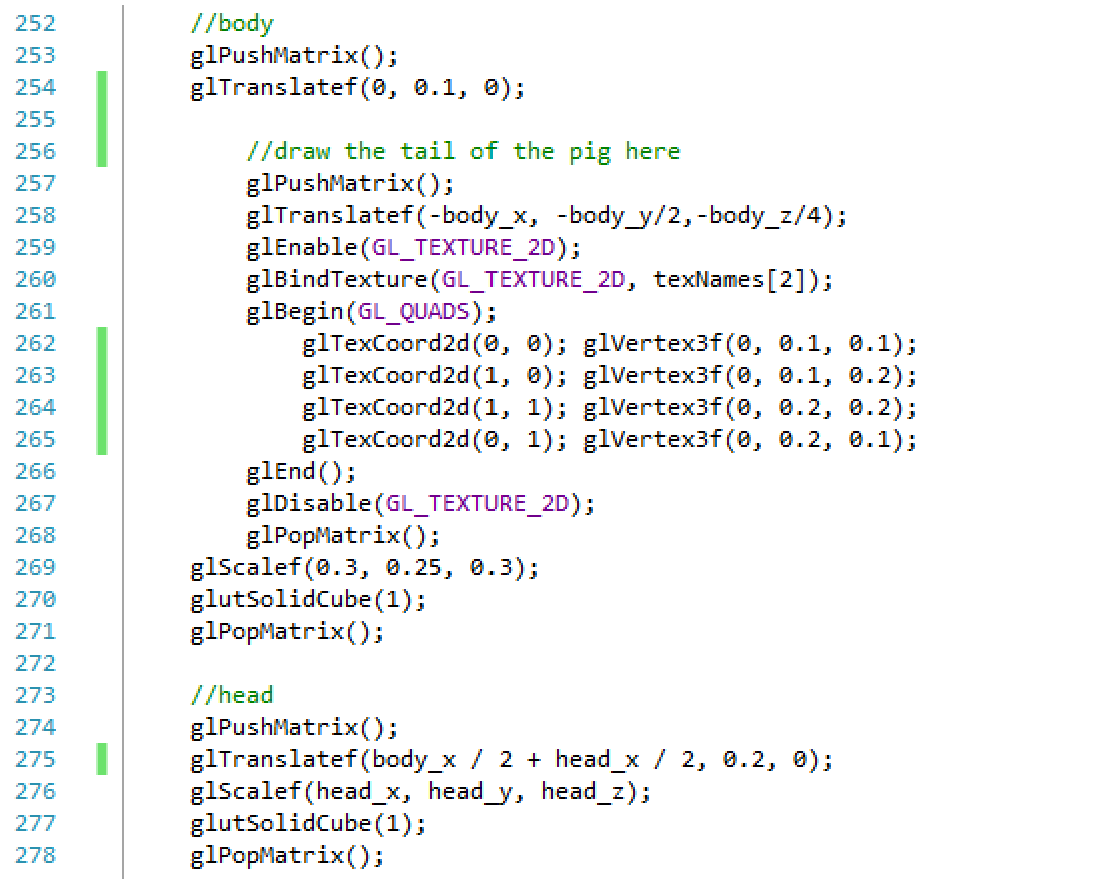
Drawing player



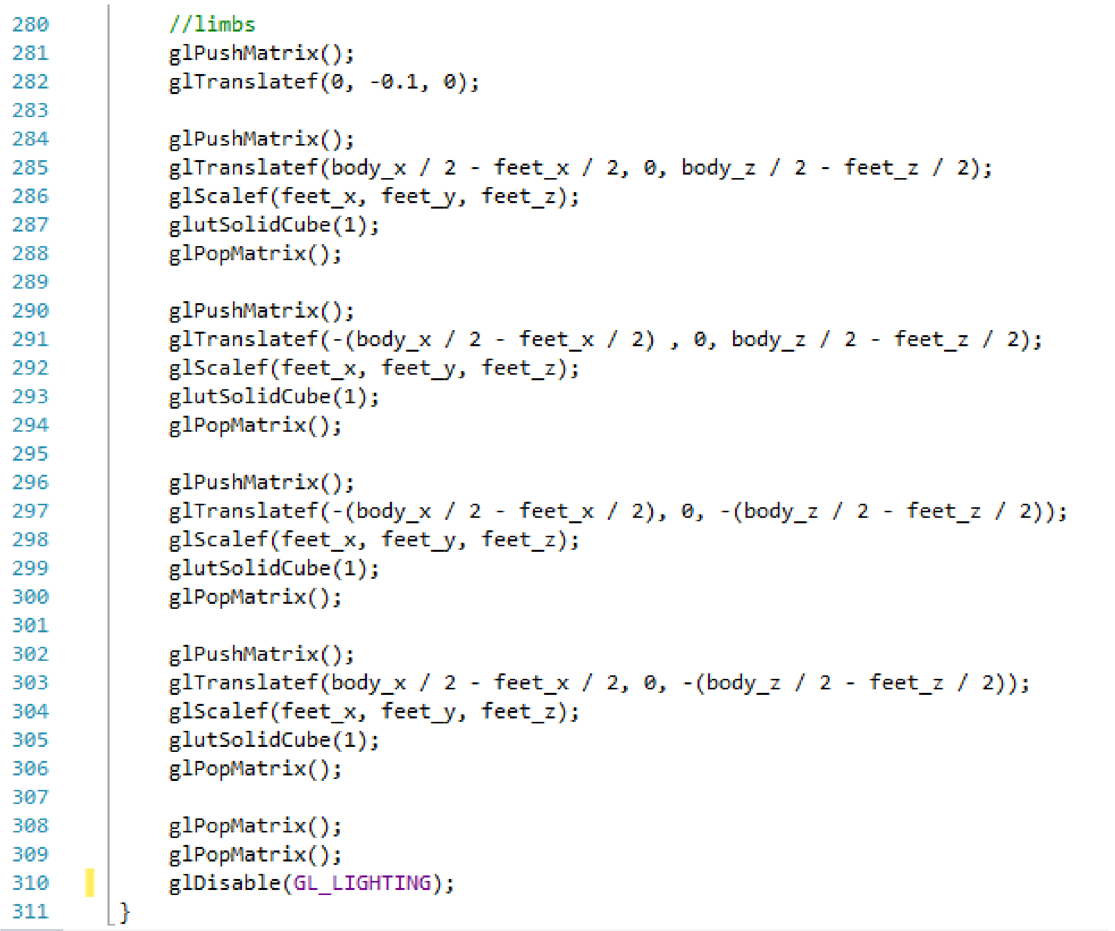
```
//lighting and material settings
GLfloat no_mat[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat mat_ambient[] = { (GLfloat)168/255.0,(GLfloat)115/255.0,
                          (GLfloat)107/255.0,0.5 };
GLfloat mat_diffuse[] = { 0.2, 0.2, 0.2, 1.0 };

GLfloat ambient[] = { (GLfloat)168 / 255.0,(GLfloat)115 / 255.0,(GLfloat)107 / 255.0, 0.5 };
GLfloat diffuse[] = { 1, (GLfloat)175 / 255.0, (GLfloat)162 / 255.0, 1.0 };
GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat position[] = { 3.0,3.0,3.0,0.0 };

//door position
int doorI, doorJ;

//player size
GLfloat body_x = 0.3, body_y = 0.25, body_z = 0.3;
GLfloat head_x = 0.175, head_y = 0.22, head_z = 0.25;
GLfloat feet_x = 0.1, feet_y = 0.2, feet_z = 0.1;
```



```
void DrawPlayer()
{
    // Draw your player here
    glPushMatrix();
    glTranslatef(_player.pos[0], _player.pos[1], _player.pos[2]);

    //fix the postion of the play to avoid rotating with the view
    glRotatef(_player.degree - 120,0,1,0);

    glEnable(GL_LIGHTING);

    //set the lighting and material of the player
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    glEnable(GL_LIGHT0);
```

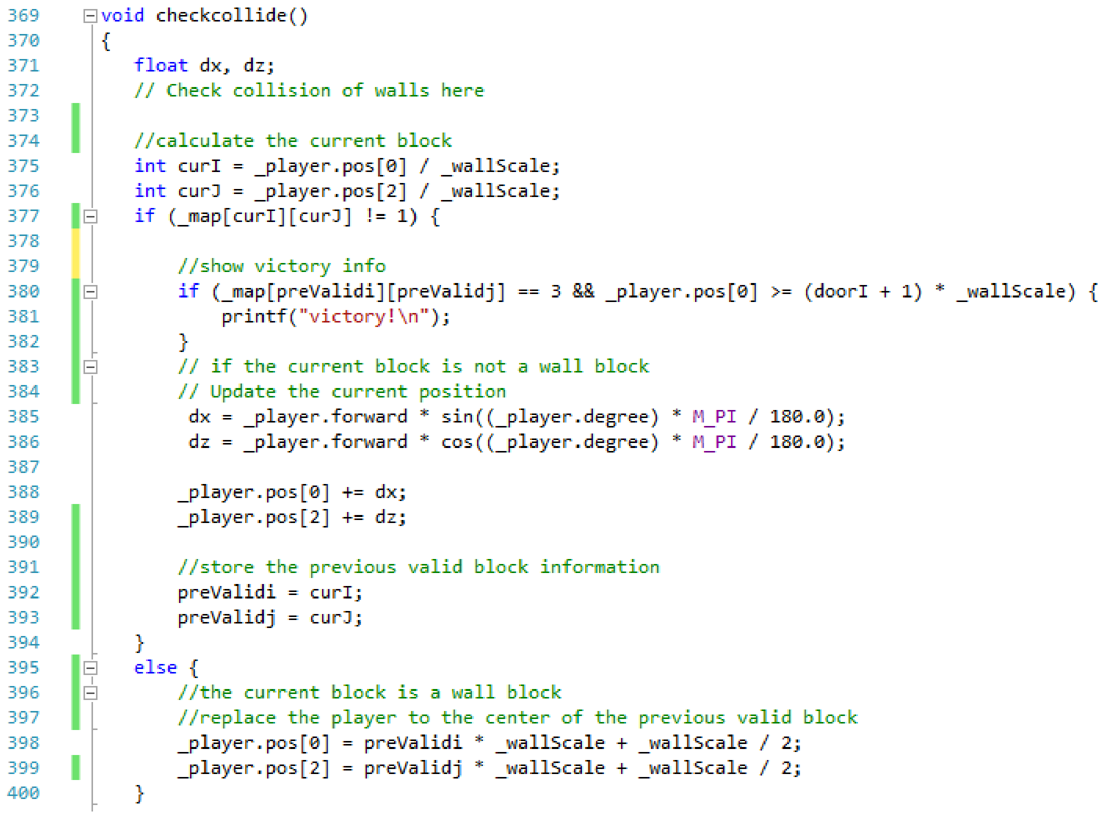


```
    //body
    glPushMatrix();
    glTranslatef(0, 0.1, 0);

        //draw the tail of the pig here
        glPushMatrix();
        glTranslatef(-body_x, -body_y/2,-body_z/4);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, texNames[2]);
        glBegin(GL_QUADS);
            glTexCoord2d(0, 0); glVertex3f(0, 0.1, 0.1);
            glTexCoord2d(1, 0); glVertex3f(0, 0.1, 0.2);
            glTexCoord2d(1, 1); glVertex3f(0, 0.2, 0.2);
            glTexCoord2d(0, 1); glVertex3f(0, 0.2, 0.1);
        glEnd();
        glDisable(GL_TEXTURE_2D);
        glPopMatrix();
    glScalef(0.3, 0.25, 0.3);
    glutSolidCube(1);
    glPopMatrix();

    //head
    glPushMatrix();
    glTranslatef(body_x / 2 + head_x / 2, 0.2, 0);
    glScalef(head_x, head_y, head_z);
    glutSolidCube(1);
    glPopMatrix();
```



```
    //limbs
    glPushMatrix();
    glTranslatef(0, -0.1, 0);

    glPushMatrix();
    glTranslatef(body_x / 2 - feet_x / 2, 0, body_z / 2 - feet_z / 2);
    glScalef(feet_x, feet_y, feet_z);
    glutSolidCube(1);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-(body_x / 2 - feet_x / 2) , 0, body_z / 2 - feet_z / 2);
    glScalef(feet_x, feet_y, feet_z);
    glutSolidCube(1);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-(body_x / 2 - feet_x / 2), 0, -(body_z / 2 - feet_z / 2));
    glScalef(feet_x, feet_y, feet_z);
    glutSolidCube(1);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(body_x / 2 - feet_x / 2, 0, -(body_z / 2 - feet_z / 2));
    glScalef(feet_x, feet_y, feet_z);
    glutSolidCube(1);
    glPopMatrix();

    glPopMatrix();
    glPopMatrix();
    glDisable(GL_LIGHTING);
}
```

Collision detection



```
void checkcollide()
{
    float dx, dz;
    // Check collision of walls here

    //calculate the current block
    int curI = _player.pos[0] / _wallScale;
    int curJ = _player.pos[2] / _wallScale;
    if (_map[curI][curJ] != 1) {

        //show victory info
        if (_map[preValidi][preValidj] == 3 && _player.pos[0] >= (doorI + 1) * _wallScale) {
            printf("victory!\n");
        }
        // if the current block is not a wall block
        // Update the current position
         dx = _player.forward * sin((_player.degree) * M_PI / 180.0);
         dz = _player.forward * cos((_player.degree) * M_PI / 180.0);

        _player.pos[0] += dx;
        _player.pos[2] += dz;

        //store the previous valid block information
        preValidi = curI;
        preValidj = curJ;
    }
    else {
        //the current block is a wall block
        //replace the player to the center of the previous valid block
        _player.pos[0] = preValidi * _wallScale + _wallScale / 2;
        _player.pos[2] = preValidj * _wallScale + _wallScale / 2;
    }
```

3. How to use my program;

Please double click the run.bat file to run the program directly.

If you want to run the source code, please ensure that `glu32.dll` and `glut32.dll` is under your system's path and `glut.h` and `glut32.lib` is under your Visual Studio library folder.

For example, my `glut.h` is under

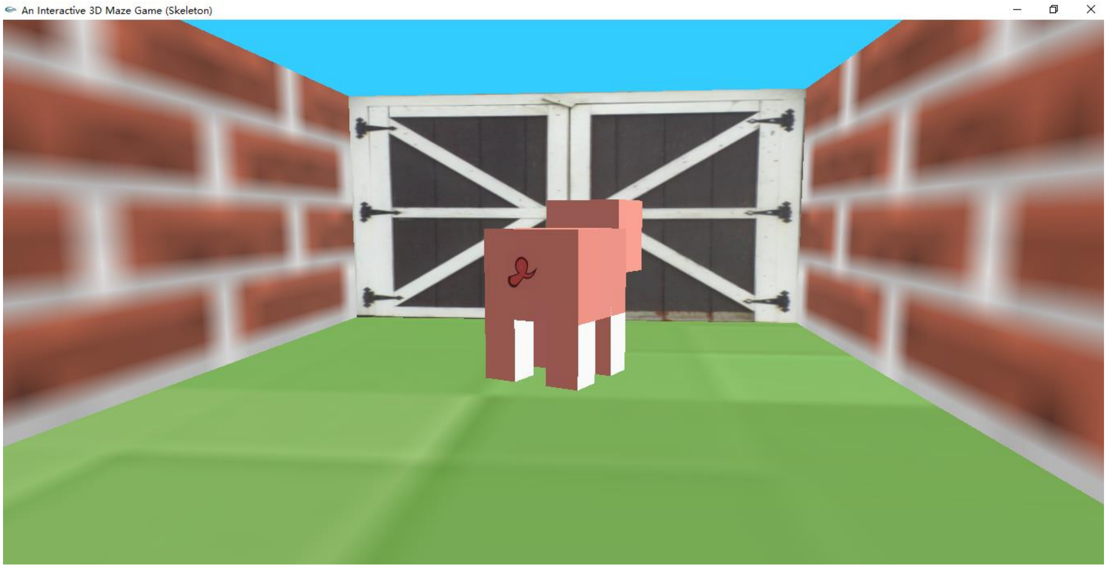
```
  C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\include\
```

and my `glut32.lib` is under

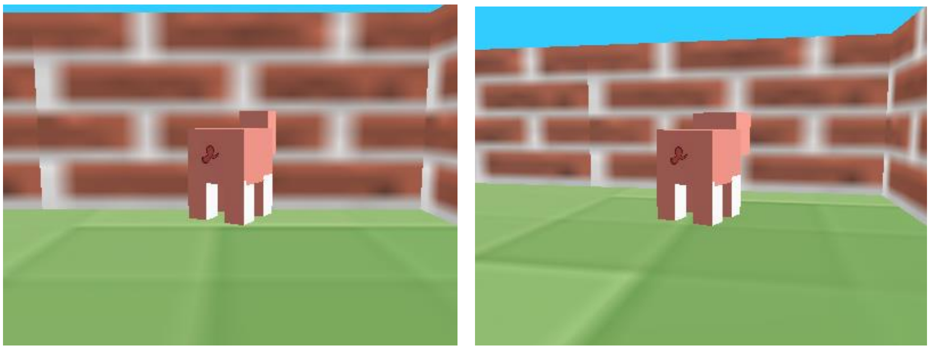
```
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\lib\GL\
```

4. Experimental Results

4.1 Program running

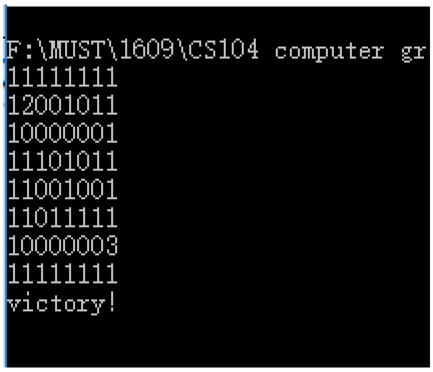


4.2 Collision detection



The player will be replaced to the last valid position

4.3 After going through the exit



```
F:\MUST\1609\CS104 computer gr
11111111
12001011
10000001
11101011
11001001
11011111
10000003
11111111
victory!
```

In the console there will be a line prompting “victory”.

5. Your feelings or opinions about this project.

During the development of this 3D interaction Maze, I was able to make use of almost all the key knowledge of OpenGL in this semester including the drawing of 3D objects, lighting and material as well as texture mapping. Coding is the easy part while understanding the whole framework actually took me a while. Building this game enables me to dive deeper into the rendering pipeline of OpenGL and clear the fog for me especially in the 3D object transformation. I hope that future study in the computer graphics will bring me more challenges and opportunities in my programming career.

**** END ****