

**A**

**Project report**

**On**

Computer Graphics & Gaming

Project Phase 3

**By**

**Student :Alfiya Javed Sayyed**

**SRN No : 202202232**

**Roll no:58**

**Div : F**

**Batch:F3**

**Second Year Engineering**

**Department of Computer Engineering**

**Science & Technology**

**Vishwakarma University**

Kondhwa Budruk, Pune - 411 048. Maharashtra, India.

**Academic Year 2023-24**

Project Number - 03

**Project Report:**

**Making a Space Race game using OpenGL.**

**Project Logic :**

The provided C code implements a simple 2D game called "Space Race" using OpenGL and GLUT library. The game features two player-controlled ships navigating through a field of moving debris. Each player takes turns moving their ship vertically, avoiding collisions with the debris while aiming to reach the top of the screen.

The game loop continuously updates the game state, handling player input for ship movement and updating debris positions. Collision detection mechanisms detect when a ship collides with debris, resetting the ship's position and keeping track of collision counts for each player.

The game employs a turn-based system, indicating which player's turn it is and displaying messages accordingly. Upon reaching the top, a player is declared the winner based on their collision count compared to their opponent's.

The user interface consists of a simple window displaying the game scene and relevant text messages, providing instructions and feedback to the players. Overall, "Space Race" offers an engaging and competitive gameplay experience within a concise and straightforward code structure.

**Explanation of Algorithms :**

1. Initialization Functions: initShips() and initDebris() initialize the ships' and debris' initial positions and states, respectively. They set up the game environment before it begins.
2. Drawing Functions: drawShips() and drawDebris() are responsible for rendering the ships and debris on the screen using OpenGL primitives. They ensure that the game elements are displayed correctly to the player.
3. Update Functions: updateShips(), updateDebris(), and checkCollisions() handle the game logic updates. They update the positions of ships and debris based on player input and the game's rules. checkCollisions() detects collisions between ships and debris and updates collision counts accordingly.
4. Display Function: display() is the main rendering function. It calls the drawing functions and displays text messages to provide feedback to the player regarding the game state, including whose turn it is, when a player reaches the top, and the collision counts.
5. Input Handling Functions: handleKeypress(), handleKeyReleased(), handleSpecialKeypress(), and handleSpecialKeyReleased() manage user input. They respond to keyboard events, allowing players to control their ships by moving them up or down.
6. Main Loop: The main loop in the main() function initializes GLUT and sets up event handlers. It then enters the GLUT main loop, where it continuously updates the game state and renders the scene until the program exits.

Overall, the algorithms in the code ensure smooth gameplay by updating the game state, handling user input, and rendering the game scene effectively.

**Code Explanation :**

1. **Header Includes**: The code includes necessary header files like **<stdio.h>**, **<stdlib.h>**, **<stdbool.h>**, **<time.h>**, **<GL/glut.h>**, **<math.h>**, and **<GL/freeglut.h>**.
2. **Constants Definition**: Constants like **SCREEN\_WIDTH**, **SCREEN\_HEIGHT**, **PLAYER\_WIDTH**, **PLAYER\_HEIGHT**, **DEBRIS\_RADIUS**, and **NUM\_DEBRIS** are defined for setting up the game environment.
3. **Structures Definition**: Two structures **Ship** and **Debris** are defined to represent the player ships and debris objects, respectively. They hold information such as position and state of each object.
4. **Initialization Functions**: **initShips()** and **initDebris()** initialize the ships' and debris' initial positions and states.
5. **Drawing Functions**: **drawShips()** and **drawDebris()** use OpenGL primitives to draw the ships and debris on the screen.
6. **Update Functions**: **updateShips()**, **updateDebris()**, and **checkCollisions()** update the game state. They handle ship movement, debris movement, and collision detection, respectively.
7. **Display Function**: **display()** renders the game scene. It calls drawing functions and displays text messages indicating whose turn it is, when a player reaches the top, and collision counts.
8. **Input Handling Functions**: **handleKeypress()**, **handleKeyReleased()**, **handleSpecialKeypress()**, and **handleSpecialKeyReleased()** manage user input from the keyboard.
9. **Main Loop**: The **main()** function initializes GLUT, sets up event handlers, and enters the GLUT main loop. It continuously updates the game state and renders the scene until the program exits.

**OpenGL Setup:-**

1. **Install Dev C++**: If you haven't already, download and install Dev C++ from its official website.
2. **Install OpenGL**: Dev C++ doesn't come with OpenGL libraries by default, so you need to install them separately. You can download the OpenGL libraries from websites like <https://www.opengl.org/> or use precompiled libraries like FreeGLUT.
3. **Setting up Dev C++ for OpenGL**:
   * Open Dev C++.
   * Go to "Tools" -> "Compiler Options".
   * In the "Parameters" tab, add the OpenGL include directory to "Include directories" and the OpenGL library directory to "Libraries directories".
   * In the "Linker" tab, add the necessary OpenGL libraries (like **opengl32**, **glu32**, **glut32**, etc.) to "Linker libraries".
4. **Test your setup**: Write a simple OpenGL program to verify that everything is set up correctly.

**Source Code:-**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

#include <GL/glut.h>

#include <math.h>

#include <GL/freeglut.h>

#define SCREEN\_WIDTH 640

#define SCREEN\_HEIGHT 480

#define PLAYER\_WIDTH 20

#define PLAYER\_HEIGHT 40

#define DEBRIS\_RADIUS 5

#define NUM\_DEBRIS 30

typedef struct {

int x;

int y;

bool isUp;

bool isDown;

bool reachedTop;

} Ship;

typedef struct {

int x;

int y;

bool isGoingLeft;

} Debris;

Ship player1;

Ship player2;

Debris debris[NUM\_DEBRIS];

bool player1Turn = true;

int player1Collisions = 0;

int player2Collisions = 0;

void initShips() {

player1.x = SCREEN\_WIDTH / 3;

player1.y = SCREEN\_HEIGHT - 20;

player1.isUp = false;

player1.isDown = false;

player1.reachedTop = false;

player2.x = 2 \* SCREEN\_WIDTH / 3;

player2.y = SCREEN\_HEIGHT - 20;

player2.isUp = false;

player2.isDown = false;

player2.reachedTop = false;

}

void initDebris() {

srand(time(NULL));

for (int i = 0; i < NUM\_DEBRIS; ++i) {

debris[i].x = (rand() % SCREEN\_WIDTH);

debris[i].y = (rand() % (SCREEN\_HEIGHT - 20)) + 10;

debris[i].isGoingLeft = (rand() % 2) == 0;

}

}

void drawShips() {

// Draw player 1 ship

glBegin(GL\_QUADS);

glColor3f(1.0f, 0.0f, 0.0f);

glVertex2f(player1.x, player1.y);

glVertex2f(player1.x + PLAYER\_WIDTH, player1.y);

glVertex2f(player1.x + PLAYER\_WIDTH, player1.y + PLAYER\_HEIGHT);

glVertex2f(player1.x, player1.y + PLAYER\_HEIGHT);

glEnd();

// Draw player 2 ship

glBegin(GL\_QUADS);

glColor3f(0.0f, 0.0f, 1.0f);

glVertex2f(player2.x, player2.y);

glVertex2f(player2.x + PLAYER\_WIDTH, player2.y);

glVertex2f(player2.x + PLAYER\_WIDTH, player2.y + PLAYER\_HEIGHT);

glVertex2f(player2.x, player2.y + PLAYER\_HEIGHT);

glEnd();

}

void drawDebris() {

for (int i = 0; i < NUM\_DEBRIS; ++i) {

glBegin(GL\_TRIANGLE\_FAN);

glColor3f(1.0f, 1.0f, 1.0f);

for (int j = 0; j < 360; ++j) {

float angle = j \* M\_PI / 180;

float x = debris[i].x + cos(angle) \* DEBRIS\_RADIUS;

float y = debris[i].y + sin(angle) \* DEBRIS\_RADIUS;

glVertex2f(x, y);

}

glEnd();

}

}

void updateShips() {

// Update player 1 position

if (player1.isUp && player1.y > 0) {

player1.y--;

if (!player1.reachedTop && player1.y == 0)

player1.reachedTop = true;

}

// Update player 2 position

if (player2.isUp && player2.y > 0) {

player2.y--;

if (!player2.reachedTop && player2.y == 0)

player2.reachedTop = true;

}

}

void updateDebris() {

for (int i = 0; i < NUM\_DEBRIS; ++i) {

if (debris[i].isGoingLeft)

debris[i].x--;

else

debris[i].x++;

// Wrap around the screen

if (debris[i].x < -DEBRIS\_RADIUS)

debris[i].x = SCREEN\_WIDTH + DEBRIS\_RADIUS;

else if (debris[i].x > SCREEN\_WIDTH + DEBRIS\_RADIUS)

debris[i].x = -DEBRIS\_RADIUS;

}

}

void checkCollisions() {

for (int i = 0; i < NUM\_DEBRIS; ++i) {

if (!player1.reachedTop && debris[i].y >= player1.y && debris[i].y <= player1.y + PLAYER\_HEIGHT) {

if (debris[i].x >= player1.x && debris[i].x <= player1.x + PLAYER\_WIDTH) {

// Collision with player 1

player1.y = SCREEN\_HEIGHT - PLAYER\_HEIGHT;

player1Collisions++;

}

}

if (!player2.reachedTop && debris[i].y >= player2.y && debris[i].y <= player2.y + PLAYER\_HEIGHT) {

if (debris[i].x >= player2.x && debris[i].x <= player2.x + PLAYER\_WIDTH) {

// Collision with player 2

player2.y = SCREEN\_HEIGHT - PLAYER\_HEIGHT;

player2Collisions++;

}

}

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawShips();

drawDebris();

// Display turn message

glColor3f(1.0f, 1.0f, 1.0f);

glRasterPos2f(20, SCREEN\_HEIGHT - 20); // Adjusted position

if (player1Turn) {

glutBitmapString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, (unsigned char \*)"Player 1's Turn (Press 'W' to move up)");

} else {

glutBitmapString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, (unsigned char \*)"Player 2's Turn (Press 'Up Arrow' to move up)");

}

// Display reached top message

if (player1.reachedTop) {

glRasterPos2f(20, SCREEN\_HEIGHT - 50); // Adjusted position

glutBitmapString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, (unsigned char \*)"Player 1 reached top!");

}

if (player2.reachedTop) {

glRasterPos2f(20, SCREEN\_HEIGHT - 80); // Adjusted position

glutBitmapString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, (unsigned char \*)"Player 2 reached top!");

}

// Display collision counts

glRasterPos2f(20, SCREEN\_HEIGHT - 110); // Adjusted position

char player1CollisionsStr[20];

char player2CollisionsStr[20];

sprintf(player1CollisionsStr, "Player 1 Collisions: %d", player1Collisions);

sprintf(player2CollisionsStr, "Player 2 Collisions: %d", player2Collisions);

glutBitmapString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, (unsigned char \*)player1CollisionsStr);

glRasterPos2f(20, SCREEN\_HEIGHT - 140); // Adjusted position

glutBitmapString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, (unsigned char \*)player2CollisionsStr);

// Determine winner

glRasterPos2f(20, SCREEN\_HEIGHT - 170); // Adjusted position

if (player1Collisions < player2Collisions) {

glutBitmapString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, (unsigned char \*)"Player 1 Wins!");

} else if (player2Collisions < player1Collisions) {

glutBitmapString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, (unsigned char \*)"Player 2 Wins!");

} else {

glutBitmapString(GLUT\_BITMAP\_TIMES\_ROMAN\_24, (unsigned char \*)"It's a Tie!");

}

glutSwapBuffers();

}

void update(int value) {

updateShips();

updateDebris();

checkCollisions();

glutPostRedisplay();

glutTimerFunc(16, update, 0);

}

void handleKeypress(unsigned char key, int x, int y) {

switch (key) {

case 'w':

if (player1Turn && !player1.reachedTop) {

player1.isUp = true;

player1.isDown = false;

}

break;

case 's':

if (player1Turn && !player1.reachedTop) {

player1.isDown = true;

player1.isUp = false;

}

break;

case 27:

exit(0);

break;

}

}

void handleKeyReleased(unsigned char key, int x, int y) {

switch (key) {

case 'w':

if (player1Turn) {

player1.isUp = false;

if (player1.reachedTop) {

player1Turn = false;

}

}

break;

case 's':

if (player1Turn) {

player1.isDown = false;

if (player1.reachedTop) {

player1Turn = false;

}

}

break;

}

}

void handleSpecialKeypress(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_UP:

if (!player1Turn && !player2.reachedTop) {

player2.isUp = true;

player2.isDown = false;

}

break;

case GLUT\_KEY\_DOWN:

if (!player1Turn && !player2.reachedTop) {

player2.isDown = true;

player2.isUp = false;

}

break;

}

}

void handleSpecialKeyReleased(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_UP:

if (!player1Turn) {

player2.isUp = false;

if (player2.reachedTop) {

player1Turn = true;

}

}

break;

case GLUT\_KEY\_DOWN:

if (!player1Turn) {

player2.isDown = false;

if (player2.reachedTop) {

player1Turn = true;

}

}

break;

}

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, SCREEN\_WIDTH, 0, SCREEN\_HEIGHT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(SCREEN\_WIDTH, SCREEN\_HEIGHT);

glutCreateWindow("Space Race");

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(handleKeypress);

glutKeyboardUpFunc(handleKeyReleased);

glutSpecialFunc(handleSpecialKeypress);

glutSpecialUpFunc(handleSpecialKeyReleased);

initShips();

initDebris();

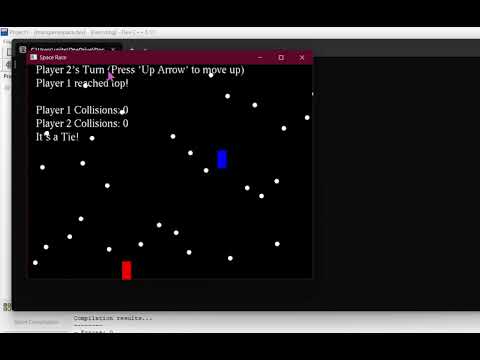
glutTimerFunc(16, update, 0);

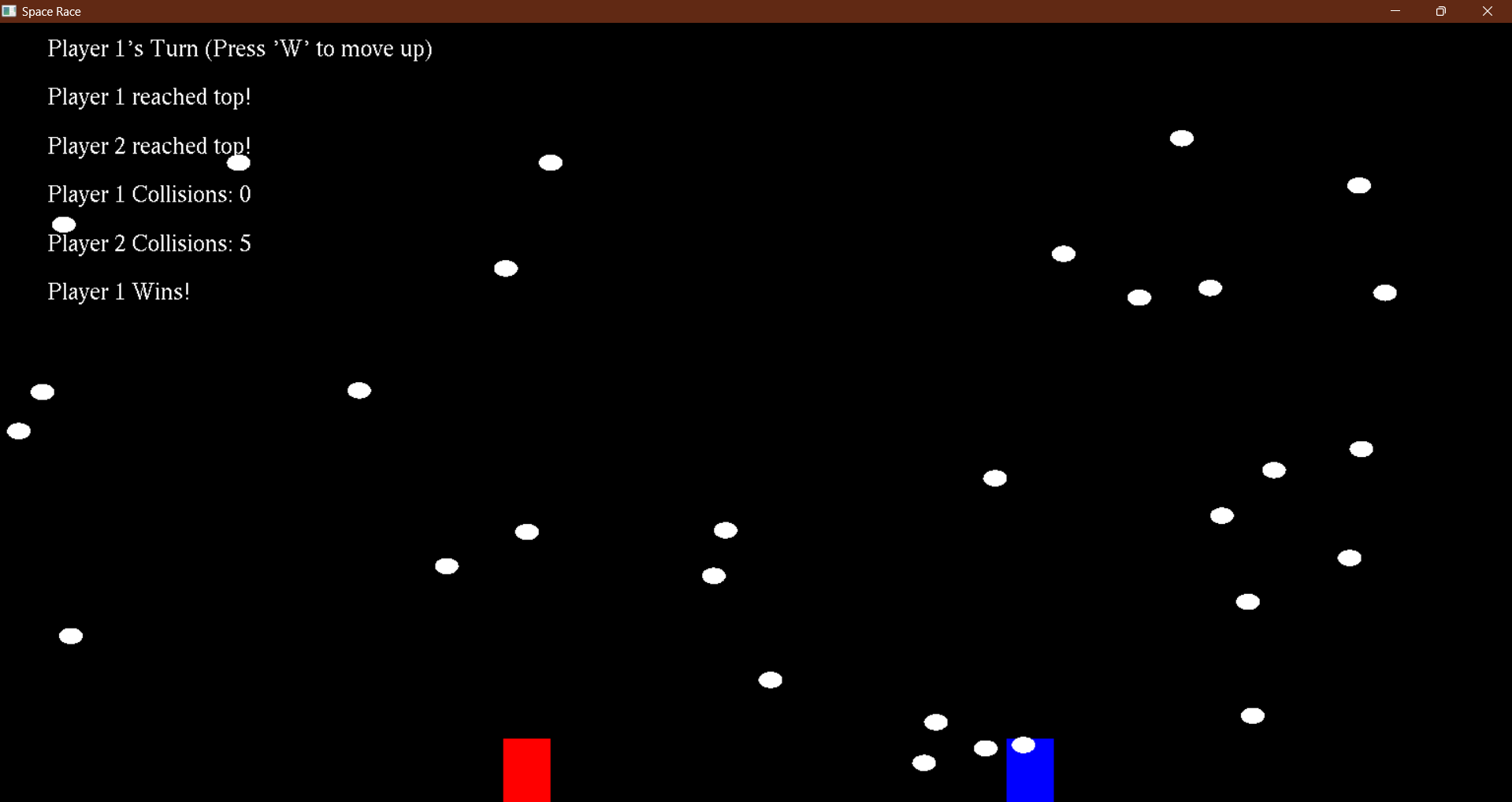
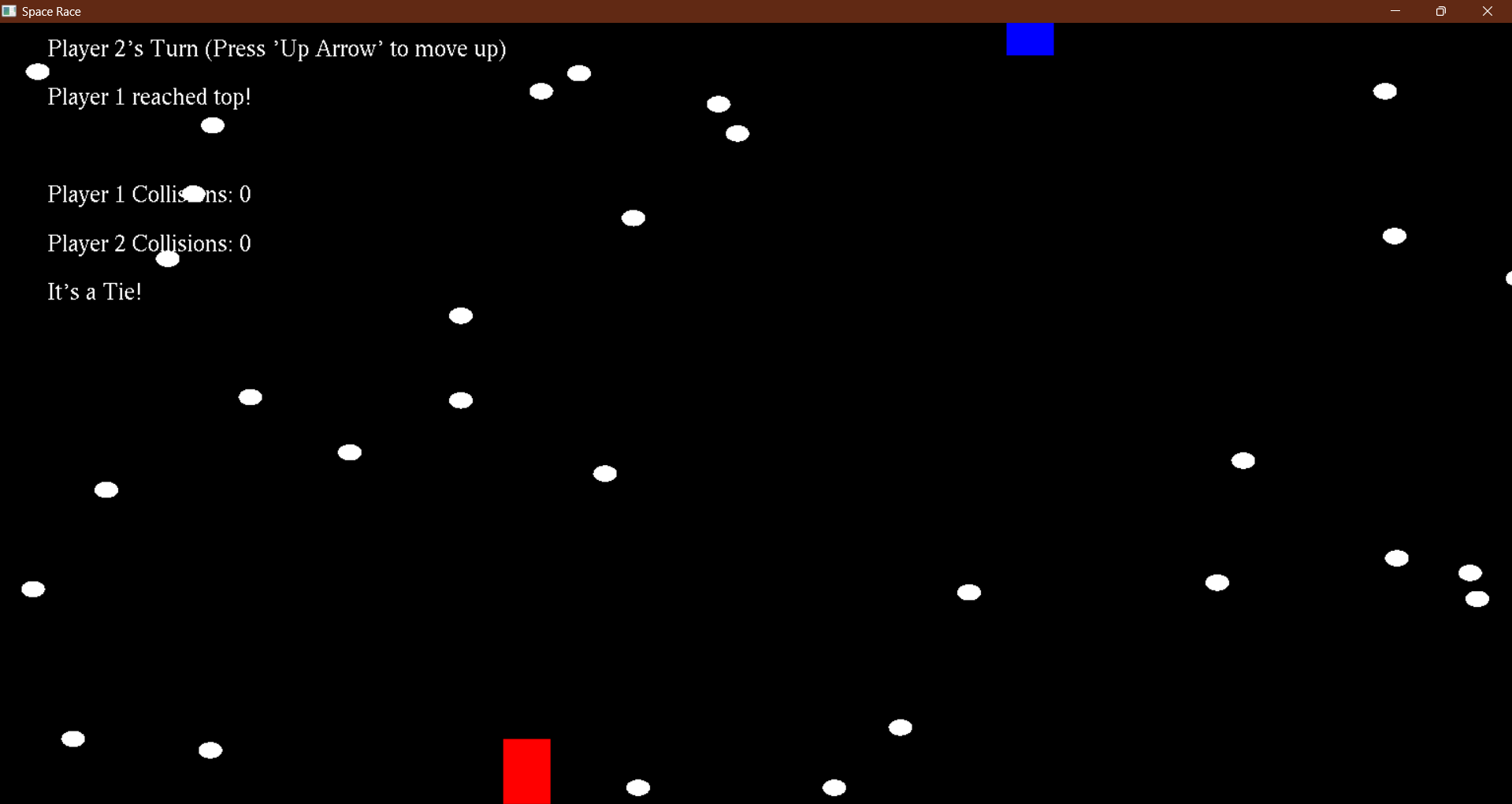
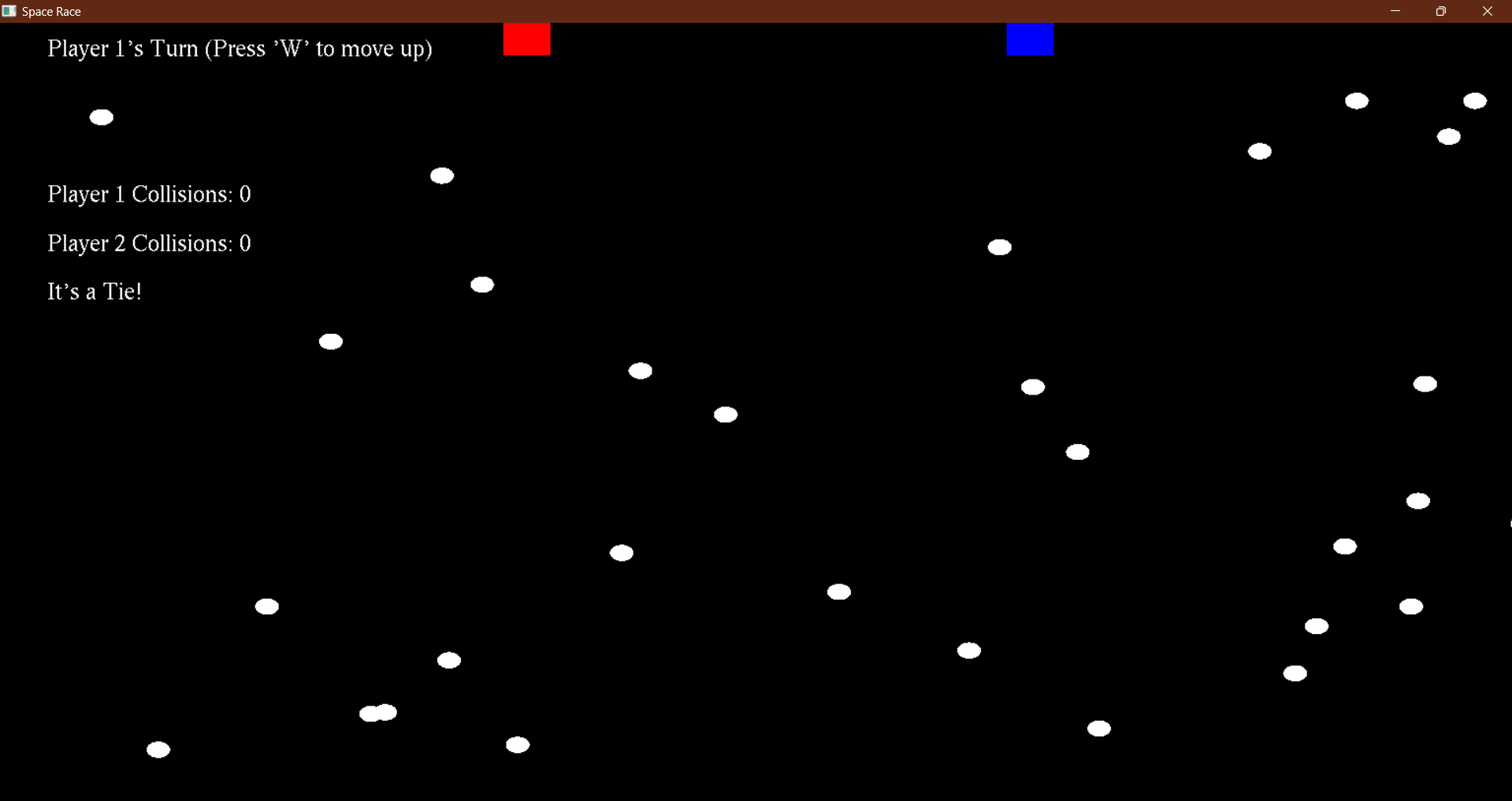
glutMainLoop();

return 0;

}

**Output:**

**[](https://www.youtube.com/embed/KHMUz-FG3BQ?feature=oembed)**



**Conclusion :**

In conclusion, the provided code demonstrates the implementation of a basic 2D game using OpenGL in C. It showcases fundamental concepts of game development, including graphics rendering, input handling, game state management, and collision detection. By leveraging OpenGL primitives and GLUT library functions, the code creates an interactive gaming experience where players control their ships to navigate through debris and reach the top of the screen.

Through this project, developers can gain insights into OpenGL programming techniques, game loop structure, and event-driven programming. Additionally, the code serves as a foundation for building more complex games by expanding upon the existing features, adding new gameplay mechanics, and enhancing visual elements.

Overall, the project offers a valuable learning opportunity for aspiring game developers to explore the basics of game development using OpenGL and C programming language. With further refinement and customization, developers can create engaging and immersive gaming experiences tailored to their creative vision and objectives.