Computer Graphics

(UCS505)

Project on

Maze Munchers

Submitted By

Cheshta Biala	**102103545**
Yatharth Gautam	**102103550**
Arshiya Kishore	**102103565**

3CO20

B.E. Third Year – COE
Submitted To:

Dr. Anupam Garg

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala

Table of Contents

Introduction to Project

Inspired by the classic Pacman arcade game, **'Maze Munchers'** is a modern rendition that combines nostalgic gameplay with contemporary graphics and mechanics. The project aims to recreate the immersive experience of the original game while introducing new elements to enhance gameplay and engagement. Utilising OpenGL for graphics rendering and GLUT for user interaction, 'Maze Munchers presents players with a series of mazes populated by pellets and ghosts. The player, controlling a character known as a 'muncher,' navigates the maze, consuming pellets to score points while evading the ghosts, which seek to impede progress. Key features of **Maze Munchers** include intuitive controls and strategic use of power-ups to enhance gameplay complexity. The game's design emphasises entertainment and cognitive engagement, requiring players to devise strategies to maximise their score while avoiding ghost encounters. **Maze Munchers** offers a compelling gaming experience that appeals to a wide audience through its blend of classic gameplay and modern design principles. The project showcases the application of OpenGL and GLUT in developing interactive and visually appealing games, demonstrating their potential for creating immersive gaming experiences.

About the Game: Maze Munchers

Inspiration: Maze Munchers is inspired by the classic arcade game Pac-Man, known for its simple yet addictive gameplay. The project seeks to recreate the nostalgic experience of the original game while adding modern graphics and mechanics to enhance player engagement.

Objective: The goal of Maze Munchers is to navigate through a series of mazes, collecting all the pellets while avoiding contact with the ghosts. Each maze presents a new challenge, requiring players to use strategy and quick reflexes to succeed.

Controls: Players use arrow keys or other specified controls to move their character, known as a "muncher," through the maze. The intuitive controls allow for precise movement, essential for avoiding ghosts and navigating the maze efficiently.

Game Elements:

- Pellets: Scattered throughout the maze are pellets that players must collect to advance. These pellets serve as the primary objective and are essential for scoring points.
- Ghosts: Ghosts roam the maze, seeking to intercept the player. If a ghost catches the player, a life is lost. Players must outmaneuver the ghosts using cunning and agility.
- Power-ups: Some levels may contain power-ups that grant temporary abilities, such as speed boosts or invincibility, enhancing gameplay and strategic options.
- Lives and Score: Players start with a set number of lives and earn points by collecting pellets. Bonus points may be awarded for completing levels quickly or collecting all pellets without losing a life.
- Game Over: The game ends when all lives are lost. However, players can continue playing, offering endless opportunities to improve their skills and achieve higher scores.

Graphics and Mechanics:

Maze Munchers utilizes OpenGL for graphics rendering, creating a visually appealing and immersive gaming experience. The game's dynamic maze generation ensures that each playthrough is unique, keeping players engaged and challenged.

Audience Appeal:

With its blend of classic gameplay and modern design, Maze Munchers appeals to a wide audience. Fans of the original Pac-Man will appreciate the nostalgia, while new players will enjoy the game's updated graphics and challenging gameplay.

Conclusion:

Maze Munchers is not just a recreation of a classic game but a reimagining that brings new life to an iconic concept. Through its intuitive controls, engaging gameplay, and dynamic design, Maze Munchers offers a compelling gaming experience that is both familiar and fresh.

Computer Graphics Concepts Used

Sr. No.	Computer Graphics Concepts	Description
1.	Texture Mapping	Texture mapping is used to apply textures to the surfaces of objects in the game. Textures are images that add detail and realism to the game's graphics. Maze Munchers uses texture mapping to render the maze walls, pellets, ghosts, and other elements with realistic and visually appealing textures.
2.	Lighting and Shading:	Complex geometric and quadratic shapes such as Spherical shapes.Lighting and shading techniques are used to enhance the visual appearance of the game. Maze Munchers uses lighting to simulate light sources in the game environment, creating shadows and highlights to make objects appear more realistic. Shading techniques such as Phong shading may be used to achieve smooth lighting transitions on surfaces.
3.	Animation	Animation is used to bring the game world to life, making objects move and interact with each other. Maze Munchers uses animation techniques to animate the player's character, ghosts, and other elements in the game, creating a dynamic and engaging gameplay experience.
4.	Physics Simulation	Used to handle collisions between objects, such as the player's character and walls or pellets, ensuring realistic interactions and gameplay mechanics.
5.	User Interaction	This involves allowing the user to interact with the screen by responding to user inputs such as mouse clicks, key presses, or touch events.
6.	Transformations	Translation: Used to move objects in the game world, such as the player's character and ghosts, allowing them to navigate the maze. Rotation: Enables rotating objects, which could be used for animating the movement of characters or objects in the game.

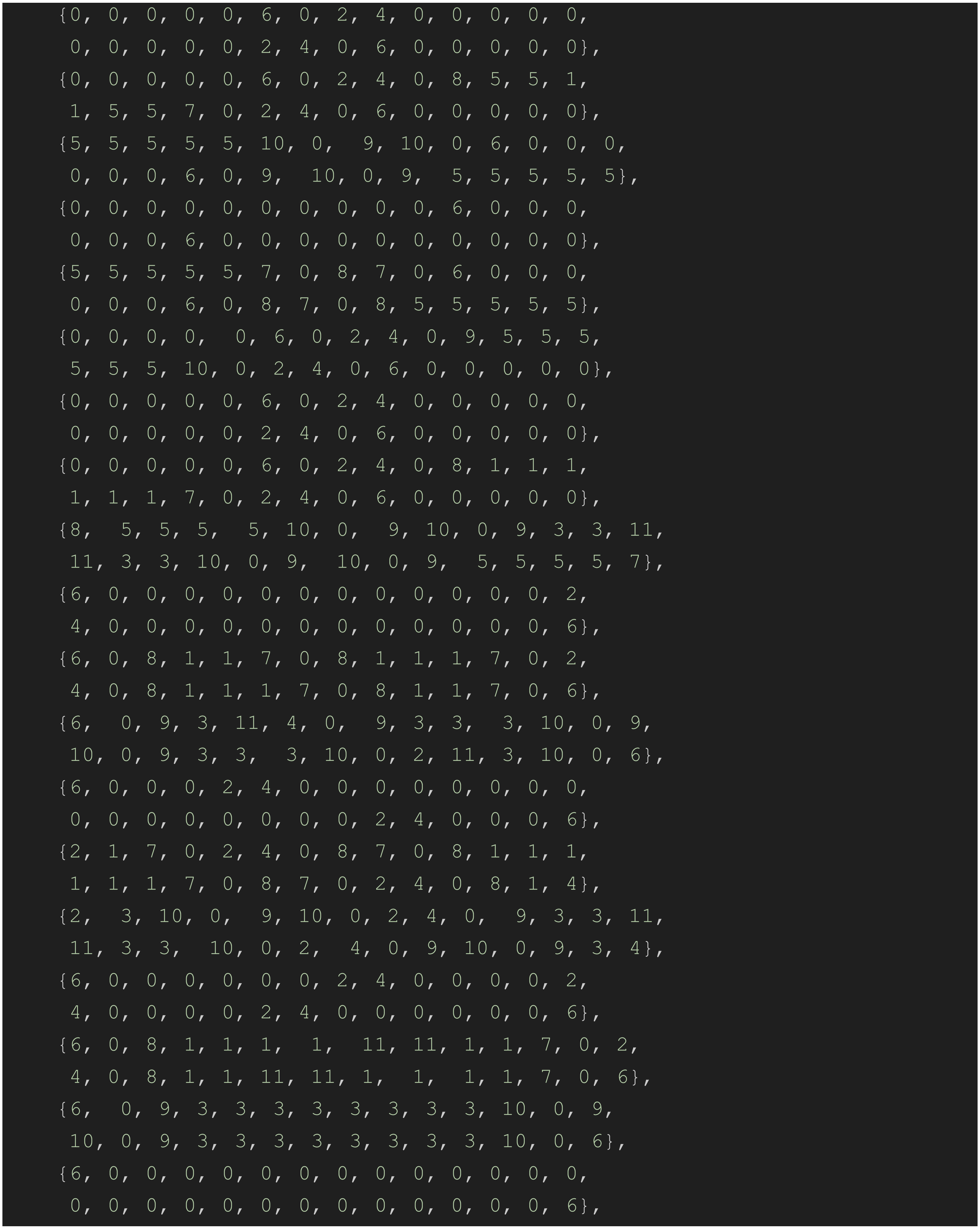
7.	Rendering Techniques	Sprite Rendering: Used for rendering 2D elements, such as the pellets and power-ups, as flat images facing the camera. Particle Systems: Could be used for rendering special effects, such as ghost movement trails or pellet animations, by simulating particles with specific behaviors.
8.	Text Rendering	Bitmap Fonts: Used for rendering text elements, such as the player's score or level information, as 2D images. Vector Fonts: Could be used for scalable and high-quality text rendering, ensuring readability and visual appeal.
9.	Projection	Orthographic Projection: Used for rendering the 2D maze layout, providing a top-down view that simplifies gameplay and navigation. Perspective Projection: Could be used for rendering 3D elements in the game, such as the player's character or the maze walls, to create a sense of depth.
10.	OpenGL Graphics Library	Maze Munchers utilizes the OpenGL graphics library for rendering graphics. OpenGL provides a set of functions for rendering 2D and 3D graphics, making it suitable for developing games with complex visual effects.

User Defined Functions

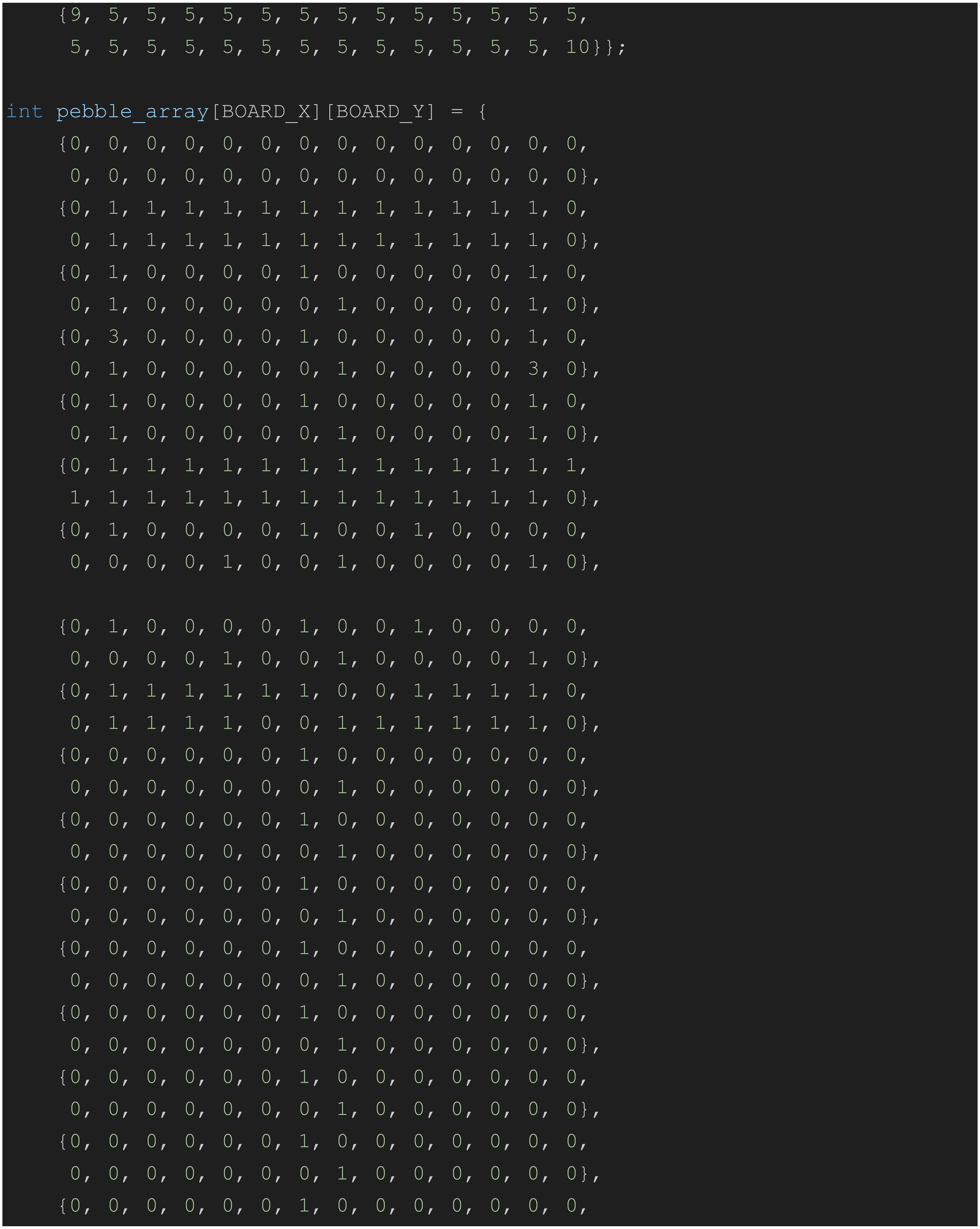
User Defined Functions	Description
Move()	Updates the position of Pacman based on its speed and angle.
bool Open()	Check if the specified position on the game board is open for movement.
Ghost	The constructor will initialize a Ghost object with specified coordinates.
~Ghost	Destructor to release resources allocated to a Ghost object.
Reinit()	Reinitializes the state of a Ghost object.
Update()	Updates the state of a Ghost object based on its position and other parameters.
Chase ()	Directs a Ghost object to chase Pacman based on its current position and direction.
Catch()	Check if a Ghost object has caught Pacman based on their positions.
Draw()	Draws a Ghost object on the screen.
Pac()	Draws the Pacman on the screen.
RenderScene()	This is the default display function. Here, the collision detection for Pacman and the conditions for Normal & super pebbles consumption, with monster movements, are covered. Options are provided for game control.
create_list_lib()	This function is used to create the basic primitive walls using display lists. Based on the position, the appropriate list is called.

Source Code

```
#include <GL/glut.h>
#include <ctype.h>
#include <math.h>
#include <stdio.h>
#include <string>
#define M_PI 3.14159265358979323846264338327950288419716939937510
#define false 0
#define true 1

const int BOARD_X = 31;
const int BOARD_Y = 28;

int board_array[BOARD_X][BOARD_Y] = {
    {8, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1,
     1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 7},
    {6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
     4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6},
    {6, 0, 8, 1, 1, 7, 0, 8, 1, 1, 1, 7, 0, 2,
     4, 0, 8, 1, 1, 1, 7, 0, 8, 1, 1, 7, 0, 6},
    {6, 0, 2, 11, 11, 4,  0, 2, 11, 11, 11, 4, 0, 2,
     4, 0, 2, 11, 11, 11, 4, 0, 2,  11, 11, 4, 0, 6},
    {6,  0, 9, 3, 3, 10, 0,  9, 3, 3, 3, 10, 0, 9,
     10, 0, 9, 3, 3, 3,  10, 0, 9, 3, 3, 10, 0, 6},
    {6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6},
    {6, 0, 8, 1, 1, 7, 0, 8, 7, 0, 8, 1, 1, 1,
     1, 1, 1, 7, 0, 8, 7, 0, 8, 1, 1, 7, 0, 6},
    {6,  0, 9, 3,  3, 10, 0, 2, 4, 0, 9, 3,  3, 11,
     11, 3, 3, 10, 0, 2,  4, 0, 9, 3, 3, 10, 0, 6},
    {6, 0, 0, 0, 0, 0, 0, 2, 4, 0, 0, 0, 0, 2,
     4, 0, 0, 0, 0, 2, 4, 0, 0, 0, 0, 0, 0, 6},
    {9, 5, 5, 5, 5, 7,  0, 2, 11, 1, 1, 7, 0, 2,
     4, 0, 8, 1, 1, 11, 4, 0, 8,  5, 5, 5, 5, 10},
    {0,  0, 0, 0, 0, 6,  0, 2, 11, 3, 3, 10, 0, 9,
     10, 0, 9, 3, 3, 11, 4, 0, 6,  0, 0, 0,  0, 0},
```

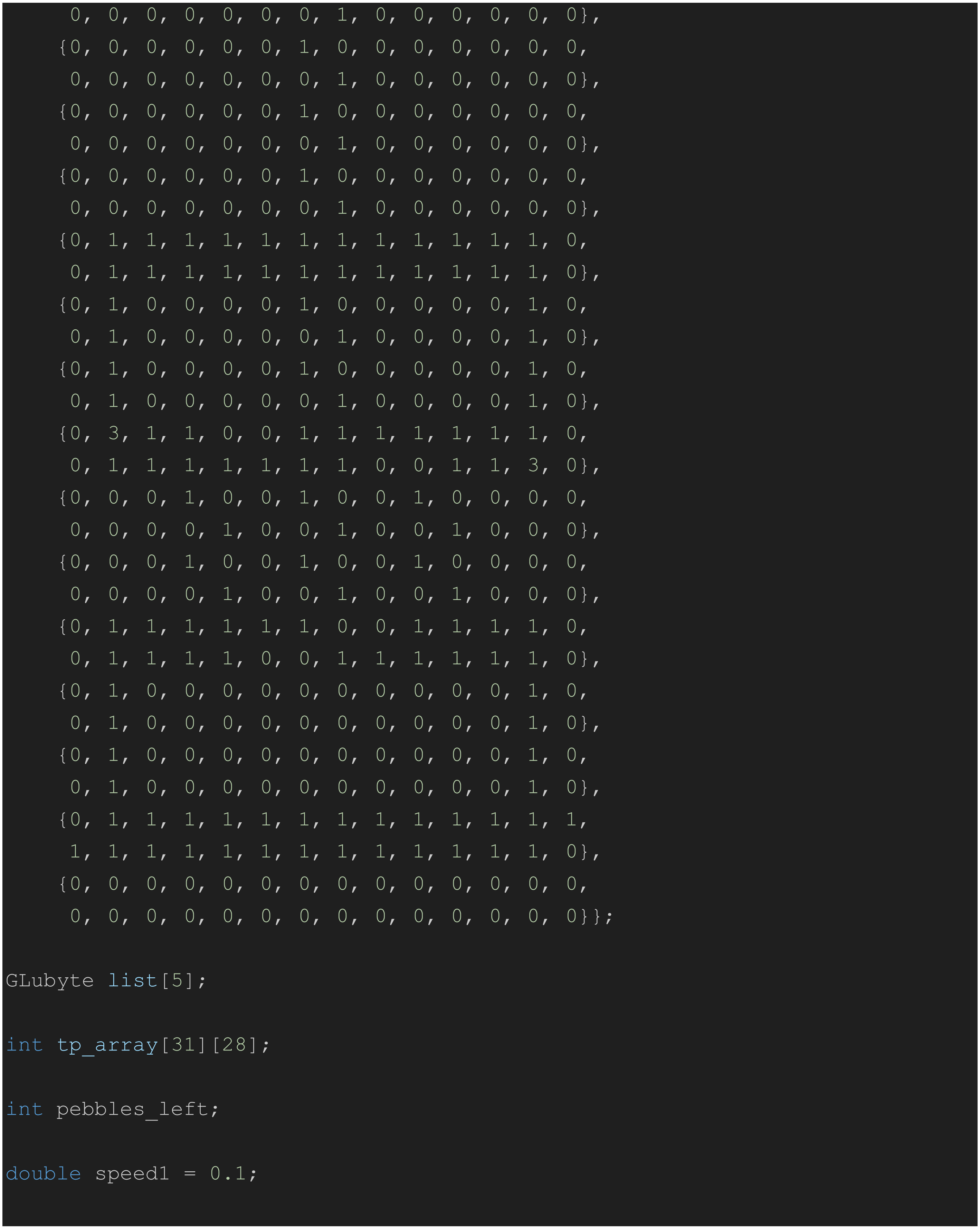


```
    {0, 0, 0, 0, 0, 6, 0, 2, 4, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 2, 4, 0, 6, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 6, 0, 2, 4, 0, 8, 5, 5, 1,
     1, 5, 5, 7, 0, 2, 4, 0, 6, 0, 0, 0, 0, 0},
    {5, 5, 5, 5, 5, 10, 0,  9, 10, 0, 6, 0, 0, 0,
     0, 0, 0, 6, 0, 9,  10, 0, 9,  5, 5, 5, 5, 5},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0,
     0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {5, 5, 5, 5, 5, 7, 0, 8, 7, 0, 6, 0, 0, 0,
     0, 0, 0, 6, 0, 8, 7, 0, 8, 5, 5, 5, 5, 5},
    {0, 0, 0, 0,  0, 6, 0, 2, 4, 0, 9, 5, 5, 5,
     5, 5, 5, 10, 0, 2, 4, 0, 6, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 6, 0, 2, 4, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 2, 4, 0, 6, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 6, 0, 2, 4, 0, 8, 1, 1, 1,
     1, 1, 1, 7, 0, 2, 4, 0, 6, 0, 0, 0, 0, 0},
    {8,  5, 5, 5,  5, 10, 0,  9, 10, 0, 9, 3, 3, 11,
     11, 3, 3, 10, 0, 9,  10, 0, 9,  5, 5, 5, 5, 7},
    {6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
     4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6},
    {6, 0, 8, 1, 1, 7, 0, 8, 1, 1, 1, 7, 0, 2,
     4, 0, 8, 1, 1, 1, 7, 0, 8, 1, 1, 7, 0, 6},
    {6,  0, 9, 3, 11, 4, 0,  9, 3, 3,  3, 10, 0, 9,
     10, 0, 9, 3, 3,  3, 10, 0, 2, 11, 3, 10, 0, 6},
    {6, 0, 0, 0, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 2, 4, 0, 0, 0, 6},
    {2, 1, 7, 0, 2, 4, 0, 8, 7, 0, 8, 1, 1, 1,
     1, 1, 1, 7, 0, 8, 7, 0, 2, 4, 0, 8, 1, 4},
    {2,  3, 10, 0,  9, 10, 0, 2, 4, 0,  9, 3, 3, 11,
     11, 3, 3,  10, 0, 2,  4, 0, 9, 10, 0, 9, 3, 4},
    {6, 0, 0, 0, 0, 0, 0, 2, 4, 0, 0, 0, 0, 2,
     4, 0, 0, 0, 0, 2, 4, 0, 0, 0, 0, 0, 0, 6},
    {6, 0, 8, 1, 1, 1,  1,  11, 11, 1, 1, 7, 0, 2,
     4, 0, 8, 1, 1, 11, 11, 1,  1,  1, 1, 7, 0, 6},
    {6,  0, 9, 3, 3, 3, 3, 3, 3, 3, 3, 10, 0, 9,
     10, 0, 9, 3, 3, 3, 3, 3, 3, 3, 3, 10, 0, 6},
    {6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6},
```

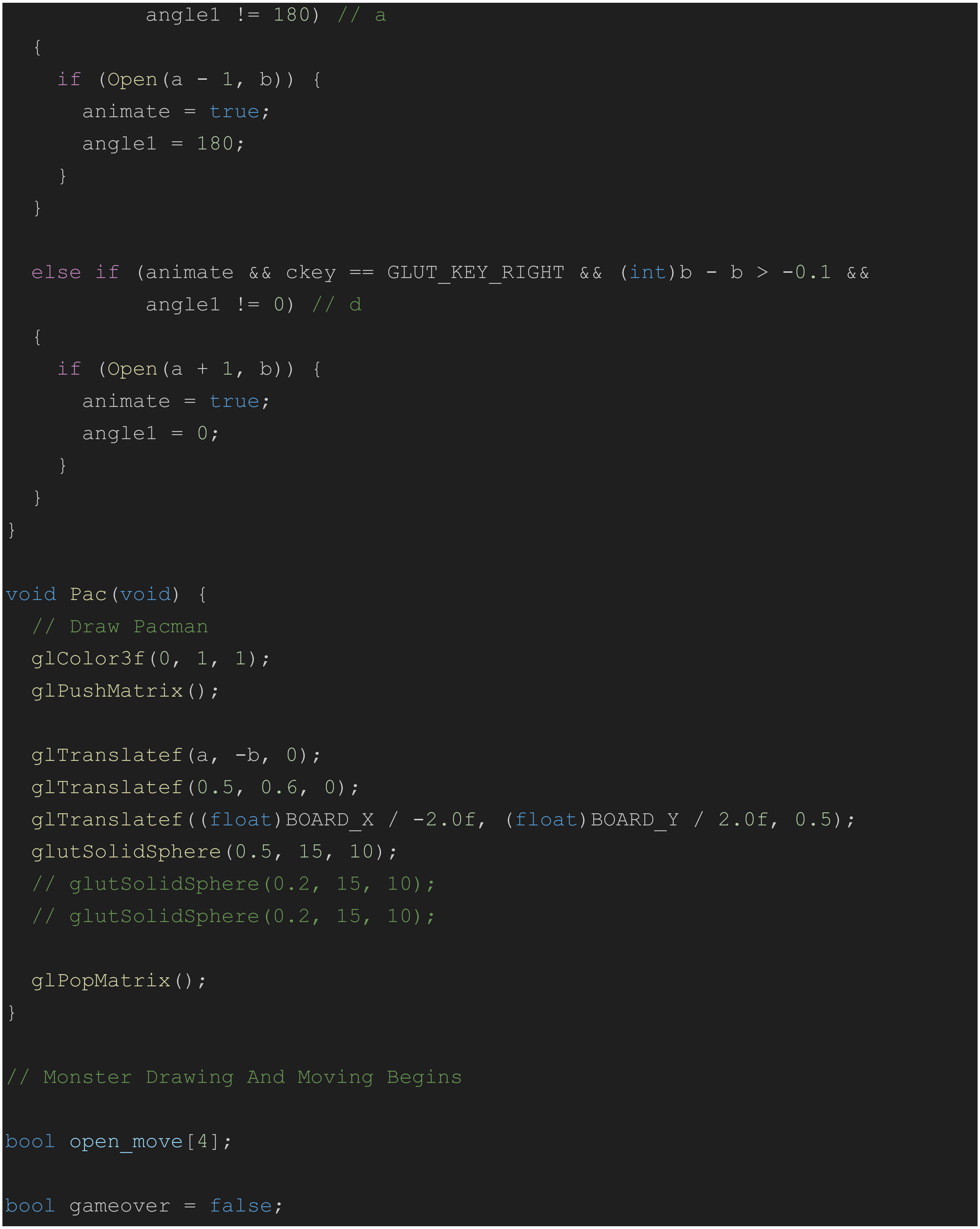


```
    {9, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
     5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 10}};

int pebble_array[BOARD_X][BOARD_Y] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
     0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
    {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
     0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
    {0, 3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
     0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 3, 0},
    {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
     0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
    {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
    {0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
     0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0},

    {0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
     0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0},
    {0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
     0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
```

]]>



```
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
     0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
    {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
     0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
    {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
     0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
    {0, 3, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
     0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 3, 0},
    {0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
     0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
    {0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
     0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
    {0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
     0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0},
    {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
     0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
    {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
     0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
    {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};

GLubyte list[5];

int tp_array[31][28];

int pebbles_left;

double speed1 = 0.1;
double angle1 = 90;

double a = 13.5, b = 23;

bool animate = false;

int lives = 3;

int points = 0;

void keys();
unsigned char ckey = 'w';
void mykey(unsigned char key, int x, int y);
bool Open(int a, int b);
void Move() {
  a += speed1 * cos(M_PI / 180 * angle1);
  b += speed1 * sin(M_PI / 180 * angle1);
  if (animate && ckey == GLUT_KEY_UP && (int)a - a > -0.1 && angle1 != 270)
// w
  {
    if (Open(a, b - 1)) {

      animate = true;

      angle1 = 270;
    }
  }

  else if (animate && ckey == GLUT_KEY_DOWN && (int)a - a > -0.1 &&
           angle1 != 90) // s
  {
    if (Open(a, b + 1)) {
      animate = true;
      angle1 = 90;
    }
  }

  else if (animate && ckey == GLUT_KEY_LEFT && (int)b - b > -0.1 &&
```

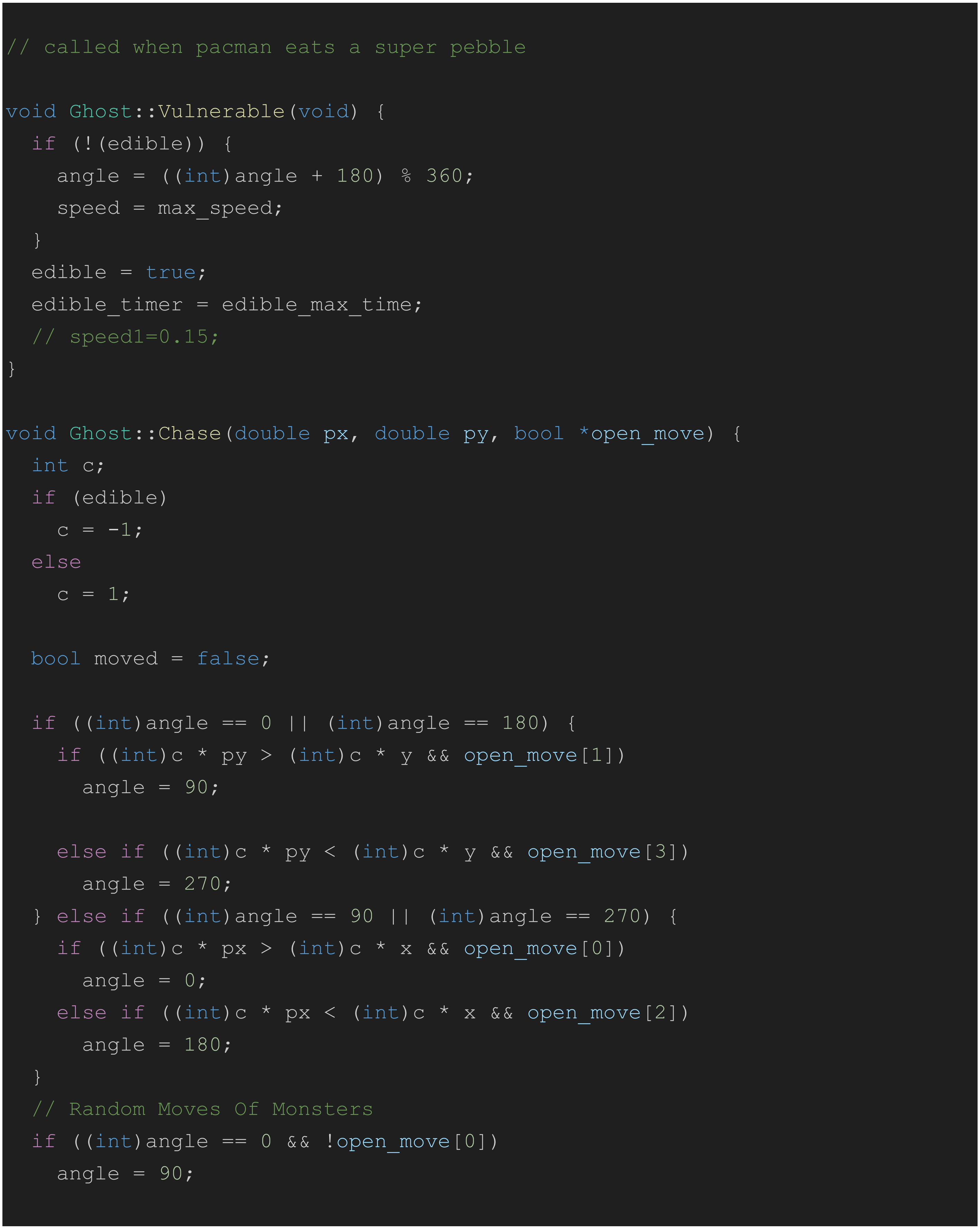


```
           angle1 != 180) // a
  {
    if (Open(a - 1, b)) {
      animate = true;
      angle1 = 180;
    }
  }

  else if (animate && ckey == GLUT_KEY_RIGHT && (int)b - b > -0.1 &&
           angle1 != 0) // d
  {
    if (Open(a + 1, b)) {
      animate = true;
      angle1 = 0;
    }
  }
}

void Pac(void) {
  // Draw Pacman
  glColor3f(0, 1, 1);
  glPushMatrix();

  glTranslatef(a, -b, 0);
  glTranslatef(0.5, 0.6, 0);
  glTranslatef((float)BOARD_X / -2.0f, (float)BOARD_Y / 2.0f, 0.5);
  glutSolidSphere(0.5, 15, 10);
  // glutSolidSphere(0.2, 15, 10);
  // glutSolidSphere(0.2, 15, 10);

  glPopMatrix();
}

// Monster Drawing And Moving Begins

bool open_move[4];

bool gameover = false;
```

```
int num_ghosts = 4;

int start_timer = 3;

class Ghost {
private:
public:
  bool edible;
  int edible_max_time;
  int edible_timer;
  bool eaten;
  bool transporting;
  float color[3];
  double speed;
  double max_speed;
  bool in_jail;
  int jail_timer;
  double angle;
  double x, y;

  Ghost(double, double);

  ~Ghost(void);

  void Move(); // Move the Monster

  void Update(void); // Update Monster State

  void Chase(double, double, bool *); // Chase Pacman

  bool Catch(double, double); // collision detection

  void Reinit(void);

  void Vulnerable(void);

  void Draw(void); // Draw the Monster
  void game_over(void);
```

```
};

Ghost *ghost[4];

Ghost::~Ghost(void) {}

Ghost::Ghost(double tx, double ty) {
  tx = x;
  ty = y;
  angle = 90;
  speed = max_speed = 1;
  color[0] = 1;
  color[1] = 0;
  color[2] = 0;
  eaten = false;
  edible_max_time = 10000;
  edible = false;
  in_jail = true;

  jail_timer = 30;
}

void Ghost::Reinit(void) {
  edible = false;
  in_jail = true;
  angle = 90;
}

// Move Monster
void Ghost::Move() {
  x += speed * cos(M_PI / 180 * angle);
  y += speed * sin(M_PI / 180 * angle);
}
void Ghost::game_over() {}

void Ghost::Update(void) {

  if ((int)x == 0 && (int)y == 14 && (!(transporting))) {
```

```
  angle = 180;
}

if (x < 0.1 && (int)y == 14) {
  x = 26.9;
  transporting = true;
}

if ((int)x == 27 && (int)y == 14 && (!(transporting))) {
  angle = 0;
}

if (x > 26.9 && (int)y == 14) {
  x = 0.1;
  transporting = true;
}
if ((int)x == 2 || (int)x == 25)
  transporting = false;
if (((int)x < 5 || (int)x > 21) && (int)y == 14 && !edible && !eaten)
  speed = max_speed / 2;
speed = max_speed;
// edibility
if (edible_timer == 0 && edible && !eaten) {

  edible = false;
  speed = max_speed;
}
if (edible)
  edible_timer--;

// JAIL
if (in_jail && (int)(y + 0.9) == 11) {
  in_jail = false;
  angle = 180;
}

if (in_jail && ((int)x == 13 || (int)x == 14)) {
  angle = 270;
  }

  // if time in jail is up, position for exit
  if (jail_timer == 0 && in_jail)

  {
    // move right to exit
    if (x < 13)
      angle = 0;
    if (x > 14)
      angle = 180;
  }

  // decrement time in jail counter
  if (jail_timer > 0)
    jail_timer--;

  // EATEN GHOST SEND TO JAIL
  if (eaten && ((int)x == 13 || (int)(x + 0.9) == 14) &&
      ((int)y > 10 && (int)y < 15)) {
    in_jail = true;
    angle = 90;
    if ((int)y == 14) {
      eaten = false;
      speed = max_speed;
      jail_timer = 66;
      x = 11;
    }
  }
}

bool Ghost::Catch(double px, double py) {
  // Collision Detection
  if (px - x < 0.2 && px - x > -0.2 && py - y < 0.2 && py - y > -0.2) {
    return true;
  }
  return false;
}
```

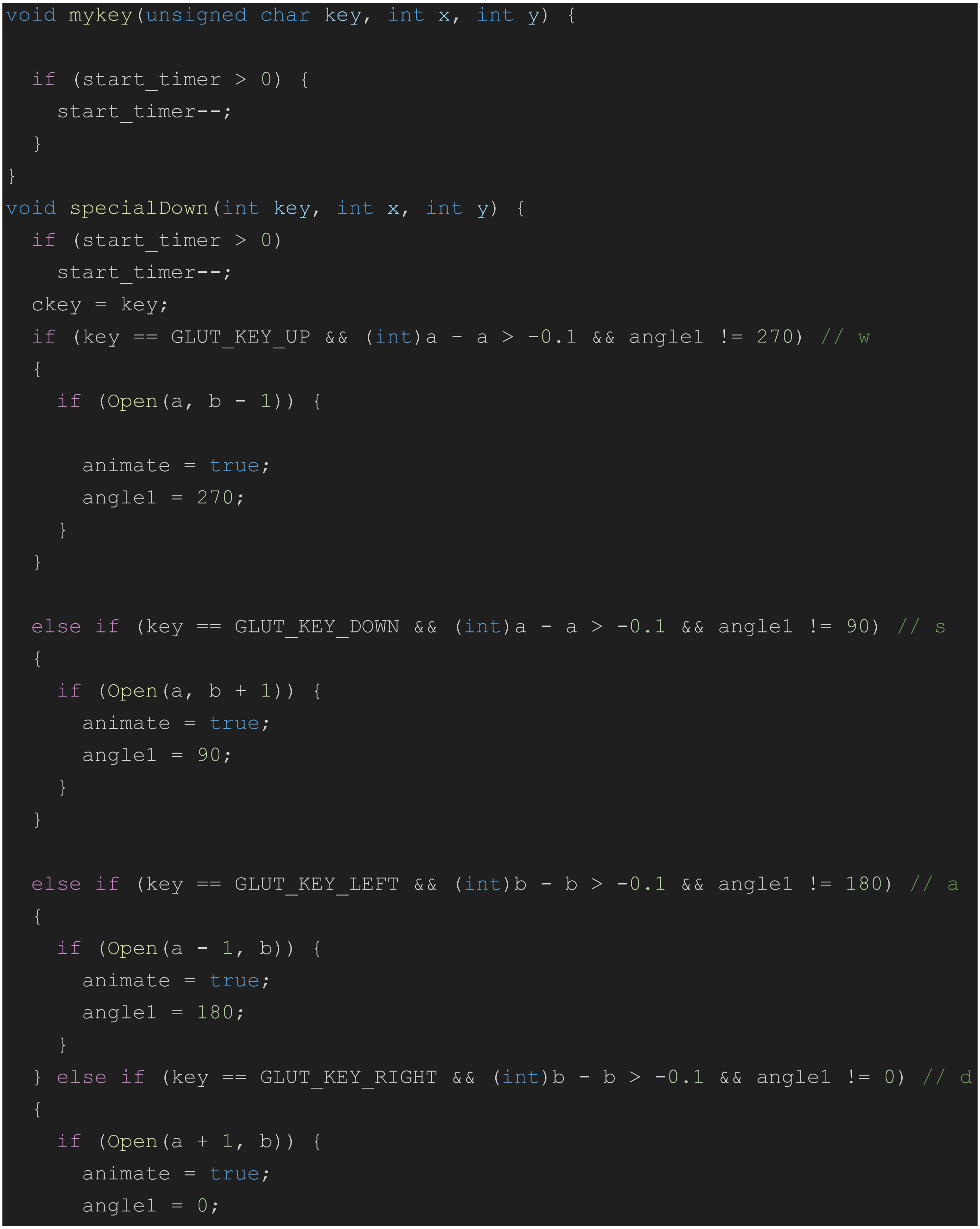


```
// called when pacman eats a super pebble

void Ghost::Vulnerable(void) {
  if (!(edible)) {
    angle = ((int)angle + 180) % 360;
    speed = max_speed;
  }
  edible = true;
  edible_timer = edible_max_time;
  // speed1=0.15;
}

void Ghost::Chase(double px, double py, bool *open_move) {
  int c;
  if (edible)
    c = -1;
  else
    c = 1;

  bool moved = false;

  if ((int)angle == 0 || (int)angle == 180) {
    if ((int)c * py > (int)c * y && open_move[1])
      angle = 90;

    else if ((int)c * py < (int)c * y && open_move[3])
      angle = 270;
  } else if ((int)angle == 90 || (int)angle == 270) {
    if ((int)c * px > (int)c * x && open_move[0])
      angle = 0;
    else if ((int)c * px < (int)c * x && open_move[2])
      angle = 180;
  }
  // Random Moves Of Monsters
  if ((int)angle == 0 && !open_move[0])
    angle = 90;
  if ((int)angle == 90 && !open_move[1])
    angle = 180;

  if ((int)angle == 180 && !open_move[2])
    angle = 270;

  if ((int)angle == 270 && !open_move[3])
    angle = 0;

  if ((int)angle == 0 && !open_move[0])
    angle = 90;
}

void Ghost::Draw(void) {

  if (!edible)
    glColor3f(color[0], color[1], color[2]);

  else {
    if (edible_timer < 150)
      glColor3f((edible_timer / 10) % 2, (edible_timer / 10) % 2, 1);
    if (edible_timer >= 150)
      glColor3f(0, 0, 1);
  }

  if (eaten)
    glColor3f(1, 1, 0); // When Eaten By PacMan Change Color To Yellow

  glPushMatrix();
  glTranslatef(x, -y, 0);
  glTranslatef(0.5, 0.6, 0);
  glTranslatef((float)BOARD_X / -2.0f, (float)BOARD_Y / 2.0f, 0.5);
  glutSolidSphere(.5, 10, 10);
  glPopMatrix();
}

void tp_restore(void) {
  for (int ISO = 0; ISO < BOARD_X; ISO++) {
```

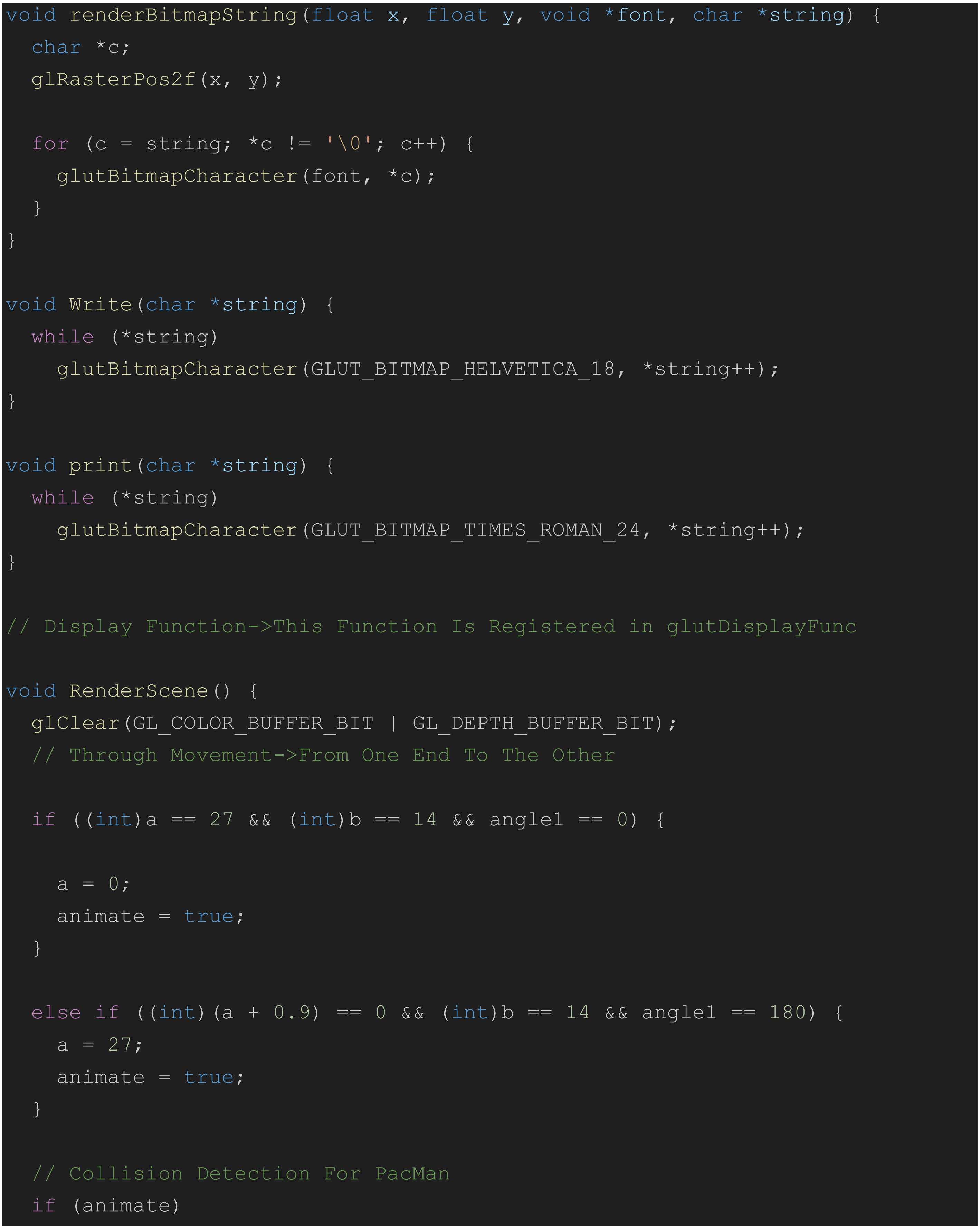
```
  for (int j = 0; j < BOARD_Y; j++) {
   tp_array[ISO][j] = pebble_array[ISO][j];
  }
 }
 pebbles_left = 244;
}

void Draw(void) {

 glColor3f(1, 0, 1);

 // split board drawing in half to avoid issues with depth
 for (int ISO = 0; ISO < BOARD_X; ISO++)

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

   glColor3f(0, 0, 1);
   int call_this = 0;

   glPushMatrix();
   glTranslatef(-(float)BOARD_X / 2.0f, -(float)BOARD_Y / 2.0f, 0);

   glTranslatef(j, BOARD_Y - ISO, 0);

   glPushMatrix();
   glTranslatef(0.5, 0.5, 0);

   switch (board_array[ISO][j]) {
   case 4:
    glRotatef(90.0, 0, 0, 1);
   case 3:
    glRotatef(90.0, 0, 0, 1);
   case 2:
    glRotatef(90.0, 0, 0, 1);
   case 1:
    call_this = 1;
    break;
```

```
      case 6:
        glRotatef(90.0, 0, 0, 1);
      case 5:
        call_this = 2;
        break;
      case 10:
        glRotatef(90.0, 0, 0, 1);
      case 9:
        glRotatef(90.0, 0, 0, 1);
      case 8:
        glRotatef(90.0, 0, 0, 1);
      case 7:
        call_this = 3;
        break;
      }

      glScalef(1, 1, 0.5);
      glTranslatef(-0.5, -0.5, 0);
      glCallList(list[call_this]);
      glPopMatrix();
      // now put on the top of the cell
      if (call_this != 0 || board_array[ISO][j] == 11) {
        glTranslatef(0, 0, -0.5);
        glCallList(list[4]);
      }
      glPopMatrix();

      if (tp_array[ISO][j] > 0) {

        glColor3f(0, 300, 1 / (float)tp_array[ISO][j]);
        glPushMatrix();
        glTranslatef(-(float)BOARD_X / 2.0f, -(float)BOARD_Y / 2.0f, 0);
        glTranslatef(j, BOARD_Y - ISO, 0);
        glTranslatef(0.5, 0.5, 0.5);
        glutSolidSphere(0.1f * ((float)tp_array[ISO][j]), 6, 6);
        glPopMatrix();
      }
    }
```

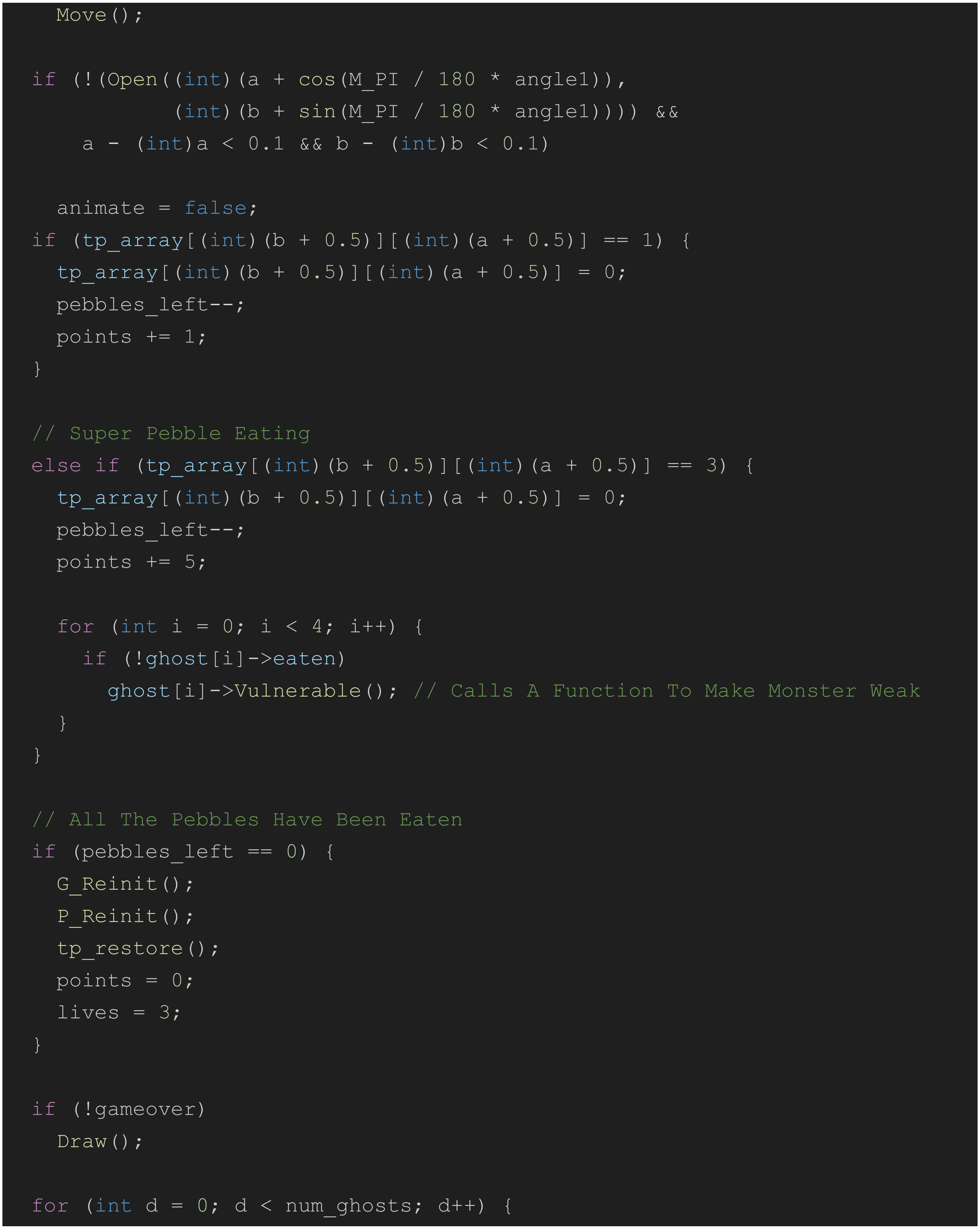
```
}

int ISO;

for (ISO = 0; ISO < BOARD_X; ISO++) {
  for (int j = BOARD_Y - 1; j >= BOARD_Y / 2; j--) {
    glColor3f(0, 0, 1);
    int call_this = 0;

    glPushMatrix();

    glTranslatef(-(float)BOARD_X / 2.0f, -(float)BOARD_Y / 2.0f, 0);
    glTranslatef(j, BOARD_Y - ISO, 0);

    glPushMatrix();
    glTranslatef(0.5, 0.5, 0);
    switch (board_array[ISO][j]) {
    case 4:
      glRotatef(90.0, 0, 0, 1);
    case 3:
      glRotatef(90.0, 0, 0, 1);
    case 2:
      glRotatef(90.0, 0, 0, 1);
    case 1:
      call_this = 1;
      break;
    case 6:
      glRotatef(90.0, 0, 0, 1);
    case 5:
      call_this = 2;
      break;
    case 10:
      glRotatef(90.0, 0, 0, 1);
    case 9:
      glRotatef(90.0, 0, 0, 1);
    case 8:
      glRotatef(90.0, 0, 0, 1);
    case 7:
        call_this = 3;
        break;
      }
      glScalef(1, 1, 0.5);
      glTranslatef(-0.5, -0.5, 0);
      glCallList(list[call_this]);

      glPopMatrix();
      // now put on top
      if (call_this != 0 || board_array[ISO][j] == 11) {
        glTranslatef(0, 0, -0.5);
        glCallList(list[4]);
      }
      glPopMatrix();

      if (tp_array[ISO][j] > 0) {
        glColor3f(0, 300, 1 / (float)tp_array[ISO][j]);
        glPushMatrix();
        glTranslatef(-(float)BOARD_X / 2.0f, -(float)BOARD_Y / 2.0f, 0);
        glTranslatef(j, BOARD_Y - ISO, 0);
        glTranslatef(0.5, 0.5, 0.5);
        glutSolidSphere(0.1f * ((float)tp_array[ISO][j]), 6, 6);
        glPopMatrix();
      }
    }
  }
  Pac();
}

bool Open(int a, int b) {
  if (board_array[b][a] > 0) {
    return false;
  }
  return true;
}

void RenderScene();
```



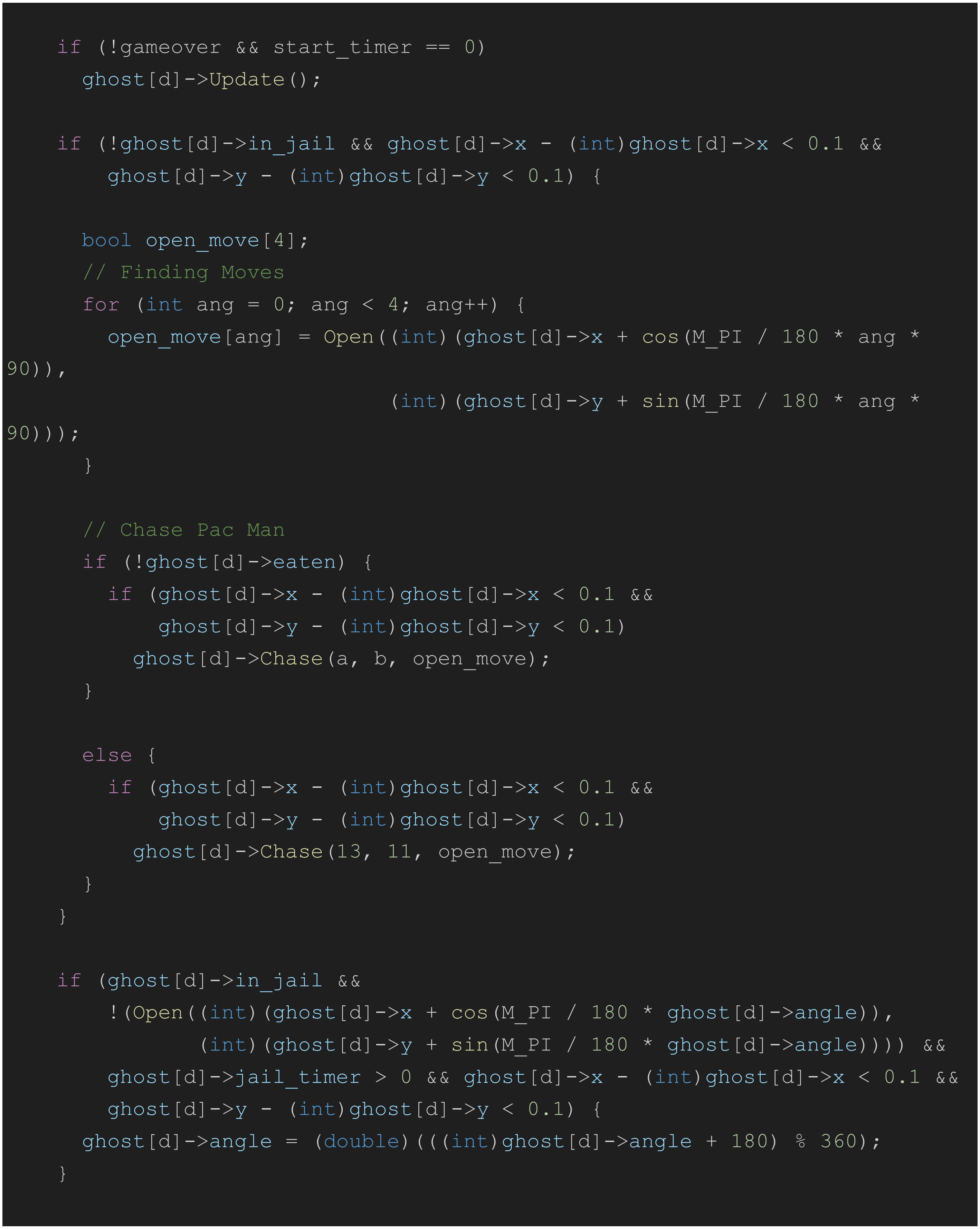
```
void mykey(unsigned char key, int x, int y) {

  if (start_timer > 0) {
    start_timer--;
  }
}
void specialDown(int key, int x, int y) {
  if (start_timer > 0)
    start_timer--;
  ckey = key;
  if (key == GLUT_KEY_UP && (int)a - a > -0.1 && angle1 != 270) // w
  {
    if (Open(a, b - 1)) {

      animate = true;
      angle1 = 270;
    }
  }

  else if (key == GLUT_KEY_DOWN && (int)a - a > -0.1 && angle1 != 90) // s
  {
    if (Open(a, b + 1)) {
      animate = true;
      angle1 = 90;
    }
  }

  else if (key == GLUT_KEY_LEFT && (int)b - b > -0.1 && angle1 != 180) // a
  {
    if (Open(a - 1, b)) {
      animate = true;
      angle1 = 180;
    }
  } else if (key == GLUT_KEY_RIGHT && (int)b - b > -0.1 && angle1 != 0) // d
  {
    if (Open(a + 1, b)) {
      animate = true;
      angle1 = 0;
    }
  }
}

void specialUp(int key, int x, int y) {}

void P_Reinit() {
  a = 13.5;
  b = 23;
  angle1 = 90;
  animate = false;
  Pac();
}

void G_Reinit(void) {

  start_timer = 3;

  // ghost initial starting positions
  int start_x[4] = {11, 12, 15, 16};
  float ghost_colors[4][3] = {
      {255, 0, 0}, {120, 240, 120}, {255, 200, 200}, {255, 125, 0}};

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

    ghost[i]->Reinit();
    ghost[i]->x = start_x[i];
    ghost[i]->y = 14;
    ghost[i]->eaten = false;
    ghost[i]->jail_timer = i * 33 + 66;
    ghost[i]->max_speed = 0.1 - 0.01 * (float)i;
    ghost[i]->speed = ghost[i]->max_speed;

    // colorize ghosts
    for (int j = 0; j < 3; j++)
      ghost[i]->color[j] = ghost_colors[i][j] / 255.0f;
  }
}
```



```
void renderBitmapString(float x, float y, void *font, char *string) {
  char *c;
  glRasterPos2f(x, y);

  for (c = string; *c != '\0'; c++) {
    glutBitmapCharacter(font, *c);
  }
}

void Write(char *string) {
  while (*string)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *string++);
}

void print(char *string) {
  while (*string)
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *string++);
}

// Display Function->This Function Is Registered in glutDisplayFunc

void RenderScene() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  // Through Movement->From One End To The Other

  if ((int)a == 27 && (int)b == 14 && angle1 == 0) {

    a = 0;
    animate = true;
  }

  else if ((int)(a + 0.9) == 0 && (int)b == 14 && angle1 == 180) {
    a = 27;
    animate = true;
  }

  // Collision Detection For PacMan
  if (animate)
```



```
    Move();

  if (!(Open((int)(a + cos(M_PI / 180 * angle1)),
             (int)(b + sin(M_PI / 180 * angle1)))) &&
      a - (int)a < 0.1 && b - (int)b < 0.1)

    animate = false;
  if (tp_array[(int)(b + 0.5)][(int)(a + 0.5)] == 1) {
    tp_array[(int)(b + 0.5)][(int)(a + 0.5)] = 0;
    pebbles_left--;
    points += 1;
  }

  // Super Pebble Eating
  else if (tp_array[(int)(b + 0.5)][(int)(a + 0.5)] == 3) {
    tp_array[(int)(b + 0.5)][(int)(a + 0.5)] = 0;
    pebbles_left--;
    points += 5;

    for (int i = 0; i < 4; i++) {
      if (!ghost[i]->eaten)
        ghost[i]->Vulnerable(); // Calls A Function To Make Monster Weak
    }
  }

  // All The Pebbles Have Been Eaten
  if (pebbles_left == 0) {
    G_Reinit();
    P_Reinit();
    tp_restore();
    points = 0;
    lives = 3;
  }

  if (!gameover)
    Draw();

  for (int d = 0; d < num_ghosts; d++) {
```



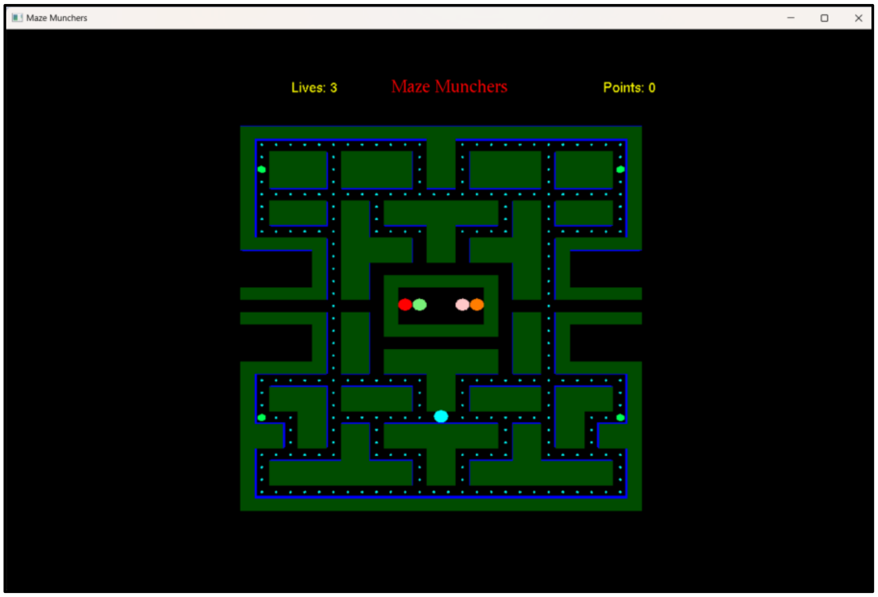
```
    if (!gameover && start_timer == 0)
      ghost[d]->Update();

    if (!ghost[d]->in_jail && ghost[d]->x - (int)ghost[d]->x < 0.1 &&
        ghost[d]->y - (int)ghost[d]->y < 0.1) {

      bool open_move[4];
      // Finding Moves
      for (int ang = 0; ang < 4; ang++) {
        open_move[ang] = Open((int)(ghost[d]->x + cos(M_PI / 180 * ang *
90)),
                              (int)(ghost[d]->y + sin(M_PI / 180 * ang *
90)));
      }

      // Chase Pac Man
      if (!ghost[d]->eaten) {
        if (ghost[d]->x - (int)ghost[d]->x < 0.1 &&
            ghost[d]->y - (int)ghost[d]->y < 0.1)
          ghost[d]->Chase(a, b, open_move);
      }

      else {
        if (ghost[d]->x - (int)ghost[d]->x < 0.1 &&
            ghost[d]->y - (int)ghost[d]->y < 0.1)
          ghost[d]->Chase(13, 11, open_move);
      }
    }

    if (ghost[d]->in_jail &&
        !(Open((int)(ghost[d]->x + cos(M_PI / 180 * ghost[d]->angle)),
               (int)(ghost[d]->y + sin(M_PI / 180 * ghost[d]->angle)))) &&
        ghost[d]->jail_timer > 0 && ghost[d]->x - (int)ghost[d]->x < 0.1 &&
        ghost[d]->y - (int)ghost[d]->y < 0.1) {
      ghost[d]->angle = (double)(((int)ghost[d]->angle + 180) % 360);
    }
    if (!gameover && start_timer == 0)
      ghost[d]->Move();
    ghost[d]->Draw();

    if (!(ghost[d]->eaten)) {
      bool collide = ghost[d]->Catch(a, b);

      // Monster Eats PacMan
      if (collide && !(ghost[d]->edible)) {
        if (!gameover)
          lives--;

        if (lives == 0) {
          gameover = true;
          lives = 0;
          ghost[d]->game_over();
        }

        P_Reinit();
        d = 4;
      }
      // PacMan Eats Monster And Sends It To Jail
      else if (collide && ((ghost[d]->edible))) {
        ghost[d]->edible = false;

        ghost[d]->eaten = true;
        ghost[d]->speed = 1;
      }
    }
  }

  char game[10] = "GAME OVER";

  if (gameover == true) {
    glColor3f(1, 0, 0);
    renderBitmapString(-5, 0.5, GLUT_BITMAP_HELVETICA_18, game);
  }
  char tmp_str[40];

  glColor3f(1, 1, 0);
  glRasterPos2f(10, 18);
  sprintf_s(tmp_str, "Points: %d", points);
  Write(tmp_str);

  glColor3f(1, 0, 0);
  glRasterPos2f(-5, 18);
  sprintf_s(tmp_str, "Maze Munchers");
  print(tmp_str);

  glColor3f(1, 1, 0);
  glRasterPos2f(-12, 18);
  sprintf_s(tmp_str, "Lives: %d", lives);
  Write(tmp_str);

  glutPostRedisplay();
  glutSwapBuffers();
}

void create_list_lib() {
  // Set Up Maze Using Lists
  list[1] = glGenLists(1);

  glNewList(list[1], GL_COMPILE);

  // North Wall
  glBegin(GL_QUADS);
  glColor3f(0, 0, 1);
  glNormal3f(0.0, 1.0, 0.0);
  glVertex3f(1.0, 1.0, 1.0);
  glVertex3f(1.0, 1.0, 0.0);
  glVertex3f(0.0, 1.0, 0.0);
  glVertex3f(0.0, 1.0, 1.0);
  glEnd();
  glEndList();
  list[2] = glGenLists(1);
```

```
glNewList(list[2], GL_COMPILE);

glBegin(GL_QUADS);
// North Wall
glColor3f(0, 0, 1);
glNormal3f(0.0, 1.0, 0.0);
glVertex3f(1.0, 1.0, 1.0);
glVertex3f(1.0, 1.0, 0.0);
glVertex3f(0.0, 1.0, 0.0);
glVertex3f(0.0, 1.0, 1.0);
// South Wall
glColor3f(0, 0, 1);
glNormal3f(0.0, -1.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 1.0);
glVertex3f(0.0, 0.0, 1.0);
glVertex3f(0.0, 0.0, 0.0);
glEnd();
glEndList();

list[3] = glGenLists(1);

glNewList(list[3], GL_COMPILE);
glBegin(GL_QUADS);
// North Wall
glColor3f(0, 0, 1);
glNormal3f(0.0f, 1.0f, 0.0f);
glVertex3f(1.0, 1.0, 1.0);
glVertex3f(1.0, 1.0, 0.0);
glVertex3f(0.0, 1.0, 0.0);
glVertex3f(0.0, 1.0, 1.0);
// East Wall
glColor3f(0, 0, 1);
glNormal3f(1.0, 0.0, 0.0);
glVertex3f(1.0, 1.0, 0.0);
glVertex3f(1.0, 1.0, 1.0);
glVertex3f(1.0, 0.0, 1.0);
glVertex3f(1.0, 0.0, 0.0);
```

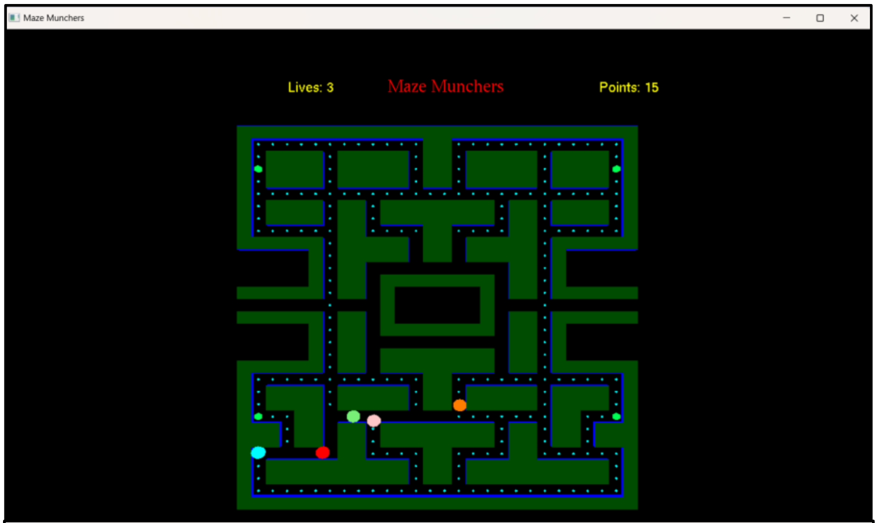
```
  glEnd();
  glEndList();

  list[4] = glGenLists(1);

  glNewList(list[4], GL_COMPILE);
  glBegin(GL_QUADS);
  // Top Wall
  glColor3f(-1, 0.3, 0);
  glNormal3f(1.0, 0.0, 1.0);
  glVertex3f(1, 1, 1.0);
  glVertex3f(0, 1, 1.0);
  glVertex3f(0, 0, 1.0);
  glVertex3f(1, 0, 1.0);
  glEnd();
  glEndList();
}

void init() {

  glEnable(GL_NORMALIZE);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluPerspective(60, 1.33, 0.005, 100);

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(-1.5, 0, 40, -1.5, 0, 0, 0.0f, 1.0f, 0.0f);
}

void erase() {
  glColor3f(0.1, 0.0, 0.0);
  glBegin(GL_POLYGON);
  glVertex2f(0, 0);
  glVertex2f(0.5, 0);
  glVertex2f(0.25, 0.5);
  glEnd();
}

int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowSize(1200, 780);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("Maze Munchers");
  init();

  glutDisplayFunc(RenderScene);

  create_list_lib();

  glutKeyboardFunc(mykey);

  glutSpecialFunc(specialDown);
  glutSpecialUpFunc(specialUp);

  glEnable(GL_DEPTH_TEST);

  int start_x[4] = {11, 12, 15, 16};

  for (int ISO = 0; ISO < num_ghosts; ISO++) {
    ghost[ISO] = new Ghost(start_x[ISO], 14);
  }

  float ghost_colors[4][3] = {
      {255, 0, 0}, {120, 240, 120}, {255, 200, 200}, {255, 125, 0}};
  int ISO;

  for (ISO = 0; ISO < num_ghosts; ISO++) {
    ghost[ISO]->x = start_x[ISO];
    ghost[ISO]->y = 14;
    ghost[ISO]->eaten = false;
    ghost[ISO]->max_speed = 0.1 - 0.01 * (float)ISO;
    ghost[ISO]->speed = ghost[ISO]->max_speed;
```

```
      // colorize ghosts
      for (int j = 0; j < 3; j++)
        ghost[ISO]->color[j] = ghost_colors[ISO][j] / 255.0f;
    }

    for (ISO = 0; ISO < BOARD_X; ISO++) {
      for (int j = 0; j < BOARD_Y; j++) {
        tp_array[ISO][j] = pebble_array[ISO][j];
      }
    }

    pebbles_left = 244;
    glShadeModel(GL_SMOOTH);
    glutMainLoop();
    return 0;
}
```

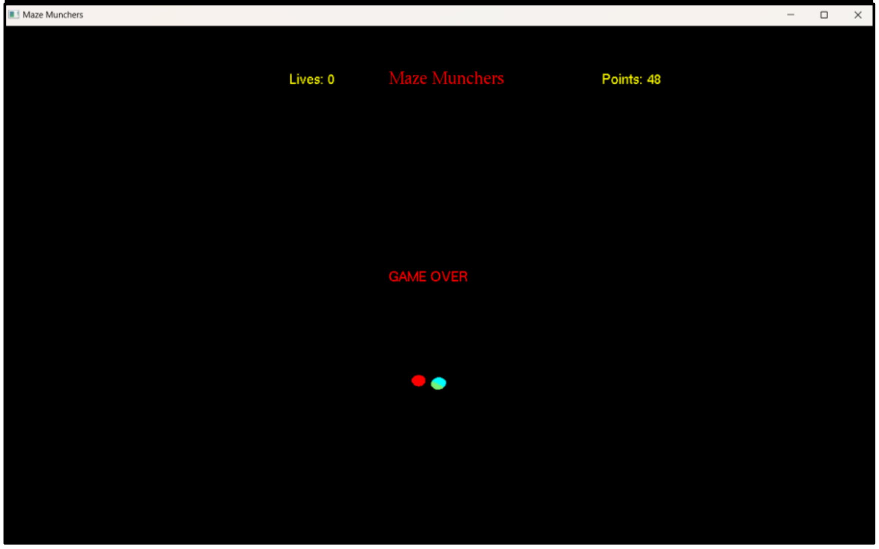
Output



Lives: 3 Maze Munchers Points: 0



Lives: 3 Maze Munchers Points: 15



Lives: 0 Maze Munchers Points: 48

GAME OVER