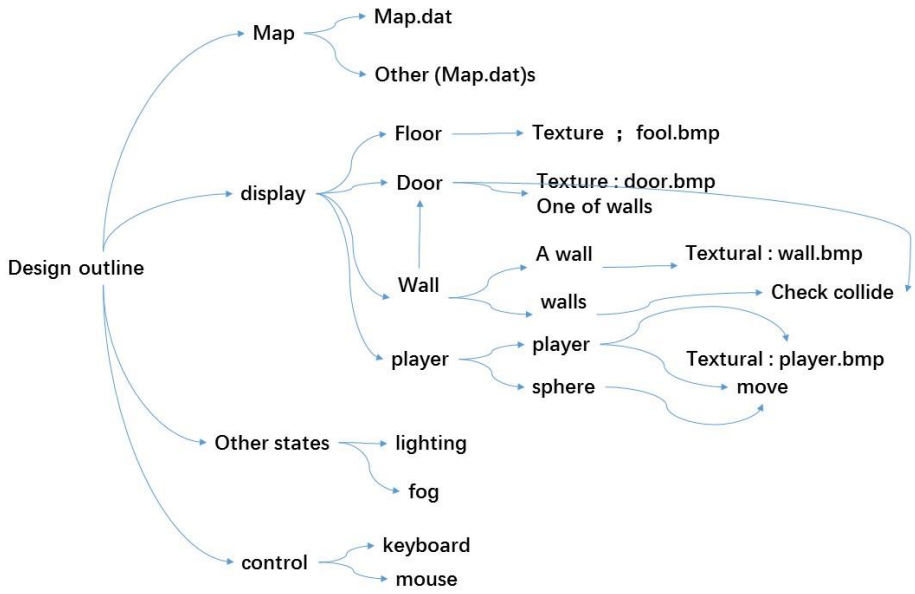
Report of An Interactive 3D Maze Game

Name: 吕沛颖

ID： 1409853G-I011-0186

Content：

a) Design outline



b) Key code fragments or algorithms of your program

1) floor part

```
void DrawGround()
{
    // Draw the ground here
    glPushMatrix();
        // covering the floor following the x-z  plane
        glTranslatef(_wallScale * _mapx / 2.0, 0.0, _wallScale * _mapz / 2.0);
        glScalef(_wallScale * _mapx, 1.0, _wallScale * _mapz);

        glColor3f(1.0, 1.0, 1.0);
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D, groundTextureID);

        glBegin(GL_QUADS);
        //setting the texture coordinates
        //point 1
        glTexCoord2f(0.0, 0.0);
        glVertex3f(-0.5, 0.0, 0.5);

        //point 2
        glTexCoord2f(_mapx, 0.0);
        glVertex3f(0.5, 0.0, 0.5);

        //point 3
        glTexCoord2f(_mapx, _mapz);
        glVertex3f(0.5, 0.0, -0.5);

        //point 4
        glTexCoord2f(0.0, _mapz);
        glVertex3f(-0.5, 0.0, -0.5);

        glEnd();
        glDisable(GL_TEXTURE_2D);
        glPopMatrix();
}
```

In the floor part, it just need to cover the x-z plane with texture following the coordinate.
Then, in the init() part.

```
    //ground
    glBindTexture(GL_TEXTURE_2D, groundTextureID);
    data = TextureLoadBitmap("texture//floor.bmp", &width, &height);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

The codes should be input to call it back.

2) the wall part

For convenience, I dispersed the adding texture part and set walls part.

Adding texture part, it seems like the floor part, which control the scale of texture and put the texture according to 4 verities.

```
}
void DrawWall(float cx, float cy, wallDir dir, GLuint textureID) {
    glBindTexture(GL_TEXTURE_2D, textureID);

    glPushMatrix();
    glScalef(1.0, 0.5, 1.0);
    glTranslatef(cx, 0.5*_wallScale, cy);
    glRotatef(dir*90.0, 0, 1, 0);

    glBegin(GL_QUADS);
    glTexCoord2f(0, 0);
    glVertex3f(-_wallScale*0.5, -_wallScale*0.5, -_wallScale*0.5);

    glTexCoord2f(0, 1);
    glVertex3f(-_wallScale*0.5, _wallScale*0.5, -_wallScale*0.5);

    glTexCoord2f(1, 1);
    glVertex3f(_wallScale*0.5, _wallScale*0.5, -_wallScale*0.5);

    glTexCoord2f(1, 0);
    glVertex3f(_wallScale*0.5, -_wallScale*0.5, -_wallScale*0.5);

    glEnd();
    glPopMatrix();

}
```

Then, in the set walls part.

I let the codes check the map at first. And the if the map spot is equal to “1”, the wall will be set here.

And the code can check whether the spot(i,j) is it on the map(_mapx, _mapz) by itself. If it meet the condition, it will do the above step.

```
void DrawWalls()
{
   // Draw the maze's walls here
    glEnable(GL_TEXTURE_2D);

    for (int i = 0; i < _mapz; ++i) {
        for (int j = 0; j < _mapx; ++j) {
            if (_map[i][j] % 2 != 0) {
                GLuint texture = _map[i][j] == 1 ? wallTextureID : doorTextureID;
                if (i - 1 >= 0 && !(_map[i - 1][j] % 2))        //north
                {
                    DrawWall((j + 0.5)*_wallScale, (i + 0.5)*_wallScale, NORTH, texture);
                }
                if (j + 1 < _mapx && !(_map[i][j + 1] % 2))     //east
                {
                    DrawWall((j + 0.5)*_wallScale, (i + 0.5)*_wallScale, EAST, texture);
                }
                if (i + 1 < _mapx && !(_map[i + 1][j] % 2))     //south
                {
                    DrawWall((j + 0.5)*_wallScale, (i + 0.5)*_wallScale, SOUTH, texture);
                }
                if (j - 1 >= 0 && !(_map[i][j - 1] % 2))        //west
                {
                    DrawWall((j + 0.5)*_wallScale, (i + 0.5)*_wallScale, WEST, texture);
                }
            }
        }
    }
    glDisable(GL_TEXTURE_2D);
}
```

In the init(), it should be called back also.

```
//wall
glBindTexture(GL_TEXTURE_2D, wallTextureID);
data = TextureLoadBitmap("texture//wall.bmp", &width, &height);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

3) the door part

In my mind, the door is one of the wall and it just has different texture.

It means if the spot is 3 here, the door texture will be set.

```
GLuint texture = _map[i][j] == 1 ? wallTextureID : doorTextureID;
```

And the same step in ini():

```
//door
glBindTexture(GL_TEXTURE_2D, doorTextureID);
data = TextureLoadBitmap("texture//door.bmp", &width, &height);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

4) the collide part

```
void checkcollide()
{
    //auto variable
    static auto timeStamp = clock() / 1000.0f;
    if (_player.forward == 0.0)
    {
        timeStamp = clock() / 1000.0f;
        return;
    }

    float dx, dz;
    // Check collision of walls here //触碰

    // Update the current position
    auto nowTime = clock() / 1000.0f;

    dx = _player.forward *(nowTime - timeStamp)* sin(_player.degree * M_PI / 180.0);
    dz = _player.forward *(nowTime - timeStamp)* cos(_player.degree * M_PI / 180.0);

    _player.rotate += (360 / (2 * M_PI * _player.pos[1] / _player.forward)) * (nowTime - timeStamp);

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

    int i = _player.pos[2] / _wallScale;
    int j = _player.pos[0] / _wallScale;

    if (_map[i][j] == 3) {
        MessageBox(NULL, "Arrival terminal", "congratulations", MB_OK);
        initplayer();
    }

    //check north
    if (_player.pos[2] - _player.pos[1]<i*_wallScale&&_map[i - 1][j] == 1){
        _player.pos[2] = i*_wallScale + _player.pos[1];
    }
    //check east
    if (_player.pos[0] + _player.pos[1]>(j + 1)*_wallScale&&_map[i][j + 1] == 1){
        _player.pos[0] = (j + 1)*_wallScale - _player.pos[1];
    }
    //check south
    if (_player.pos[2] + _player.pos[1]>(i + 1)*_wallScale&&_map[i + 1][j] == 1){
        _player.pos[2] = (i + 1)*_wallScale - _player.pos[1];
    }
    //check west
    if (_player.pos[0] - _player.pos[1]<j*_wallScale&&_map[i][j - 1] == 1){
        _player.pos[0] = j*_wallScale + _player.pos[1];
    }
```

Let's see the middle codes.

It is a code fragments to show a result window when player arrive the door.

```
if (_map[i][j] == 3) {
    MessageBox(NULL, "Arrival terminal", "congratulations", MB_OK);
    initplayer();
}
```

With the thinking of the collision may appear while the player is close to walls, the collide-checking computes the relationship between the radius of sphere and walls.

According to the formulas, the code can check whether the palyer is touching the wall or not.

And the four directions have been named as NORTH, EAST, SOUTH and WEST.

```
//check north
if (_player.pos[2] - _player.pos[1]<i*_wallScale&&_map[i - 1][j] == 1){
    _player.pos[2] = i*_wallScale + _player.pos[1];
}
//check east
if (_player.pos[0] + _player.pos[1]>(j + 1)*_wallScale&&_map[i][j + 1] == 1){
    _player.pos[0] = (j + 1)*_wallScale - _player.pos[1];
}
//check south
if (_player.pos[2] + _player.pos[1]>(i + 1)*_wallScale&&_map[i + 1][j] == 1){
```

```
        _player.pos[2] = (i + 1)*_wallScale - _player.pos[1];
    }
    //check west
    if (_player.pos[0] - _player.pos[1]<j*_wallScale&&_map[i][j - 1] == 1){
        _player.pos[0] = j*_wallScale + _player.pos[1];
    }
```

Since the maze is formed by grids, your object will only hit any wall if it is intersecting any grid line(s), so now our first problem is to check whether there is intersection of the sphere with the grid lines.

The equation for sphere:

$$(x - C_x)^2 + (y - C_y)^2 = r^2$$

to check if it intersects the surrounding 4 grid lines

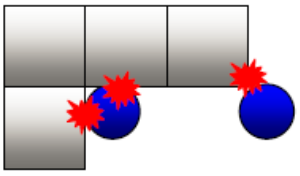
$$x = G_x,\ x = G_x + \text{wallScale}$$

$$y = G_y,\ y = G_y + \text{wallScale}$$

we can substitute a value to it , for example to check whether it intersect $x = G_x$.

$$(G_x - C_x)^2 + (y - C_y)^2 = r^2$$

$$y^2 - 2\ C_y * y + C_y^2 + (G_x - C_y)^2 - r^2 = 0$$

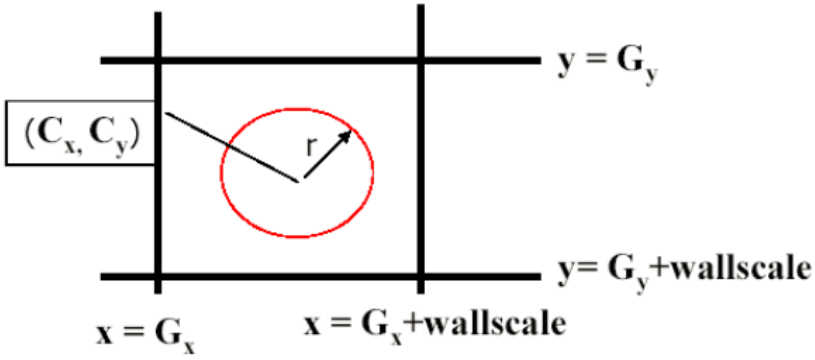


For this equation to have solution, we can calculate its determinant

$$\text{Det} = 4\ C_y^2 - 4 * (C_y^2 + (G_x - C_x)^2 - r^2)$$

$$\text{Det} = 4 * ((G_x - C_x)^2 - r^2)$$

If Det >= 0, it means intersection occurs, otherwise, it doesn't.



5)the material part and DrawPlayer()

This part appeared in DrawPlayer().

At the same time, the DrawPlayer() ensure the start position of player and the texture as well as the shape.

```
// Draw your player here
 GLfloat a_ambient[] = { 0.0,0.0,1.0,1.0 };  //blue
 GLfloat a_diffuse[] = { 0.0,0.0,1.0,1.0 };   //blue
 GLfloat a_shininess[] = { 10 };


 glEnable(GL_TEXTURE_2D);
 glBindTexture(GL_TEXTURE_2D, playerTextureID);
 glPushMatrix();

 //drew the plater and set it into this position
 glTranslatef(_player.pos[0], _player.pos[1], _player.pos[2]);
 glScalef(_player.pos[1], _player.pos[1], _player.pos[1]);
 glRotatef(_player.degree, 0, 1, 0);
 glRotatef(_player.rotate, 1, 0, 0);

 glBegin(GL_TRIANGLE_STRIP); //三角网

 //materica state
 glMaterialfv(GL_FRONT, GL_AMBIENT, a_ambient);
 glMaterialfv(GL_FRONT, GL_DIFFUSE, a_diffuse);
 glMaterialfv(GL_FRONT, GL_SHININESS, a_shininess);
```

6) the fog and lighting part(in init())

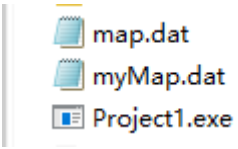
```
  //============fog================================
  float fogColor[] = { 1.0, 0.0, 0.0, 1.0 };
  glFogi(GL_FOG_MODE, GL_LINEAR);       //kind of fog
  glFogfv(GL_FOG_COLOR, fogColor);      //color
  glFogf(GL_FOG_DENSITY, 0.05f);        //density
  glHint(GL_FOG_HINT, GL_DONT_CARE);    //rendering method
  glFogf(GL_FOG_START, 1.0f);
  glFogf(GL_FOG_END, 15.0f);
  glEnable(GL_FOG);
```

```
   //============lighting=====================
   GLfloat light1_pos[] = { 10.0,10.0,-10.0,1.0 };
   GLfloat light1_dif[] = { 1.0,1.0,1.0, }; //whlit
   GLfloat light1_spe[] = { 0.0,0.0,1.0,1.0 };   //blue
   glLightfv(GL_LIGHT1, GL_POSITION, light1_pos);
   glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_dif);
   glLightfv(GL_LIGHT1, GL_SPECULAR, light1_spe);
   glEnable(GL_LIGHT1);
```

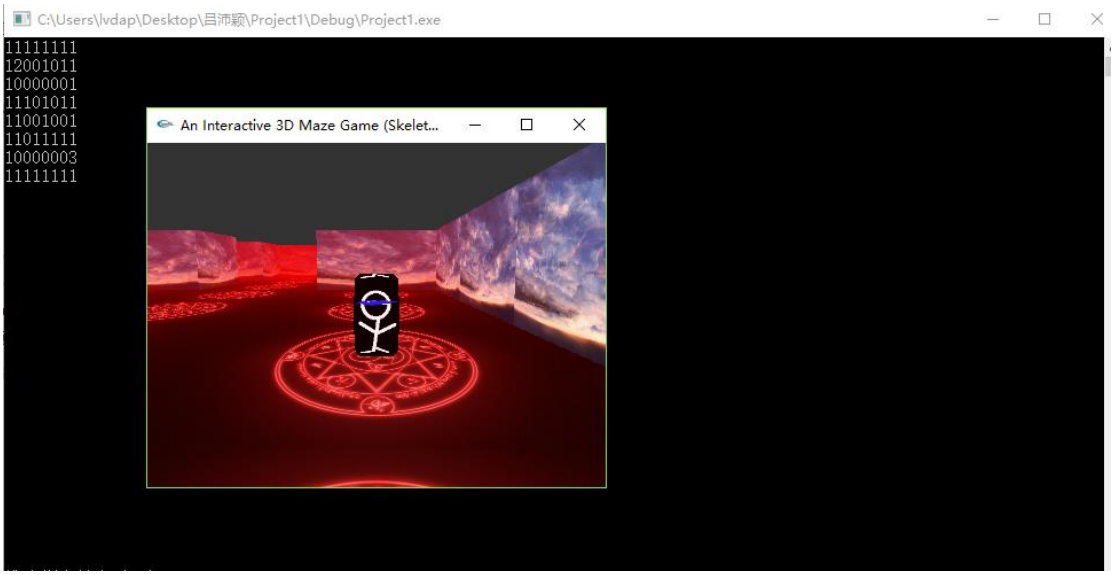
With the feature of the fog and lighting, the floor will show different degrees of shining.

c) user manual

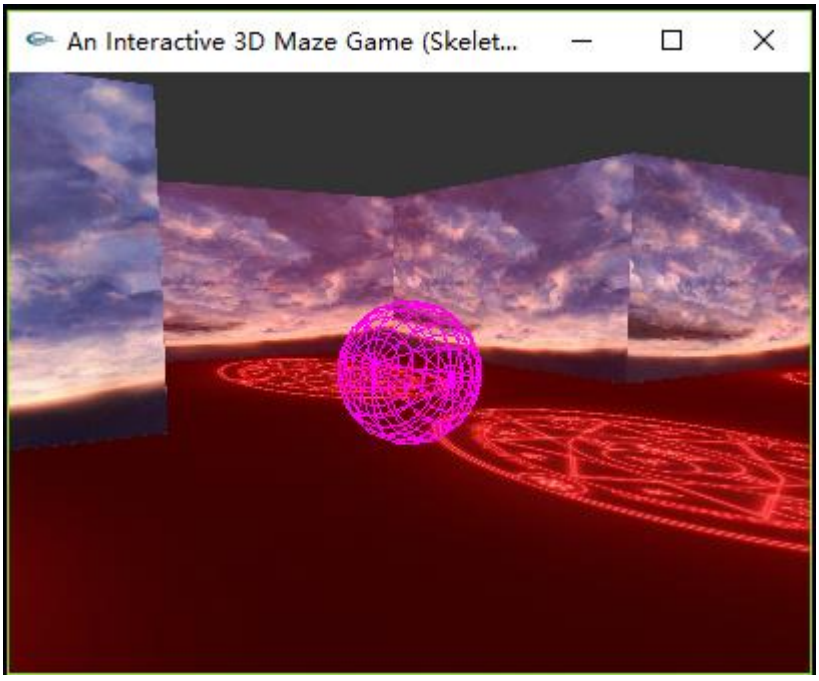
open the “debug” folder,
and then, pull the “.dat” file pull into the “.exe” file



Then, the page will appear.



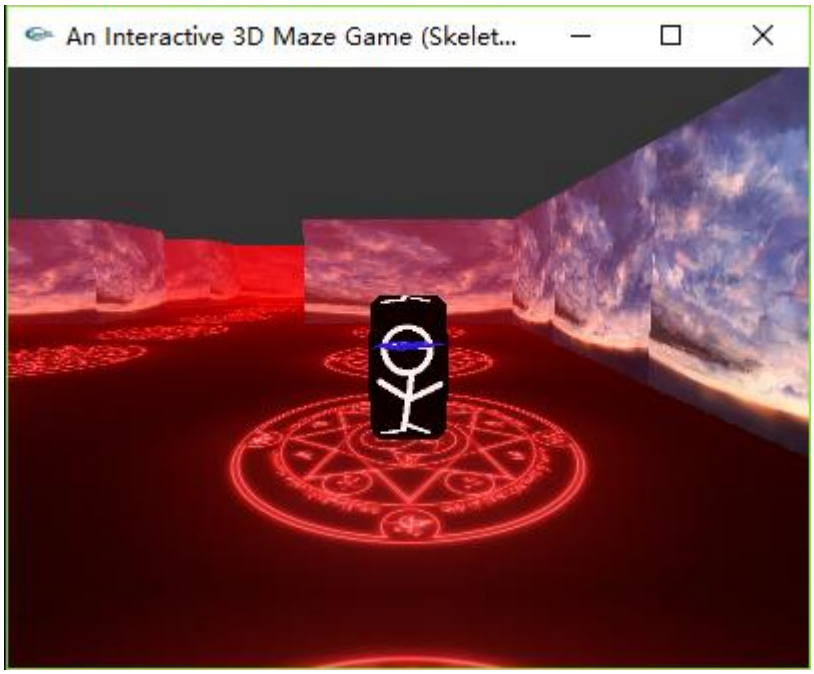
If you do not like the appearance of player, you can push the 's' to change.



If you want to go straight, you should hit both left and right mouse buttons.
If you want to turn left, you can just hit the left button, and If you want to turn right, you have to hit the right button.

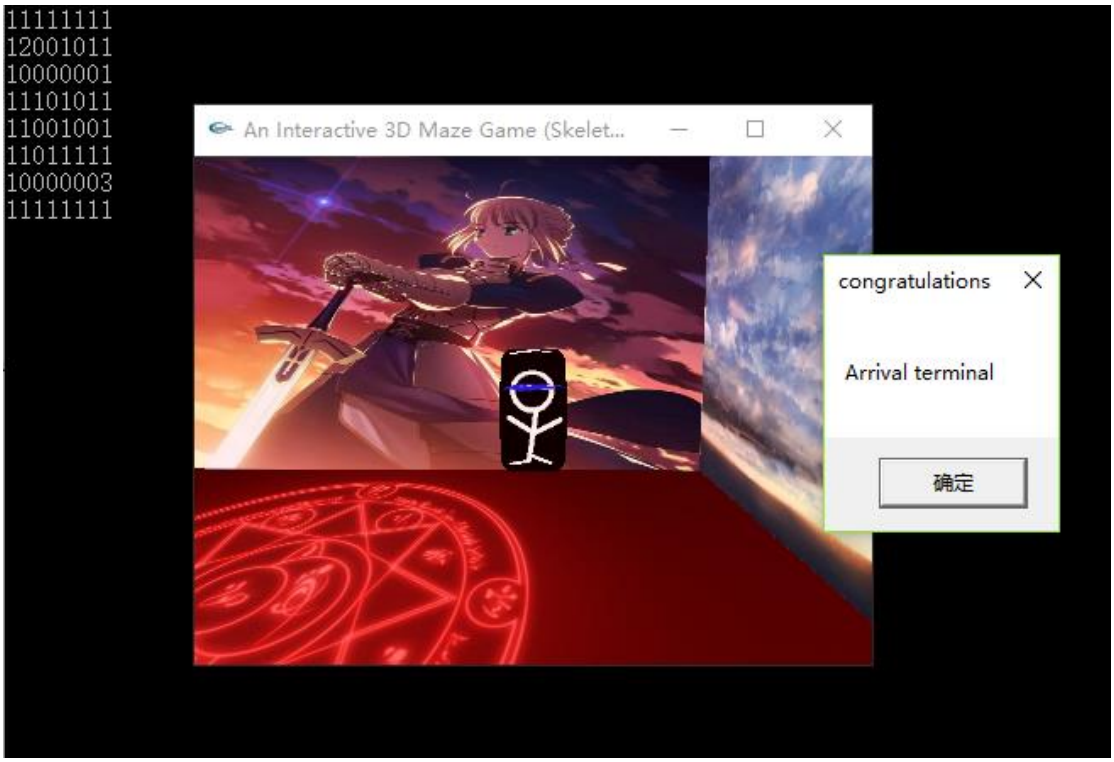
d) Experimental Results

The position which you will start at.



The position which you are at the corner.

The position which you win.
Touching the win wall.



e) Feeling and opinions

Feeling:

It is a funny game design.

In this project, the knowledge in class can be applied as a global. And I also can open mind to add something I like here.

However, there are some points I haven't know before, I have to search the method and scan the others code in internet which made me know more.

In my opinion, no player visual sense is more cool to play a maze.

Opinion:

Using mouse to change the view is inconvenience, it is a challenge to change it by moving the mouse which make the game more like a real game.

And some traps can be added, such as when touching a trap the player have to restart.