Cross Road Game
OpenGL Mini Project Documentation

William Mikhail 7661
Platform macOS

Overview

The **Cross Road Game** is a simple OpenGL-based 2D game where the player controls a block that attempts to cross roads filled with moving obstacles (cars). The goal of the game is to avoid colliding with the cars while moving forward.

The game uses the **W, A, S, D** keys to control the player's movements. The game ends if the player collides with any obstacles, and the player can restart by pressing the **R** key.

Features

- **Player Movement:** Use the **W, A, S, D** keys to move the character up, down, left, and right.
- **Obstacles:** Moving cars that represent obstacles the player must avoid.
- **Game Over State:** The game ends if the player collides with any car.
- **Restart Functionality:** The player can restart the game by pressing **R** after a game over.
- **Simple Graphics:** The game uses basic colored blocks for the player and obstacles.

Code Breakdown

1. Character Properties

The character (player) is represented by a simple block that moves on the screen. The following variables define the character's position and size:

```
// Character properties
float characterX = 0.0f, characterY = -0.8f;
float characterSize = 0.05f;
```

- **characterX** and **characterY** determine the position of the player on the screen.
- **characterSize** defines the size of the character.

2. Obstacle Properties

The moving obstacles (cars) are represented by several blocks. Each car has a position and speed that determines how fast it moves.

```
// Obstacle properties
float carX[3] = {-0.8f, 0.0f, 0.8f}; // Initial positions of cars
float carY[3] = {0.5f, 0.0f, -0.5f}; // Lanes
```

```
float carSpeed[3] = {0.01f, 0.015f, 0.02f}; // Speeds
```

- **carX[]** stores the x-coordinate for each car.
- **carY[]** stores the y-coordinate (lane) for each car.
- **carSpeed[]** determines the speed of each car.

3. Drawing a Rectangle

A helper function, drawRectangle(), is used to draw the player character and the cars. It draws a simple rectangle based on the given coordinates and dimensions:

```
// Function to draw a rectangle
void drawRectangle(float x, float y, float width, float height) {
    glBegin(GL_QUADS);
    glVertex2f(x - width / 2, y - height / 2);
    glVertex2f(x + width / 2, y - height / 2);
    glVertex2f(x + width / 2, y + height / 2);
    glVertex2f(x - width / 2, y + height / 2);
    glEnd();
}
```

This function uses OpenGL's **GL_QUADS** to create rectangles. The player and cars are drawn using this function with their respective positions and sizes.

4. Display Function

The **display()** function is responsible for rendering the game scene. It clears the screen and draws the player character and the obstacles:

```
// Function to display the scene
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    if (!gameOver) {
        // Draw the player character
        glColor3f(0.0f, 1.0f, 0.0f); // Green
        drawRectangle(characterX, characterY, characterSize,
characterSize);

        // Draw the cars
        glColor3f(1.0f, 0.0f, 0.0f); // Red
        for (int i = 0; i < 3; i++) {
            drawRectangle(carX[i], carY[i], 0.1f, 0.05f);
        }
    } else {
        // Game Over message
        glColor3f(1.0f, 1.0f, 1.0f); // White
        glRasterPos2f(-0.2f, 0.0f);
```

```
        const char *message = "Game Over!";
        while (*message) {
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *message++);
        }
    }

    glFlush();
}
```

- If the game is not over, the function draws the player and the cars.
- If the game is over, it displays the **"Game Over!"** message.

5. Updating the Game State

The **update()** function handles the logic for moving the cars and checking for collisions. It is called repeatedly using a timer:

```
// Function to update the scene
void update(int value) {
    if (!gameOver) {
        // Move cars
        for (int i = 0; i < 3; i++) {
            carX[i] -= carSpeed[i];
            if (carX[i] < -1.0f) carX[i] = 1.0f; // Reset car when it
moves off-screen
        }

        // Check for collisions
        for (int i = 0; i < 3; i++) {
            if (characterY < carY[i] + 0.05f && characterY > carY[i] -
0.05f &&
                characterX < carX[i] + 0.05f && characterX > carX[i] -
0.05f) {
                gameOver = true;
            }
        }

        glutPostRedisplay();
        glutTimerFunc(16, update, 0); // Call update every 16ms (~60 FPS)
    }
}
```

- Cars move to the left by subtracting their speed from the **carX[]** positions.
- If any car collides with the player (based on their positions), the game ends.

6. Handling User Input

The **handleKeys()** function processes keyboard input. The player uses the **W, A, S, D** keys to move and **R** to restart the game:

```
// Function to handle keyboard input
void handleKeys(unsigned char key, int x, int y) {
    if (!gameOver) {
        switch (key) {
            case 'w': // Move up
                characterY += 0.1f;
                break;
            case 's': // Move down
                characterY -= 0.1f;
                break;
            case 'a': // Move left
                characterX -= 0.1f;
                break;
            case 'd': // Move right
                characterX += 0.1f;
                break;
        }
    }

    // Restart the game
    if (gameOver && (key == 'r' || key == 'R')) {
        characterX = 0.0f;
        characterY = -0.8f;
        gameOver = false;
        glutTimerFunc(16, update, 0); // Restart the update loop
    }

    glutPostRedisplay();
}
```

- The character's position is updated based on the pressed key.
- If the game is over and **R** is pressed, the game resets.

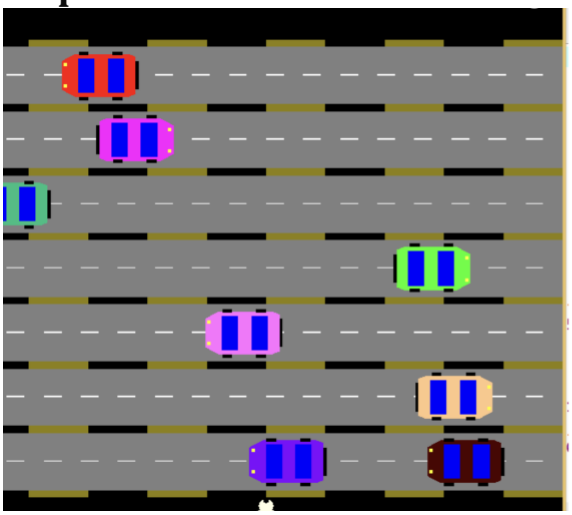
7. Initialization

The **init()** function initializes OpenGL settings such as background color and coordinate system:

```
// Initialize OpenGL settings
void init() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black background
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0); // Set the coordinate system
}
```

This function sets up the clear color for the screen and defines the 2D coordinate system.

Inspiration



1 Add to cart

Computer Graphics Project

Cross Road Game Computer Graphics Mini Project in OpenGL

~~₹500.00~~ **₹299.00**

The Source Code is Downloadable immediately after the successful payment

Project Description

This game is inspired by **Crossy Road**, a popular arcade game where the player controls a character trying to cross busy roads while avoiding moving vehicles. The goal is to create a simpler version with basic graphics and gameplay, while aiming to enhance it with future improvements.

Future Improvements

- **Home Page:** Add a start page where the player can begin a new game or see high scores.
- **Improved Graphics:** Enhance the graphics by adding textures and animations.
- **Additional Features:**
  - Implement shooting mechanics, allowing the player to shoot at obstacles.
  - Add power-ups to make the game more engaging.
- **Sound Effects:** Incorporate sound effects for actions like moving, collisions, and background music.

Conclusion

The **Cross Road Game** offers a simple yet engaging experience, with a focus on basic OpenGL game development. With the ability to move, avoid obstacles, and restart after a game over, it serves as a foundation for more complex games. Future updates will include improved gameplay mechanics, graphics, and additional features.