Computer graphics and multimedia
UDL MINF 20-21

Universitat de Lleida

Package 4

Danillo Lange Souza Borges dos Santos

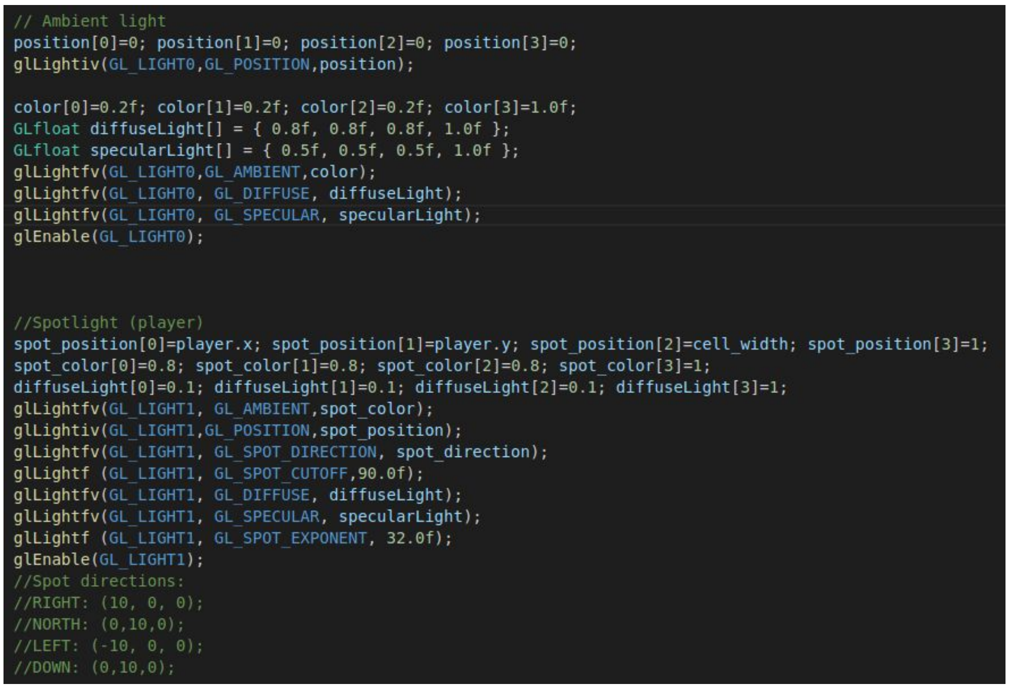
Bugs fixed:

Task 1. Inclusion of lighting effects to the game

Requirements:

● Include a light source providing ambient light. The intensity of this light source has to be low. This light source is required for avoiding that polygons not lighted by some other light source appear completely black.

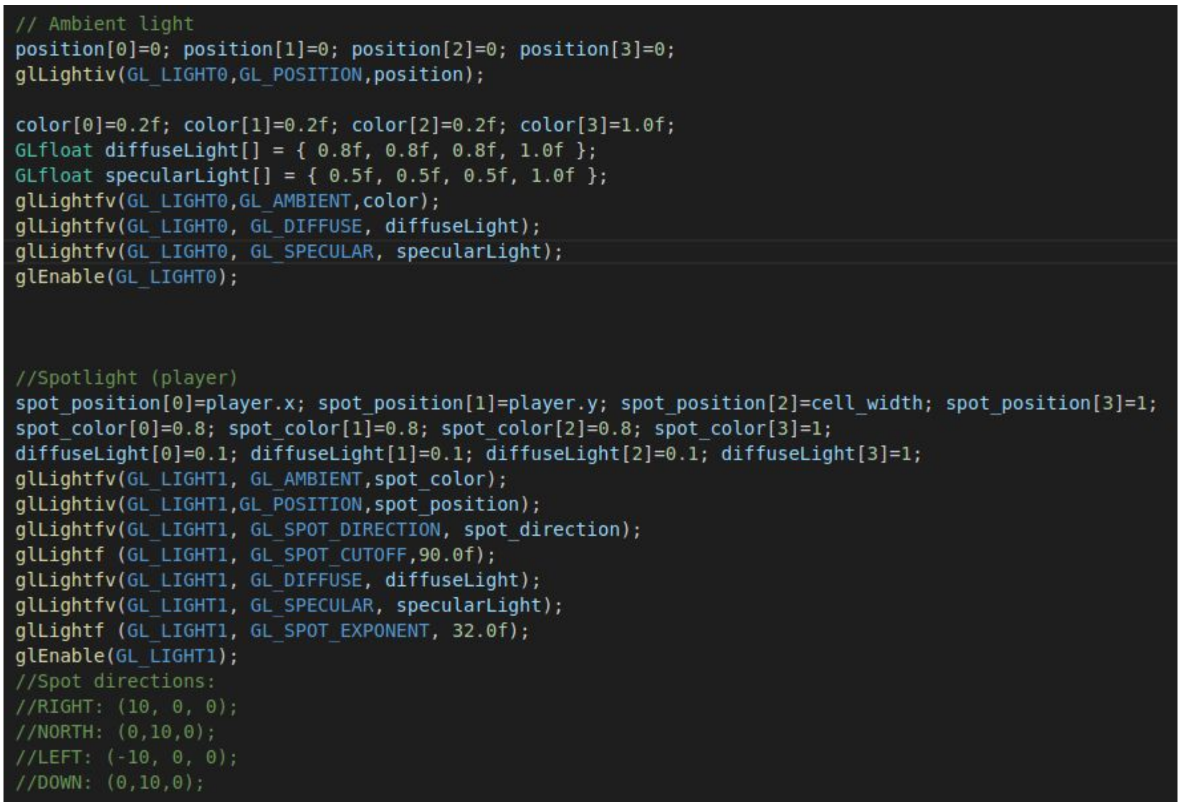
The ambient light source has been implemented in the display function, it works in a way that all objects receive equal low light intensity, to avoid them being completely black.



```
// Ambient light
position[0]=0; position[1]=0; position[2]=0; position[3]=0;
glLightiv(GL_LIGHT0,GL_POSITION,position);

color[0]=0.2f; color[1]=0.2f; color[2]=0.2f; color[3]=1.0f;
GLfloat diffuseLight[] = { 0.8f, 0.8f, 0.8f, 1.0f };
GLfloat specularLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };
glLightfv(GL_LIGHT0,GL_AMBIENT,color);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight);
glEnable(GL_LIGHT0);



//Spotlight (player)
spot_position[0]=player.x; spot_position[1]=player.y; spot_position[2]=cell_width; spot_position[3]=1;
spot_color[0]=0.8; spot_color[1]=0.8; spot_color[2]=0.8; spot_color[3]=1;
diffuseLight[0]=0.1; diffuseLight[1]=0.1; diffuseLight[2]=0.1; diffuseLight[3]=1;
glLightfv(GL_LIGHT1, GL_AMBIENT,spot_color);
glLightiv(GL_LIGHT1,GL_POSITION,spot_position);
glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, spot_direction);
glLightf (GL_LIGHT1, GL_SPOT_CUTOFF,90.0f);
glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuseLight);
glLightfv(GL_LIGHT1, GL_SPECULAR, specularLight);
glLightf (GL_LIGHT1, GL_SPOT_EXPONENT, 32.0f);
glEnable(GL_LIGHT1);
//Spot directions:
//RIGHT: (10, 0, 0);
//NORTH: (0,10,0);
//LEFT: (-10, 0, 0);
//DOWN: (0,10,0);
```

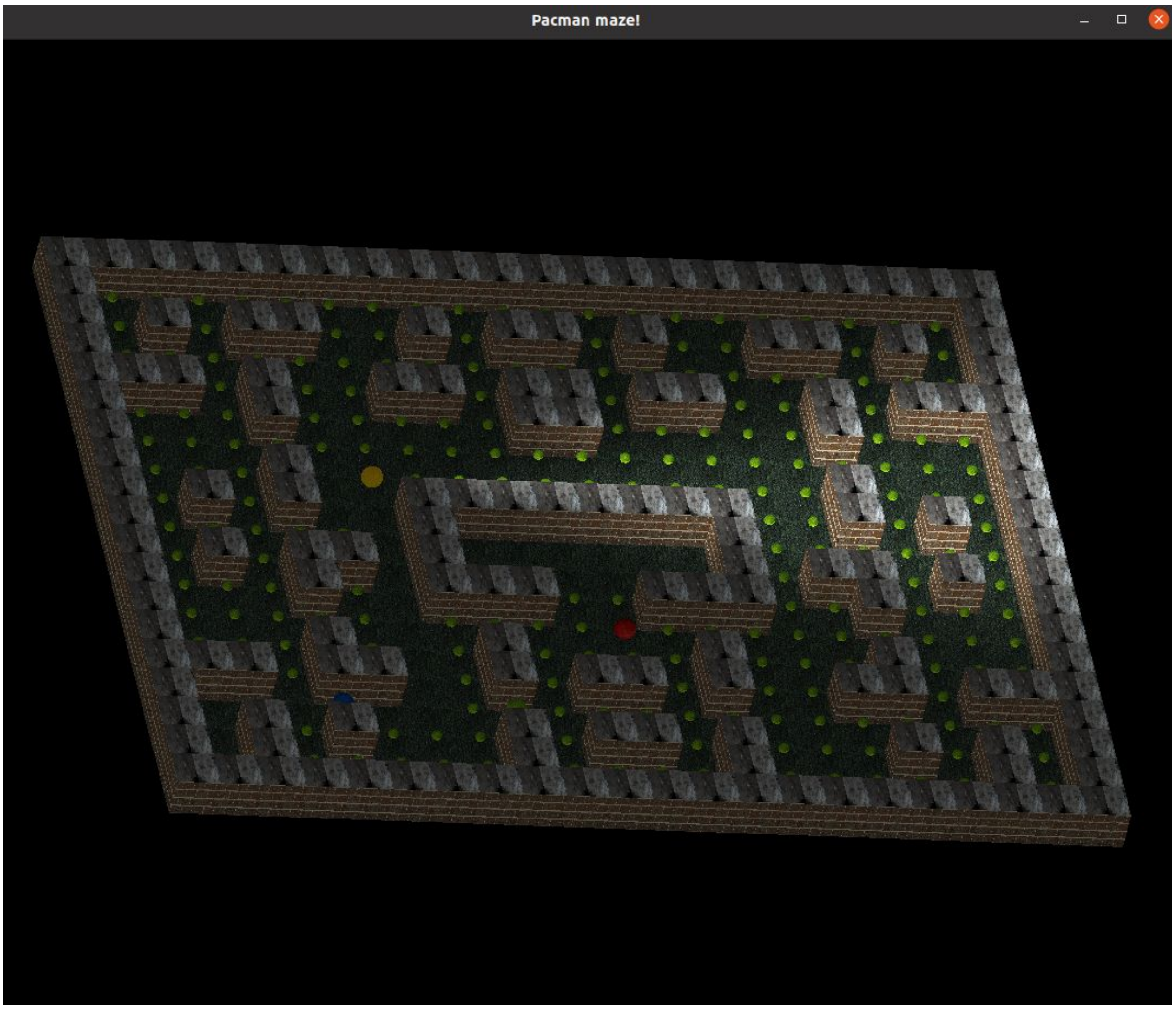
- Include a directional light source in front of the main character. In this way we will simulate it carrying a flashlight.

The direction light has been implemented, its position is the same as the player's so it follows it simulating a flashlight, the direction of the light is decided by the player's movement, when they go right the light points right, for example.



```
// Ambient light
position[0]=0; position[1]=0; position[2]=0; position[3]=0;
glLightiv(GL_LIGHT0,GL_POSITION,position);

color[0]=0.2f; color[1]=0.2f; color[2]=0.2f; color[3]=1.0f;
GLfloat diffuseLight[] = { 0.8f, 0.8f, 0.8f, 1.0f };
GLfloat specularLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };
glLightfv(GL_LIGHT0,GL_AMBIENT,color);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight);
glEnable(GL_LIGHT0);



//Spotlight (player)
spot_position[0]=player.x; spot_position[1]=player.y; spot_position[2]=cell_width; spot_position[3]=1;
spot_color[0]=0.8; spot_color[1]=0.8; spot_color[2]=0.8; spot_color[3]=1;
diffuseLight[0]=0.1; diffuseLight[1]=0.1; diffuseLight[2]=0.1; diffuseLight[3]=1;
glLightfv(GL_LIGHT1, GL_AMBIENT,spot_color);
glLightiv(GL_LIGHT1,GL_POSITION,spot_position);
glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, spot_direction);
glLightf (GL_LIGHT1, GL_SPOT_CUTOFF,90.0f);
glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuseLight);
glLightfv(GL_LIGHT1, GL_SPECULAR, specularLight);
glLightf (GL_LIGHT1, GL_SPOT_EXPONENT, 32.0f);
glEnable(GL_LIGHT1);
//Spot directions:
//RIGHT: (10, 0, 0);
//NORTH: (0,10,0);
//LEFT: (-10, 0, 0);
//DOWN: (0,10,0);
```

The spot_direction is updated on the keyboardFunc()

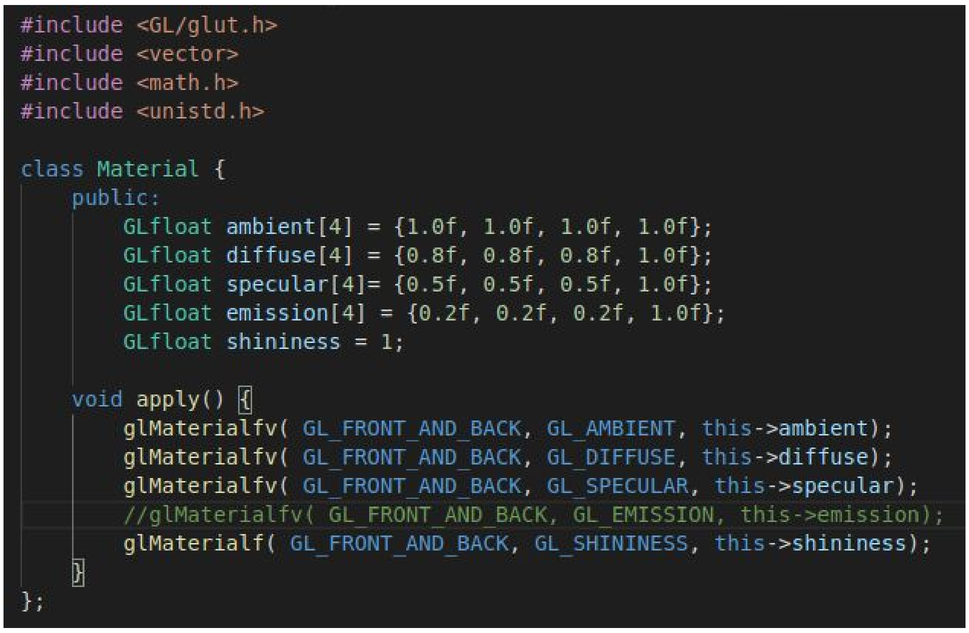


● When drawing textured polygons in lighting mode, set the material (procedure “glMaterialfv”) to (1.0,1.0,1.0,1.0).

We apply the materials properties for all the objects

```
material_1.apply();
```

The materials attributes and functions are located in file Material.cpp

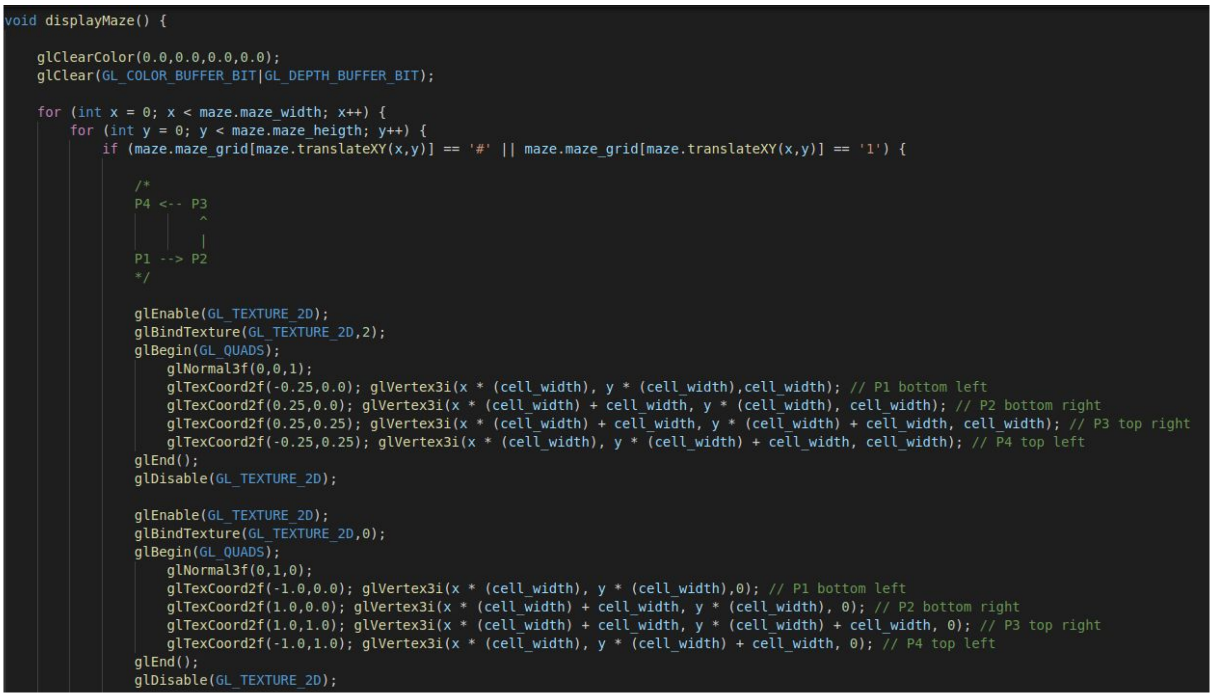


```cpp
#include <GL/glut.h>
#include <vector>
#include <math.h>
#include <unistd.h>

class Material {
    public:
        GLfloat ambient[4] = {1.0f, 1.0f, 1.0f, 1.0f};
        GLfloat diffuse[4] = {0.8f, 0.8f, 0.8f, 1.0f};
        GLfloat specular[4]= {0.5f, 0.5f, 0.5f, 1.0f};
        GLfloat emission[4] = {0.2f, 0.2f, 0.2f, 1.0f};
        GLfloat shininess = 1;

    void apply() {
        glMaterialfv( GL_FRONT_AND_BACK, GL_AMBIENT, this->ambient);
        glMaterialfv( GL_FRONT_AND_BACK, GL_DIFFUSE, this->diffuse);
        glMaterialfv( GL_FRONT_AND_BACK, GL_SPECULAR, this->specular);
        //glMaterialfv( GL_FRONT_AND_BACK, GL_EMISSION, this->emission);
        glMaterialf( GL_FRONT_AND_BACK, GL_SHININESS, this->shininess);
    }
};
```

- When implementing lighting effects, remember to carefully set the normal vectors at polygon vertices.

All normal vertices have been calculated, for the spheres the calculus is on the Sphere.cpp, and for the remaining objects it is located on the function displayMaze():

The next image shows an example of the normal vector definition for 2 faces of a cube:



```cpp
void displayMaze() {

    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

    for (int x = 0; x < maze.maze_width; x++) {
        for (int y = 0; y < maze.maze_heigth; y++) {
            if (maze.maze_grid[maze.translateXY(x,y)] == '#' || maze.maze_grid[maze.translateXY(x,y)] == '1') {

                /*
                P4 <-- P3
                        ^
                        |
                P1 --> P2
                */

                glEnable(GL_TEXTURE_2D);
                glBindTexture(GL_TEXTURE_2D,2);
                glBegin(GL_QUADS);
                    glNormal3f(0,0,1);
                    glTexCoord2f(-0.25,0.0); glVertex3i(x * (cell_width), y * (cell_width),cell_width); // P1 bottom left
                    glTexCoord2f(0.25,0.0); glVertex3i(x * (cell_width) + cell_width, y * (cell_width), cell_width); // P2 bottom right
                    glTexCoord2f(0.25,0.25); glVertex3i(x * (cell_width) + cell_width, y * (cell_width) + cell_width, cell_width); // P3 top right
                    glTexCoord2f(-0.25,0.25); glVertex3i(x * (cell_width), y * (cell_width) + cell_width, cell_width); // P4 top left
                glEnd();
                glDisable(GL_TEXTURE_2D);

                glEnable(GL_TEXTURE_2D);
                glBindTexture(GL_TEXTURE_2D,0);
                glBegin(GL_QUADS);
                    glNormal3f(0,1,0);
                    glTexCoord2f(-1.0,0.0); glVertex3i(x * (cell_width), y * (cell_width),0); // P1 bottom left
                    glTexCoord2f(1.0,0.0); glVertex3i(x * (cell_width) + cell_width, y * (cell_width), 0); // P2 bottom right
                    glTexCoord2f(1.0,1.0); glVertex3i(x * (cell_width) + cell_width, y * (cell_width) + cell_width, 0); // P3 top right
                    glTexCoord2f(-1.0,1.0); glVertex3i(x * (cell_width), y * (cell_width) + cell_width, 0); // P4 top left
                glEnd();
                glDisable(GL_TEXTURE_2D);
```