Computer Graphics (UCS505)

Space Shooting Game

Branch

B.E. 3rd Year – CSE

Submitted By –

- **Parijaat Gaur (102216104)**
- **Rachit Sinha (102216087)**

Submitted To –

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

INTRODUCTION

Project Overview:

This project is a two-player 2D space shooting game developed as a part of the Computer Graphics laboratory coursework. The game is designed using OpenGL and the GLUT (OpenGL Utility Toolkit) library, focusing on applying fundamental concepts of computer graphics in an interactive and engaging environment.

The core objective of this project is to demonstrate practical implementation of graphics theory through:

- Manual object modeling using OpenGL primitives,
- 2D transformations (translation, scaling, rotation) to simulate motion,
- Custom raster text rendering using bitmap fonts,
- Interactive UI design implemented through OpenGL's event-driven architecture.

Each spaceship and its components – including the alien character, dome, lights, and weapons – are constructed using low-level OpenGL drawing routines (e.g., glBegin(GL_POLYGON), glVertex2f, and glutSolidSphere). These components collectively demonstrate knowledge of hierarchical modeling, color interpolation, and composite transformations.

Screen navigation (intro, menu, game, instructions, and game over) is handled using a finite state machine, enabling event-based view transitions. Player inputs, both keyboard and mouse, are captured using GLUT's callback functions and mapped to geometric transformations applied to the game objects.

Real-time collision detection – a critical gameplay element – is implemented using analytical geometry techniques (line-circle intersection using discriminants), showcasing the integration of mathematical modeling within the graphics pipeline.

Through this project, essential concepts such as object-space rendering, viewport-to-world coordinate mapping, and double buffering for smooth frame updates are explored in practice, solidifying the student's understanding of interactive graphics systems and OpenGL-based 2D rendering.

Scope of the Project:

The Space Shooting Game has been developed to explore and apply key principles of 2D computer graphics through OpenGL. The project emphasizes manual implementation of object construction, animation, and interaction using foundational OpenGL functions and techniques, without the use of external rendering or physics engines.

The entire scene, including spaceships, characters, UI elements, and lasers, is rendered using procedural graphics – constructed from first principles using OpenGL primitives such as GL_POLYGON, GL_LINES, and glutSolidSphere. These elements are transformed and animated in response to user input using affine transformations including translation and scaling, demonstrating an understanding of local and global coordinate systems.

Player movement, shooting, and in-game animations are implemented through dynamic updates to object positions within the OpenGL modelview matrix. User interactions are handled using GLUT's event callbacks, allowing real-time input response, menu navigation, and shooting mechanics. Mouse coordinates are mapped from screen space to the orthographic projection system to manage GUI events.

The visual interface of the application spans multiple display modes – intro, menu, instructions, gameplay, and game over – implemented using a state-driven rendering logic. This is achieved through a structured view controller that maps user input and application state to the appropriate rendering routines.

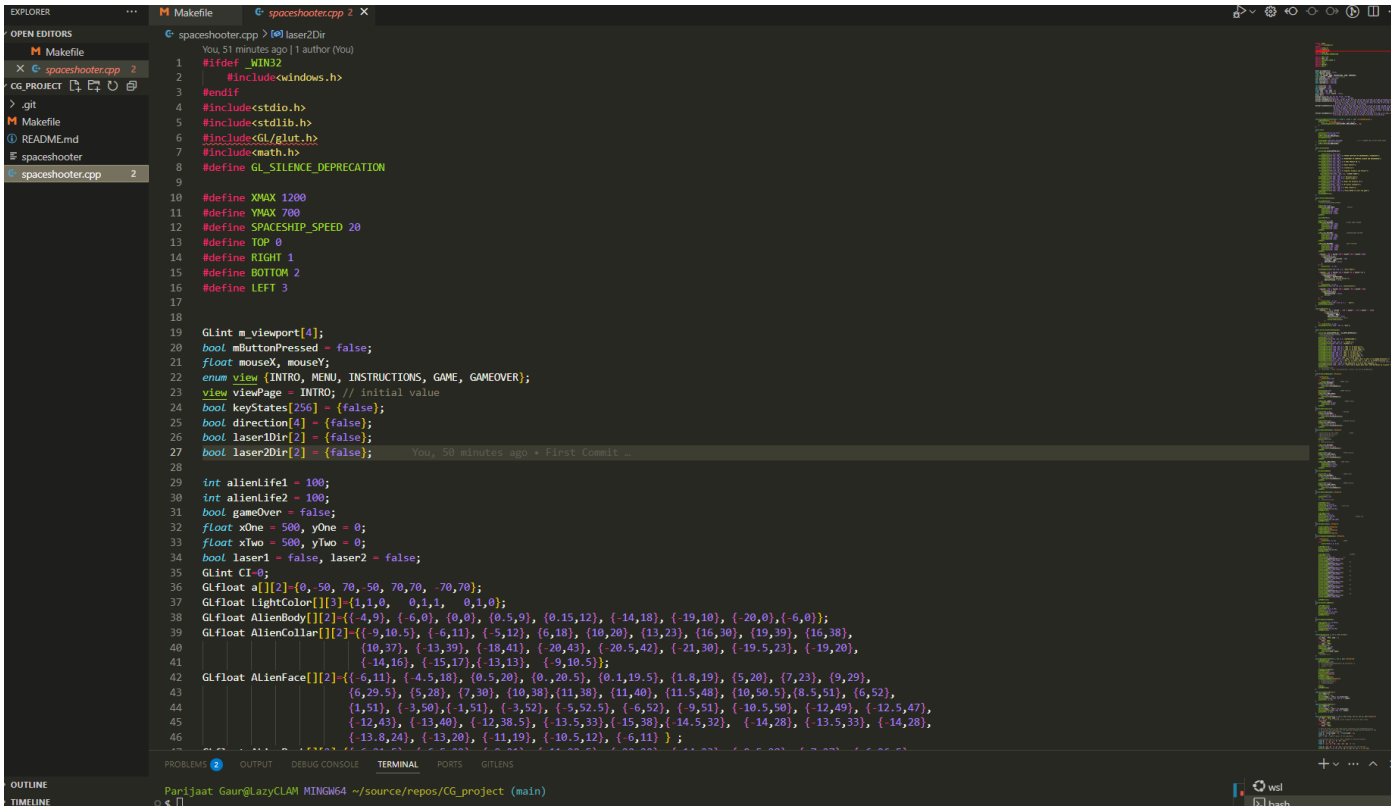
Laser interactions involve collision detection using analytical geometry, where line equations are solved for intersection with circular spaceship boundaries, exemplifying geometric modeling applied to interactive graphics.

By constraining all graphical behavior to OpenGL-based rendering, event handling, and logic, the project remains focused on demonstrating concepts such as object-space construction, viewport handling, coordinate mapping, and interactive rendering pipelines.

USER DEFINED FUNCTIONS

Function Name	Function Description
displayRasterText(x, y, z, text)	Renders text on screen using GLUT bitmap fonts. Demonstrates manual control over raster position and string rendering, contributing to UI construction within the OpenGL pipeline.
introScreen()	Draws the introductory splash screen with stylized raster text. Implements a static layout using low-level rendering calls to convey project context.
startScreenDisplay()	Renders the main menu interface, including interactive button areas. Detects mouse input and uses simple geometric hit-testing to transition game states.
instructionsScreenDisplay()	Displays the control layout and objectives for both players. Focuses on structured raster text layout and contextual game information using OpenGL's immediate mode.
gameScreenDisplay()	Core game rendering function. Renders both player spaceships, lasers, and health bars. Applies 2D transformations and conditionally invokes collision detection routines.
displayGameOverMessage()	Displays the win/loss message based on game state. A lightweight rendering function, but completes the state-driven display flow.
DrawAlienBody(isPlayer1)	Constructs the alien's body polygonally from vertex arrays. Demonstrates procedural modeling and color state manipulation.
DrawAlienCollar()	Renders a decorative collar using vertex strip and polygon modes, showcasing complex shape rendering and layering.
DrawAlienFace(isPlayer1)	Constructs the alien face as a compound polygon. Uses outline drawing and composite shape techniques.
DrawAlienBeak()	Draws the alien's beak using detailed polygon and line strip drawing. Highlights precision modeling using vertex sequences.
DrawAlienEyes(isPlayer1)	Renders two semi-elliptical eyes using glutSolidSphere with scaling and transformation — a practical example of non-uniform scaling in OpenGL.
DrawAlien(isPlayer1)	Composite function that assembles all alien parts. Demonstrates hierarchical modeling by combining local transformations.
DrawSpaceshipBody(isPlayer1)	Renders the spaceship base using solid spheres, lights, and color cycling. Illustrates transformation stacking and color interpolation.
DrawSteeringWheel()	Adds detail to the spaceship interior using wireframe spheres. An example of enhancing realism using simple geometric primitives.
DrawSpaceshipDoom()	Draws the transparent dome of the spaceship using alpha blending and transformation. Demonstrates translucent rendering and visual layering.
DrawLaser(x, y, dir[])	Draws a directional laser beam using GL_LINES, dynamically oriented based on player input. Highlights simple real-time animation logic.
SpaceshipCreate(x, y, isPlayer1)	Central rendering routine for placing and drawing a player's spaceship. Combines translation, rotation, and scaled rendering.
DisplayHealthBar1() / DisplayHealthBar2()	Show current health of players using dynamic raster text. Incorporates formatted string rendering into gameplay UI.
checkLaserContact(x, y, dir[], xp, yp, player1)	Performs geometric collision detection using line-circle intersection via discriminant analysis — a direct application of computational geometry in graphics.
display()	Master display callback. Selectively renders different views based on current state, showcasing state-driven rendering architecture. Also resets modelview after each frame.
keyOperations()	Polls key states and applies corresponding transformations to player positions. Encodes movement and action logic tied directly to OpenGL rendering effects.
passiveMotionFunc(x, y)	Translates screen-space mouse movement into orthographic coordinates. Demonstrates viewport mapping and continuous interaction handling.
mouseClick(button, state, x, y)	Sets input flags based on mouse clicks. Used in conjunction with position tracking to trigger view transitions.
keyPressed(key, x, y) / keyReleased(key, x, y)	Updates the keyStates array to track pressed keys, enabling real-time input control of game objects.

CODE SNIPPETS



```
#ifdef _WIN32
    #include<windows.h>
#endif
#include<stdio.h>
#include<stdlib.h>
#include<GL/glut.h>
#include<math.h>
#define GL_SILENCE_DEPRECATION

#define XMAX 1200
#define YMAX 700
#define SPACESHIP_SPEED 20
#define TOP 0
#define RIGHT 1
#define BOTTOM 2
#define LEFT 3


GLint m_viewport[4];
bool mButtonPressed = false;
float mouseX, mouseY;
enum view {INTRO, MENU, INSTRUCTIONS, GAME, GAMEOVER};
view viewPage = INTRO; // initial value
bool keyStates[256] = {false};
bool direction[4] = {false};
bool laser1Dir[2] = {false};
bool laser2Dir[2] = {false};

int alienLife1 = 100;
int alienLife2 = 100;
bool gameOver = false;
float xOne = 500, yOne = 0;
float xTwo = 500, yTwo = 0;
bool laser1 = false, laser2 = false;
GLint CI=0;
GLfloat a[][2]={0,-50, 70,-50, 70,70, -70,70};
GLfloat LightColor[][3]={1,1,0,   0,1,1,   0,1,0};
GLfloat AlienBody[][2]={{-4,9}, {-6,0}, {0,0}, {0.5,9}, {0.15,12}, {-14,18}, {-19,10}, {-20,0},{-6,0}};
GLfloat AlienCollar[][2]={{-9,10.5}, {-6,11}, {-5,12}, {6,18}, {10,20}, {13,23}, {16,30}, {19,39}, {16,38},
                          {10,37}, {-13,39}, {-18,41}, {-20,43}, {-20.5,42}, {-21,30}, {-19.5,23}, {-19,20},
                          {-14,16}, {-15,17},{-13,13},  {-9,10.5}};
GLfloat AlienFace[][2]={{-6,11}, {-4.5,18}, {0.5,20}, {0.,20.5}, {0.1,19.5}, {1.8,19}, {5,20}, {7,23}, {9,29},
                        {6,29.5}, {5,28}, {7,30}, {10,38},{11,38}, {11,40}, {11.5,48}, {10,50.5},{8.5,51}, {6,52},
                        {1,51}, {-3,50},{-1,51}, {-3,52}, {-5,52.5}, {-6,52}, {-9,51}, {-10.5,50}, {-12,49}, {-12.5,47},
                        {-12,43}, {-13,40}, {-12,38.5}, {-13.5,33},{-15,38},{-14.5,32},  {-14,28}, {-13.5,33}, {-14,28},
                        {-13.8,24}, {-13,20}, {-11,19}, {-10.5,12}, {-6,11} } ;
```

(Mentioning 5-6 significant user-defined functions in this report. The rest can be found in separately provided source code. Next page onward…)

```
void instructionsScreenDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    //SetDisplayMode(MENU_SCREEN);
    //colorBackground();
    glColor3f(1, 0, 0);
    displayRasterText(-900 ,550 ,0.4 ,"INSTRUCTIONS");
    glColor3f(1, 0, 0);
    displayRasterText(-1000 ,400 ,0.4 ,"PLAYER 1");
    displayRasterText(200 ,400 ,0.4 ,"PLAYER 2");
    glColor3f(1, 1, 1);
    displayRasterText(-1100 ,300 ,0.4 ,"Key 'w' to move up.");
    displayRasterText(-1100 ,200 ,0.4 ,"Key 's' to move down.");
    displayRasterText(-1100 ,100 ,0.4 ,"Key 'd' to move right.");
    displayRasterText(-1100 ,0 ,0.4 ,"Key 'a' to move left.");
    displayRasterText(100 ,300 ,0.4 ,"Key 'i' to move up.");
    displayRasterText(100 ,200 ,0.4 ,"Key 'k' to move down.");
    displayRasterText(100 ,100 ,0.4 ,"Key 'j' to move right.");
    displayRasterText(100 ,0 ,0.4 ,"Key 'l' to move left.");
    displayRasterText(-1100 ,-100 ,0.4 ,"Key 'c' to shoot, Use 'w' and 's' to change direction.");
    displayRasterText(100 ,-100 ,0.4 ,"Key 'm' to shoot, Use 'i' and 'k' to change direction.");
    //displayRasterText(-1100 ,-100 ,0.4 ,"The packet can be placed only when 's' is pressed before.");
    displayRasterText(-1100, -300,0.4,"The Objective is to kill your opponent.");
    displayRasterText(-1100 ,-370 ,0.4 ,"Each time a player gets shot, LIFE decreases by 5 points.");
    backButton();
    //if(previousScreen)
    //  nextScreen = false ,previousScreen = false; //as set by backButton()
}

void DrawAlienBody(bool isPlayer1)
{
    if(isPlayer1)
        glColor3f(0,1,0);
    else
        glColor3f(1,1,0);       //BODY color
    glBegin(GL_POLYGON);
    for(int i=0;i<=8;i++)
        glVertex2fv(AlienBody[i]);
    glEnd();

    glColor3f(0,0,0);           //BODY Outline
    glLineWidth(1);
    glBegin(GL_LINE_STRIP);
    for(int i=0;i<=8;i++)
        glVertex2fv(AlienBody[i]);
    glEnd();

    glBegin(GL_LINES);                //BODY effect
        glVertex2f(-13,11);
        glVertex2f(-15,9);
    glEnd();
}
```

```
void DrawAlienBeak()
{
    glColor3f(1,1,0);                //BEAK color
    glBegin(GL_POLYGON);
    for(int i=0;i<=14 ;i++)
        glVertex2fv(ALienBeak[i]);
    glEnd();

    glColor3f(0,0,0);                //BEAK outline
    glBegin(GL_LINE_STRIP);
    for(int i=0;i<=14 ;i++)
        glVertex2fv(ALienBeak[i]);
    glEnd();
}
void DrawAlienEyes(bool isPlayer1)
{
    // if(isPlayer1)
    glColor3f(0,1,1);
    // else
    //  glColor3f(0,0,0);

    glPushMatrix();
    glRotated(-10,0,0,1);
    glTranslated(-6,32.5,0);       //Left eye
    glScalef(2.5,4,0);
    glutSolidSphere(1,20,30);
    glPopMatrix();

    glPushMatrix();
    glRotated(-1,0,0,1);
    glTranslated(-8,36,0);                              //Right eye
    glScalef(2.5,4,0);
    glutSolidSphere(1,100,100);
    glPopMatrix();
}
```

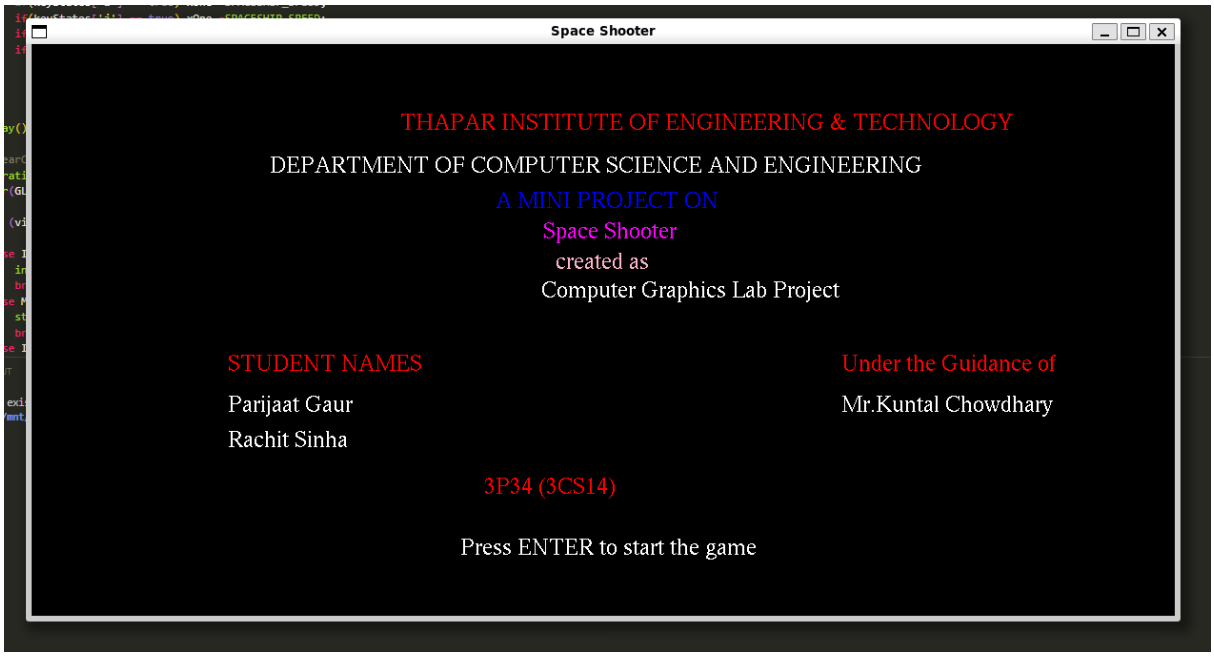
```
void DrawSpaceshipBody(bool isPlayer1)       You, 59 minutes ago • First Commit
{
    if(isPlayer1)
        glColor3f(1, 0, 0);          //BASE
    else
        glColor3f(0.5, 0, 0.5);

    glPushMatrix();
    glScalef(70,20,1);
    glutSolidSphere(1,50,50);
    glPopMatrix();

    glPushMatrix();                          //LIGHTS
    glScalef(3,3,1);
    glTranslated(-20,0,0);           //1
    glColor3fv(LightColor[(CI+0)%3]);
    glutSolidSphere(1,1000,1000);
    glTranslated(5,0,0);                     //2
    glColor3fv(LightColor[(CI+1)%3]);
    glutSolidSphere(1,1000,1000);
    glTranslated(5,0,0);                     //3
    glColor3fv(LightColor[(CI+2)%3]);
    glutSolidSphere(1,1000,1000);
    glTranslated(5,0,0);                     //4
    glColor3fv(LightColor[(CI+0)%3]);
    glutSolidSphere(1,1000,1000);
    glTranslated(5,0,0);                     //5
    glColor3fv(LightColor[(CI+1)%3]);
    glutSolidSphere(1,1000,1000);
    glTranslated(5,0,0);                     //6
    glColor3fv(LightColor[(CI+2)%3]);
    glutSolidSphere(1,1000,1000);
    glTranslated(5,0,0);                     //7
    glColor3fv(LightColor[(CI+0)%3]);
    glutSolidSphere(1,1000,1000);
    glTranslated(5,0,0);                     //8
    glColor3fv(LightColor[(CI+1)%3]);
    glutSolidSphere(1,1000,1000);
    glTranslated(5,0,0);                     //9
    glColor3fv(LightColor[(CI+2)%3]);
    glutSolidSphere(1,1000,1000);

    glPopMatrix();
}
void DrawSteeringWheel()
{
    glPushMatrix();
    glLineWidth(3);
    glColor3f(0.20,0.,0.20);
    glScalef(7,4,1);
    glTranslated(-1.9,5.5,0);
    glutWireSphere(1,8,8);
    glPopMatrix();

}
```

```
void DrawLaser(int x, int y, bool dir[]) {
    //glPushMatrix();
    int xend = -XMAX, yend = y;
    if(dir[0])
        yend = YMAX;
    else if(dir[1])
        yend = -YMAX;
    glLineWidth(5);
    glColor3f(1, 0, 0);
    glBegin(GL_LINES);
        glVertex2f(x, y);
        glVertex2f(xend, yend);
    glEnd();
    //glPopMatrix();
}

void SpaceshipCreate(int x, int y, bool isPlayer1){
    glPushMatrix();
    glTranslated(x,y,0);
    // if(!checkIfSpaceShipIsSafe() && alienLife1 ){
    //  alienLife1-=10;
    //  xStart -= 23;
    // }
    DrawSpaceshipDoom();
    glPushMatrix();
    glTranslated(4,19,0);
    DrawAlien(isPlayer1);
    glPopMatrix();
    DrawSteeringWheel();
    DrawSpaceshipBody(isPlayer1);
    // DrawSpaceShipLazer();
    // if(mButtonPressed) {
    //  DrawLazerBeam();
    // }
    glEnd();
    glPopMatrix();
}

void DisplayHealthBar1() {
    char temp1[40];
    glColor3f(1 ,1 ,1);
    sprintf(temp1,"  LIFE = %d",alienLife1);
    displayRasterText(-1100 ,600 ,0.4 ,temp1);
    glColor3f(1 ,0 ,0);
}

void DisplayHealthBar2() {
    char temp2[40];
    glColor3f(1 ,1 ,1);
    sprintf(temp2,"  LIFE = %d",alienLife2);
    displayRasterText(800 ,600 ,0.4 ,temp2);
    glColor3f(1 ,0 ,0);
}
```

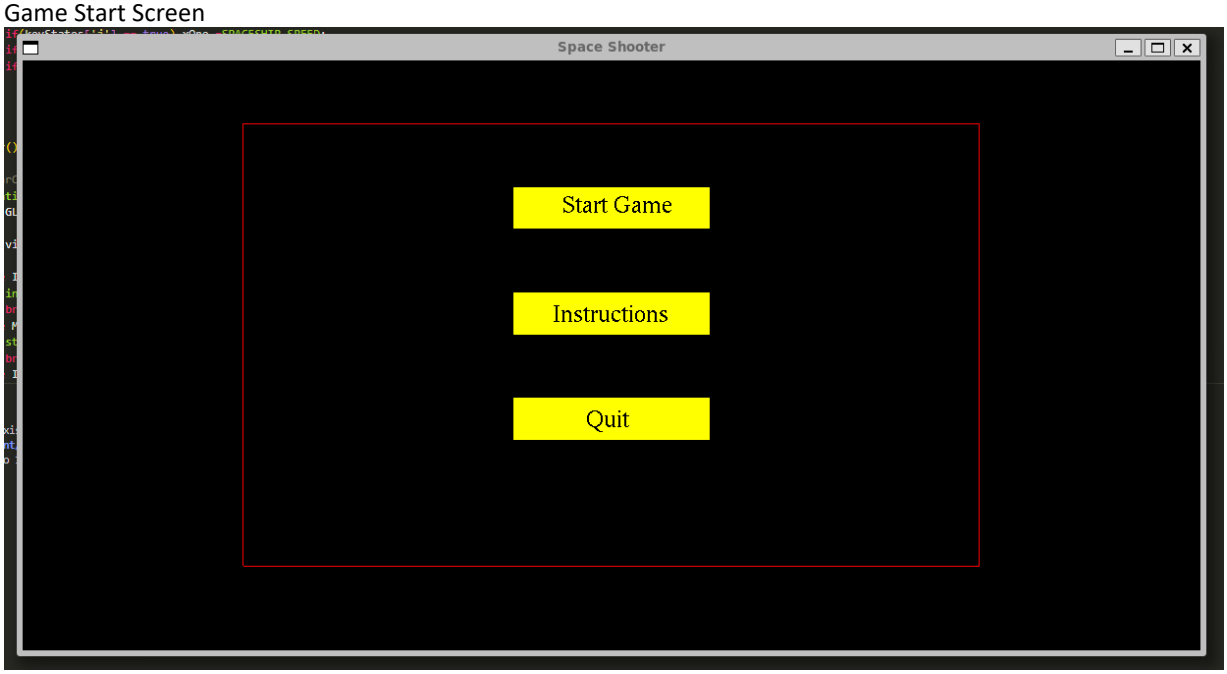
```
void display()
{
    //glClearColor(, 0 , 0, 1);
    keyOperations();
    glClear(GL_COLOR_BUFFER_BIT);

    switch (viewPage)
    {
        case INTRO:
            introScreen();
            break;
        case MENU:
            startScreenDisplay();
            break;
        case INSTRUCTIONS:
            instructionsScreenDisplay();
            break;
        case GAME:
            gameScreenDisplay();
            //reset scaling values
            glScalef(1/2 ,1/2 ,0);
            break;
        case GAMEOVER:
            displayGameOverMessage();
            startScreenDisplay();
            break;
    }

    glFlush();
    glLoadIdentity();
    glutSwapBuffers();
}
```

SCREENSHOTS

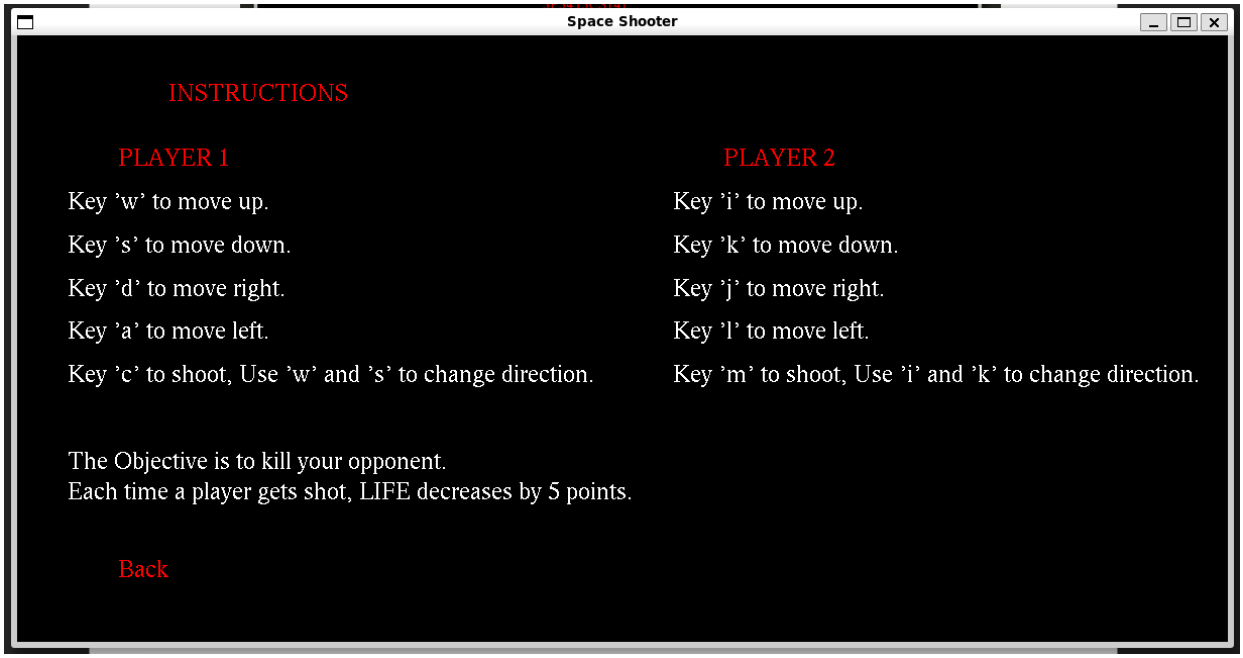
Preliminary Wireframe



Game Start Screen

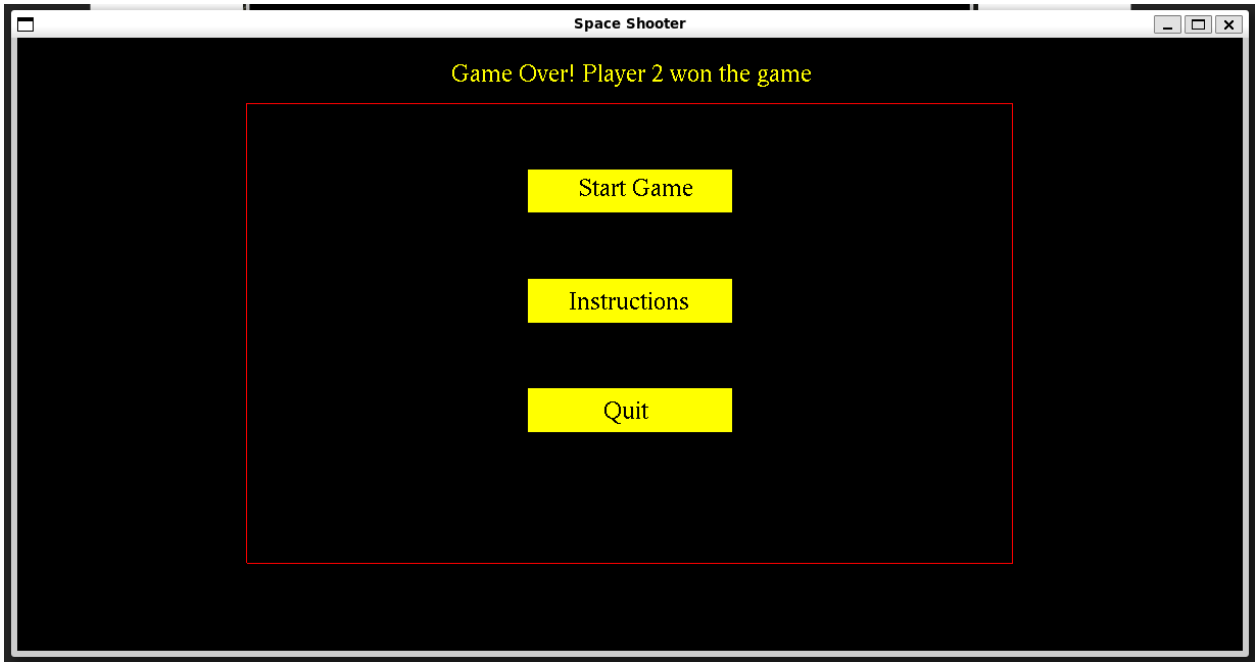


Instructions Wireframe



Gameplay

Game-over screen



References

OpenGL Programming Guide (Red Book)
https://www.khronos.org/registry/OpenGL-Refpages/gl4/
GLUT API Documentation
https://www.opengl.org/resources/libraries/glut/spec3/
The OpenGL Utility Toolkit (GLUT) Programming Interface (by Mark Kilgard)
https://www.opengl.org/documentation/specs/glut/
OpenGL Reference Manual (Blue Book)
https://www.khronos.org/registry/OpenGL-Refpages/
Computer Graphics with OpenGL (by Donald Hearn and M. Pauline Baker)
GLUT for Windows (FreeGLUT or original GLUT installers)
https://freeglut.sourceforge.net/