Galaxy Wars

Project Report

Submitted for

Computer Graphics (UCS505)

Submitted by:

Geetansh Mohindru 102203718

Shrey Dhar Dubey 102383011

BE Third Year, COE

Sub-Group: 3C25

Submitted to:

Dr. Vishal Mehra

Computer Science & Engineering Department

Thapar Institute of Engineering & Technology, Patiala

EVEN SEM (Jan-May 2025)

TABLE OF CONTENTS

Chapter 1: Introduction

1.1 Project Overview:

Block - Buster is a two-dimensional, arcade-style shooting game developed using the OpenGL graphics library in C++. The primary objective of this project is to create an interactive and engaging graphical application that effectively demonstrates foundational concepts in computer graphics, such as object modelling, animation, real-time rendering, user input handling, collision detection, and state-based gameplay logic.

The game is set within a minimalistic yet visually dynamic virtual environment. Players control a shooter mounted on a horizontally movable spacecraft positioned at the bottom of the screen. The goal is to intercept and destroy descending enemy blocks before they reach the bottom boundary. These enemy blocks are categorized into two types:

- **Normal Enemies**: Small, red-coloured blocks that can be eliminated with a single bullet. Each destroyed normal enemy adds 1 point to the player's score.
- **Trojan Enemies**: Larger, purple-coloured blocks with a blinking visual effect. These enemies require two successful hits to be destroyed and award 3 points upon elimination.

Enemy blocks are spawned at random horizontal positions along the top of the screen and move vertically downward at a steady speed. The player navigates the spacecraft using the left and right arrow keys and fires bullets by pressing the spacebar.

Immediate visual feedback- such as colour changes, explosion effects, and blinking animations—is provided for events like firing, hitting an enemy, or destroying a block. These graphical effects are implemented using OpenGL primitives such as polygons and circles, giving the game a clean and responsive aesthetic.

Beyond its entertainment value, *Blockbuster* serves as a practical demonstration of how core computer graphics principles can be applied in the development of real-time interactive applications. The project reflects a comprehensive understanding of rendering pipelines, event-driven programming, and graphical system design, making it both an educational tool and a solid foundation for more advanced graphical projects.

1.2 Scope of the Project:

- ### Educational Scope:

  a) This project introduces fundamental graphics programming concepts using OpenGL, including rendering 2D shapes like triangles, rectangles, and spheres.
  b) It enhances understanding of real-time game mechanics such as collision detection, animation loops, and keyboard input handling.
  c) The addition of unique enemy behaviours and scoring logic fosters problem-solving and object-oriented design skills.
  d) Overall, it serves as a strong foundation for exploring computer graphics and game development.

- **Technical Scope:**

  a) This project uses OpenGL with C++ to create a 2D shooting game featuring triangular shooters, spherical bullets, and rectangular enemies.
  b) It covers key graphics programming techniques like shape rendering, transformations, frame control (40 FPS), and event-driven input.
  c) Collision logic and object spawning are implemented to simulate game dynamics.
  d) The code structure supports extensibility for advanced features like power-ups or level design.

- **Potential for Future Enhancements:**

  The Block Buster Game project provides a scalable foundation for future development, with a wide range of potential enhancements, including:

a) **Power-Ups & Special Weapons** – Add collectable items that boost fire rate or damage.
b) **Level Progression** – Introduce increasing difficulty or themed stages.
c) **Sound Effects & Music** – Integrate background music and collision sounds.
d) **High Score System** – Save and display top scores using file handling.

Chapter 2: Function Descriptions

S No.	Function Name	Function Description
1.	`drawRect()`	Used to draw elements like the player, enemies, and UI components.
2.	`drawCircle()`	Used to represent bullets and visual explosion effects.
3.	`drawText()`	Displays text on the screen using a specified GLUT bitmap font. Used for displaying UI labels like score and instructions.
4.	`drawExplosion()`	Renders a multi-layered explosion effect using concentric circles and randomly scattered spark points. Used when an enemy is destroyed.
5.	`drawPlayer()`	Renders the player's spaceship at the current position. Combines rectangles and triangles for detailed visual design.
6.	`drawEnemy()`	Draws an enemy. Displays different visuals for normal (red) and Trojan (blinking purple) enemies based on isTrojan and blinkTimer.
7.	`drawLegend()`	Displays a side panel showing game controls, scoring rules, enemy color coding, and contributor names. Enhances user understanding.
8.	`spawnEnemy()`	Spawns a new enemy at a random position along the top of the screen. Randomly decides whether the enemy is a normal or a Trojan type.
9.	`display()`	The main rendering function is called repeatedly by OpenGL. Draws all game elements: player, enemies, bullets, explosions, score, and the legend.
10.	`update(int value)`	Core game logic updater. Moves player, bullets, and enemies. Detects collisions, handles scoring, spawns' enemies, and updates explosion animations and blinking timers.
11.	`specialDown()`	Handles keyboard arrow key presses. Sets movement flags when the left or right keys are pressed to move the player.
12.	`specialUp()`	Handles keyboard arrow key release. Stops player movement when the arrow keys are released.
13.	`keyPressed()`	Handles keyboard events. Spacebar fires a bullet, and the ESC key exits the game.
14.	`main()`	Program's entry point. Initializes OpenGL and GLUT, sets up the window, callback functions, and starts the main rendering loop.

Chapter 3: Code Snippets

```
#include <glut.h>
#include <vector>
#include <ctime>
#include <string>
#include <cmath>
#include <cstdlib>

const int WIDTH = 800;
const int HEIGHT = 600;
const float PLAYER_WIDTH = 50.0f;
const float PLAYER_HEIGHT = 20.0f;
const float BULLET_RADIUS = 5.0f;
const float ENEMY_SIZE = 30.0f;
const float TROJAN_SIZE = 40.0f;

float playerX = WIDTH / 2.0f;
float bulletSpeed = 5.0f;
float enemySpeed = 1.0f;
int killCount = 0;

bool leftPressed = false, rightPressed = false;

struct Bullet {
    float x, y;
    bool alive;
};

struct Enemy {
    float x, y;
    bool alive;
    bool isTrojan;
    int hitCount;  // To count hits for Trojan enemies
    float blinkTimer;  // For controlling the blinking effect
};

struct Explosion {
    float x, y;
    int timer;
};

std::vector<Bullet> bullets;
std::vector<Enemy> enemies;
std::vector<Explosion> explosions;

void drawRect(float x, float y, float w, float h) {
    glBegin(GL_QUADS);
    glVertex2f(x, y);
    glVertex2f(x + w, y);
    glVertex2f(x + w, y + h);
    glVertex2f(x, y + h);
    glEnd();
}

void drawCircle(float x, float y, float radius) {
```

```
    glBegin(GL_POLYGON);
    for (int i = 0; i < 360; i += 10) {
        float angle = i * 3.14159f / 180;
        glVertex2f(x + cos(angle) * radius, y + sin(angle) * radius);
    }
    glEnd();
}

void drawText(float x, float y, const std::string& text, void* font =
GLUT_BITMAP_HELVETICA_12) {
    glRasterPos2f(x, y);
    for (char c : text)
        glutBitmapCharacter(font, c);
}

void drawExplosion(float x, float y) {
    glPushMatrix();
    glTranslatef(x, y, 0);

    // Outer flare (orange-red)
    glColor3f(1.0, 0.3, 0.0);
    drawCircle(0, 0, 18);

    // Mid flare (yellow)
    glColor3f(1.0, 1.0, 0.0);
    drawCircle(0, 0, 10);

    // Inner core (white)
    glColor3f(1.0, 1.0, 1.0);
    drawCircle(0, 0, 5);

    // Sparks (Random effects for explosion)
    glColor3f(1.0, 0.7, 0.1);
    glBegin(GL_POINTS);
    for (int i = 0; i < 30; ++i) {
        float angle = (rand() % 360) * 3.14159f / 180.0f;
        float dist = rand() % 10 + 5;
        glVertex2f(cos(angle) * dist, sin(angle) * dist);
    }
    glEnd();

    glPopMatrix();
}

void drawPlayer() {
    float centerX = playerX + PLAYER_WIDTH / 2;
    float y = 20;

    glColor3f(1.0f, 1.0f, 1.0f);
    drawRect(centerX - 5, y, 10, 40);

    glColor3f(0.0f, 1.0f, 1.0f);
    drawRect(centerX - 3, y + 20, 6, 10);

    glColor3f(0.8f, 0.2f, 0.2f);
    glBegin(GL_TRIANGLES);
```

```
    glVertex2f(centerX - 5, y + 15);
    glVertex2f(centerX - 20, y + 5);
    glVertex2f(centerX - 5, y);

    glVertex2f(centerX + 5, y + 15);
    glVertex2f(centerX + 20, y + 5);
    glVertex2f(centerX + 5, y);
    glEnd();
}

void drawEnemy(float x, float y, bool isTrojan, float blinkTimer) {
    float cx = x + (isTrojan ? TROJAN_SIZE : ENEMY_SIZE) / 2;
    float cy = y;

    if (isTrojan) {
        // Trojans: Bigger and flashing
        glColor3f(1.0f, 0.0f, 1.0f); // Bright purple
        if (int(blinkTimer) % 2 == 0)
            glColor3f(1.0f, 0.0f, 1.0f); // Flashing color (no color
change, you can modify if needed)
        else
            glColor3f(0.6f, 0.0f, 0.6f); // Darker purple
        drawRect(cx - 15, cy, 30, 40); // Trojan shape
    }
    else {
        // Normal Enemies: Smaller and steady color
        glColor3f(1.0f, 0.0f, 0.0f); // Red
        drawRect(cx - 10, cy, 20, 30); // Normal shape
    }
}

void drawLegend() {
    float boxX = WIDTH - 220;
    float boxY = HEIGHT - 260; // Adjusted Y position
    float boxW = 200;
    float boxH = 250; // Increased box height

    glColor3f(0.1f, 0.1f, 0.1f);
    drawRect(boxX, boxY, boxW, boxH);

    glColor3f(0.4f, 1.0f, 0.6f);
    glBegin(GL_LINE_LOOP);
    glVertex2f(boxX, boxY);
    glVertex2f(boxX + boxW, boxY);
    glVertex2f(boxX + boxW, boxY + boxH);
    glVertex2f(boxX, boxY + boxH);
    glEnd();

    glColor3f(0.4f, 1.0f, 0.6f);
    drawText(boxX + 10, boxY + 235, " ~ Computer Graphics Project ~ ");
    drawText(boxX + 10, boxY + 220, " ~~~~~ GALAXY WARS ~~~~~ ");

    glColor3f(1, 1, 1);
    drawText(boxX + 10, boxY + 195, "LEFT/RIGHT : Move ship");
    drawText(boxX + 10, boxY + 180, "SPACE : Shoot bullet");
    drawText(boxX + 10, boxY + 165, "+1 Point for Normal Enemy");
```

```
    drawText(boxX + 10, boxY + 150, "+3 Points for Trojan Enemy");
    drawText(boxX + 10, boxY + 135, "Don't let them pass!");
    drawText(boxX + 10, boxY + 120, "Trojan Enemy: Purple");
    drawText(boxX + 10, boxY + 105, "Normal Enemy: Red");

    glColor3f(0.6f, 0.9f, 1.0f);
    drawText(boxX + 10, boxY + 85, "102203718-Geetansh Mohindru");
    drawText(boxX + 10, boxY + 70, "102383011-Shrey Dhar Dubey");
    drawText(boxX + 10, boxY + 55, "3C25 / 3CO10");

    glColor3f(0.7f, 1.0f, 0.9f);
    drawText(boxX + 10, boxY + 40, "Submitted to: Dr. Vishal Mehra");
}

void spawnEnemy() {
    Enemy e;
    e.x = float(rand() % (WIDTH - int(ENEMY_SIZE)));
    e.y = HEIGHT;
    e.alive = true;
    e.isTrojan = rand() % 2 == 0;
    e.hitCount = 0;  // Initial hits count
    e.blinkTimer = 0.0f; // Initially not blinking
    enemies.push_back(e);
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    drawPlayer();

    glColor3f(1, 1, 0);
    for (auto& b : bullets)
        if (b.alive) drawCircle(b.x, b.y, BULLET_RADIUS);

    for (auto& e : enemies)
        if (e.alive) drawEnemy(e.x, e.y, e.isTrojan, e.blinkTimer);

    for (auto& ex : explosions)
        if (ex.timer > 0) drawExplosion(ex.x, ex.y);

    glColor3f(1, 1, 1);
    drawText(10, HEIGHT - 30, "Kills: " + std::to_string(killCount),
GLUT_BITMAP_HELVETICA_18);

    drawLegend();  // Draw the legend

    glutSwapBuffers();
}

void update(int value) {
    if (leftPressed && playerX > 0) playerX -= 5;
    if (rightPressed && playerX < WIDTH - PLAYER_WIDTH) playerX += 5;

    for (auto& b : bullets) {
        if (b.alive) b.y += bulletSpeed;
        if (b.y > HEIGHT) b.alive = false;
```

```
    }

    for (auto& e : enemies) {
        if (e.alive) e.y -= enemySpeed;
        if (e.y < 0) e.alive = false;
    }

    // Checking for collisions between bullets and enemies
    for (auto& b : bullets) {
        for (auto& e : enemies) {
            if (b.alive && e.alive && b.x > e.x && b.x < e.x + (e.isTrojan
? TROJAN_SIZE : ENEMY_SIZE) &&
                b.y > e.y && b.y < e.y + (e.isTrojan ? TROJAN_SIZE :
ENEMY_SIZE)) {
                b.alive = false;
                if (e.isTrojan) {
                    e.hitCount++; // Increase hit count for Trojan
                    if (e.hitCount >= 2) {
                        e.alive = false;
                        killCount += 3; // +3 points for Trojan
                        explosions.push_back({ e.x + (e.isTrojan ?
TROJAN_SIZE : ENEMY_SIZE) / 2, e.y + (e.isTrojan ? TROJAN_SIZE :
ENEMY_SIZE) / 2, 15 });
                    }
                }
                else {
                    e.alive = false;
                    killCount += 1; // +1 point for regular enemy
                    explosions.push_back({ e.x + ENEMY_SIZE / 2, e.y +
ENEMY_SIZE / 2, 15 });
                }
            }
        }
    }

    // Update Trojan blink timer
    for (auto& e : enemies) {
        if (e.isTrojan && e.hitCount > 0) {
            e.blinkTimer += 0.1f;
            if (e.blinkTimer > 2.0f) e.blinkTimer = 0.0f; // Reset blink
timer
        }
    }

    // Remove explosions after they have been shown for a brief period
    for (auto it = explosions.begin(); it != explosions.end();) {
        if (--it->timer <= 0)
            it = explosions.erase(it); // Remove explosion after timer
reaches 0
        else
            ++it;
    }

    if (rand() % 100 < 2) spawnEnemy();

    glutPostRedisplay();
```

```
    glutTimerFunc(16, update, 0);
}

void specialDown(int key, int, int) {
    if (key == GLUT_KEY_LEFT) leftPressed = true;
    if (key == GLUT_KEY_RIGHT) rightPressed = true;
}

void specialUp(int key, int, int) {
    if (key == GLUT_KEY_LEFT) leftPressed = false;
    if (key == GLUT_KEY_RIGHT) rightPressed = false;
}

void keyPressed(unsigned char key, int, int) {
    if (key == ' ') {
        bullets.push_back({ playerX + PLAYER_WIDTH / 2, 40, true });
    }
    if (key == 27)
        exit(0);
}

int main(int argc, char** argv) {
    srand(time(0));

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(WIDTH, HEIGHT);
    glutCreateWindow("Galaxy Wars");

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0f, WIDTH, 0.0f, HEIGHT, -1.0f, 1.0f);

    glutDisplayFunc(display);
    glutSpecialFunc(specialDown);
    glutSpecialUpFunc(specialUp);
    glutKeyboardFunc(keyPressed);
    glutTimerFunc(25, update, 0);

    glutMainLoop();

    return 0;
}
```

Chapter 4: User Interface

4.1 Screenshot

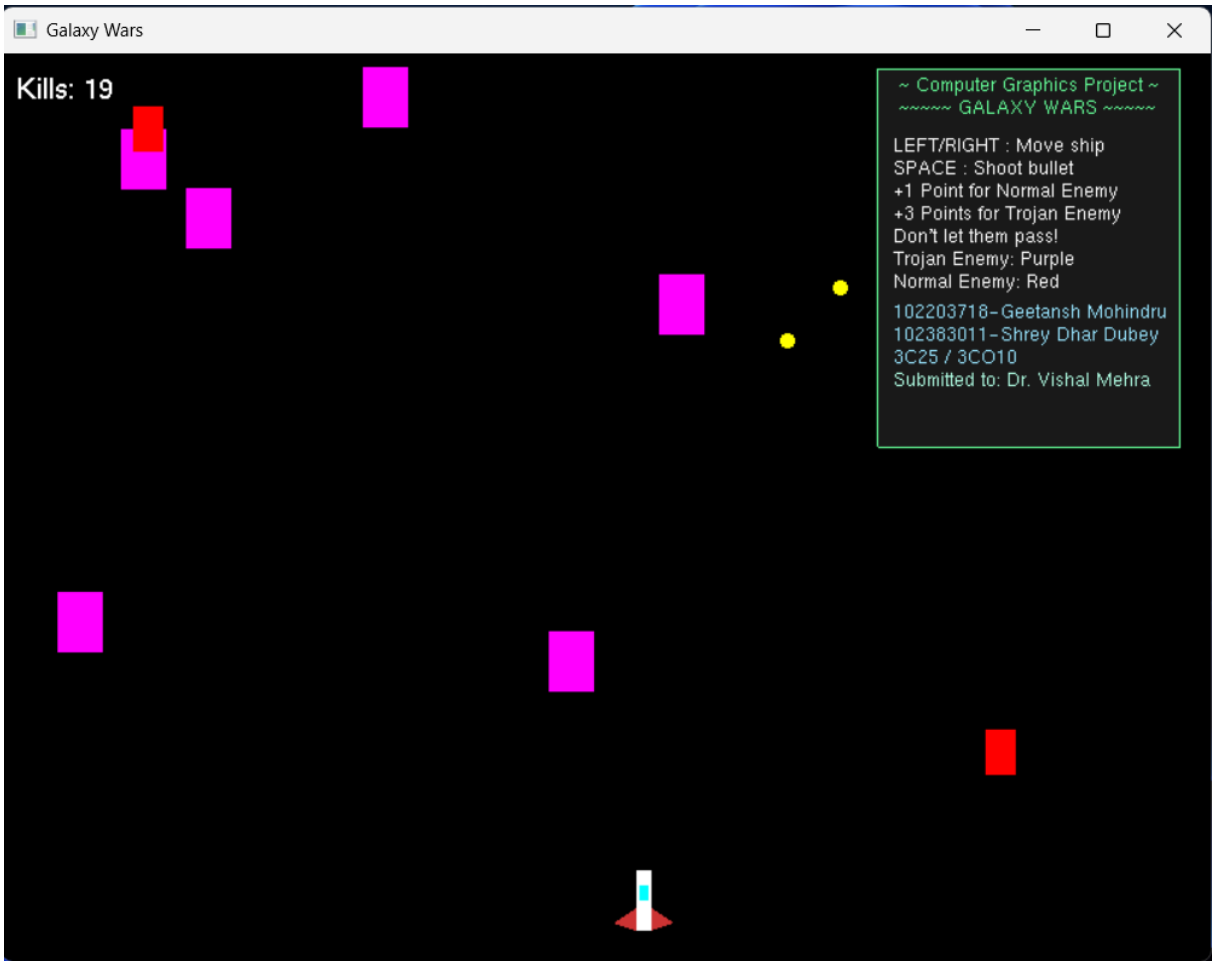

Fig. 1 Depiction of the Galaxy Wars

4.2 Game Features

- **Player Control:** Move left and right, shoot upward bullets.
- **Enemy Types:**
  - 🔴 **Normal Enemies:** Red, die in one hit, +1 point.
  - 🟣 **Trojan Enemies:** Purple, need 2 hits, blink on damage, +3 points.
- 💥 **Explosions:** Colourful visual effects on enemy destruction.
- 📊 **Score Tracking:** Real-time kill count display.
- 🗂️ **In-game Legend:** Shows controls and contributors.

4.3 Game Controls

Key	Action
⬅️ / ➡️	Move Ship Left/Right
SPACE	Shoot Bullet
ESC	Exit Game

Chapter 5: References

1. OpenGL Programming Guide (The Red Book), 9th Edition, Addison-Wesley, 2013. A comprehensive resource for understanding the modern OpenGL rendering pipeline and techniques.

2. **GLUT API Documentation:** https://www.opengl.org/resources/libraries/glut/ Official documentation for GLUT (OpenGL Utility Toolkit), used for handling windows, keyboard input, and rendering.

3. **Learn OpenGL:** https://learnopengl.com. A popular modern tutorial website for learning OpenGL concepts and graphics programming in C++.

4. **Geeks for Geeks:** Computer Graphics in C++, https://www.geeksforgeeks.org/computer-graphics/ .Helpful tutorials and code examples on implementing basic graphics features in C++.

5. **CPP Reference:** https://en.cppreference.com C++ standard library reference used for syntax and STL functions.

6. **Stack Overflow:** https://stackoverflow.com. Community support and troubleshooting platform used for resolving programming issues and optimizations.