Computer Graphics (UCS505)

Project Name:
Enemy Rush

Branch

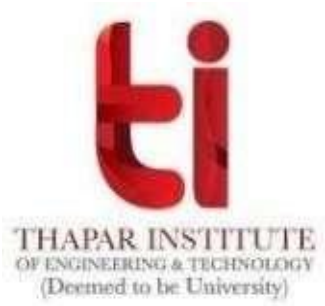
B.E. 3rd Year – COE

Submitted By

Bhavya Goyal (102203638)
Shreyansh Srivastava (102203642)
Shaambhavi Sharma (102383058)

Submitted To –
Raj Kumar Tekchandani

Computer Science and Engineering Department



Thapar Institute of Engineering and Technology

Patiala – 147001

Introduction

In the dynamic landscape of modern entertainment, the realm of computer graphics stands as a beacon of creativity and innovation. From breathtaking movie visuals to immersive video games, computer graphics have permeated every facet of our digital lives, reshaping how we interact with virtual worlds.

The cornerstone of computer graphics lies in its ability to simulate reality and unleash boundless creativity. From photorealistic renderings that mimic the complexities of the physical world to stylized animations that defy conventional norms, computer graphics offer a canvas where imagination knows no bounds. Whether it's the meticulous rendering of light and shadow or the seamless animation of characters, every aspect of computer graphics contributes to crafting compelling visual narratives.

This project delves into the fascinating realm of computer graphics through the lens of a captivating shooting game, offering an exhilarating journey into the fusion of technology and imagination.

OpenGL serves two main purposes :
● To hide the complexities of interfacing with different 3D accelerators, by presenting programmer with a single, uniform API
● To hide the differing capabilities of hardware platforms , by requiring that all implementations support the full OpenGL collector set.

OpenGL has historically been influential on the development of 3D accelerators, promoting a base level of functionality that is now common in consumer level hardware:
● Rasterized points, lines and polygons are basic primitives.
● A transform and lighting pipeline.
● Z buffering

Project Overview:

The project is a simple shooting game implemented using OpenGL and GLUT libraries in C++. The objective of the game is for the player to control a shooter and destroy enemy objects while avoiding collisions. It features various game elements such as a main menu, and an about screen. The game includes scoring mechanisms, multiple levels, and a life system where players lose a life if an enemy object passes the shooter. The project provides an entertaining and engaging gaming experience for users and takes you back to the retro style of gaming.

Basic Working:

- Initialize the OpenGL context and window.
- Set up the initial game state variables.
- Depending on the game state:
    - If in the main menu state (mainmenu == 1), render the main menu.
    - If in the how to play state (howtoplay == 1), render the instructions screen.
    - If in the game over state (game_over == 1), render the game over screen.
    - If in the gameplay state (game_play == 1):
        - Render the game scene
        - Draw the player's character (shooter).
        - Draw bullets if they are fired (bullet_stat[i] == 1).
        - Draw any other game elements (e.g., enemies, obstacles).
- Handle mouse clicks:
    - If in the main menu state:
    - Check the clicked area and transition to the corresponding state (gameplay, how to play, about, or exit).
    - If in other states:
        - Handle clicks based on the current state (e.g., return to the main menu from game over screen).
- Handle keyboard input:
    - If the 'ESC' key is pressed, exit the game.
    - Handle other key presses based on the current state (not explicitly defined in the provided code).

Scope of the Project:

The project aims to design and develop a 2D first-person shooter (FPS) game using OpenGL and C++. The primary objective is to create an interactive virtual environment where players eliminate enemy targets, and achieve high scores. The game demonstrates key principles of computer graphics including 3D rendering, camera manipulation, lighting, texture mapping, and collision detection.

This project covers the development of:

- A dynamic 3D environment with realistic terrain, skybox, and obstacles.
- Player mechanics including movement, aiming, and shooting.
- Enemy logic with autonomous movement and interaction.
- Score tracking and user feedback mechanisms.
- Real-time rendering techniques using OpenGL for visual effects and performance.

The scope also extends to understanding the application of computer graphics concepts in practical scenarios, offering a foundation for more advanced simulations and game development. While the current version is a single-player game with basic functionality, the structure allows for future extensions such as multiplayer features, AI enhancements, and improved user interface components.

Computer Graphics Concepts Used

1. OpenGL (Open Graphics Library): OpenGL is utilized for rendering 2D and 3D graphics in the game. It provides functions for drawing geometric shapes, applying textures, and implementing lighting effects. Additionally, OpenGL facilitates the creation of smooth animations and transitions between different game states, enhancing the overall visual appeal of the game.

2. GLUT (OpenGL Utility Toolkit): GLUT simplifies the setup of OpenGL projects by handling window creation, event handling, and input management. It allows for the creation of interactive graphical applications with minimal code. GLUT also provides functionality for creating menus, which is utilized in the project to implement the main menu and other user interface elements.

3. Coordinate Systems: The game utilizes coordinate systems to position and move game objects within the virtual environment. It involves transformations such as translation, rotation, and scaling to manipulate objects. The use of coordinate systems enables precise positioning of game elements on the screen and facilitates smooth movement animations.

4. Rendering Techniques: Various rendering techniques are employed to display game elements, including polygons, lines, and points. The use of solid spheres, lines, and polygons creates visually distinctive game objects such as the shooter, enemies, and bullets. Additionally, techniques such as texture mapping and shading are utilized to add detail and realism to the game environment.

5. Text Rendering: Text rendering is implemented to display information such as scores, life count, and menu options. Bitmap fonts are utilized to render text on the screen, enhancing the user interface and providing essential feedback to the player. The positioning and styling of text elements are carefully designed to ensure readability and aesthetic appeal.

6. Event Handling: The project utilizes event handling mechanisms to respond to user input, including keyboard and mouse interactions. Input events trigger actions such as moving the shooter, shooting bullets, and navigating through menu screens. Event handling ensures smooth and responsive gameplay, allowing players to interact with the game intuitively.

User Defined Functions

1. renderBitmapString(): This function is responsible for rendering text on the screen using OpenGL's bitmap fonts. It takes input parameters for position, font style, and the string to be rendered. Inside the function, it iterates over each character in the string and renders it using the specified font and position.

2. MainMenu(): Renders the main menu screen with options for starting a new game, viewing high scores, learning how to play, and accessing information about the game. It creates graphical elements such as buttons and text to present these options in an intuitive and visually appealing manner.

3. drawBullet(): Draws a bullet object on the screen. This function is responsible for rendering the bullet sprite or shape at its current position on the screen. It may also handle animations or effects associated with the bullet, such as trails or impact animations.

4. drawenemy(): Draws enemy objects on the screen. This function renders the enemy sprites or shapes at their current positions. It may also handle animations or visual effects associated with the enemies, such as movement animations or attack animations.

5. gameOver(): Renders the game over screen displaying the final score. This function creates the visual elements of the game over screen, including text displaying the final score achieved by the player. It may also provide options for restarting the game or returning to the main menu.

6. drawShooter(): Draws the shooter object on the screen. This function is responsible for rendering the player-controlled shooter sprite or shape at its current position. It may also handle animations or visual effects associated with the shooter, such as movement animations or firing animations.

7. new_update(): Updates the game state including movement of objects, collision detection, and scoring. This function is the core of the game's logic, responsible for updating the positions and states of all game objects based on user input and game rules. It handles tasks such as moving the shooter and enemies, detecting collisions between objects, and updating the player's score.

8. score(): Displays the current score on the screen. This function creates graphical elements to render the player's current score at a designated location on the screen. It updates the displayed score whenever the player earns points during gameplay.

9. showLife(): Displays the remaining life of the player on the screen. This function creates visual elements to represent the player's remaining life, such as hearts or a life bar, and updates the display as the player takes damage or gains extra lives during gameplay.

10. howToPlay(): Renders the screen with instructions on how to play the game. This function creates graphical elements to present the game's controls, objectives, and rules in a clear and concise manner. It may include text, images, or animations to help players understand the game mechanics.

11. handleResize(): Handles window resizing events. This function adjusts the game's viewport and projection matrix to ensure that the game remains visually consistent and playable when the window size changes. It may also update the positions and sizes of UI elements to fit the new window dimensions.

12. handleKeypress1(): Handles keyboard inputs such as spacebar for shooting and ESC key for exiting the game. This function detects and processes keyboard events, translating key presses into game actions such as shooting bullets or pausing the game. It may also handle special key combinations or sequences.

13. getCoord(): Retrieves mouse coordinates for handling mouse click events. This function captures the current position of the mouse cursor relative to the game window, allowing the game to respond to mouse interactions such as clicking on menu options or targeting enemies.

14. handleKeypress2(): Handles special keyboard inputs such as arrow keys for moving the shooter. This function interprets arrow key presses to move the player-controlled shooter object horizontally across the screen.

15. handleMouseclick(): Handles mouse click events for interacting with menu options. This function detects when the player clicks on a menu option or button and triggers the corresponding action, such as starting a new game or accessing a submenu. It may also handle mouse hover effects or highlight selected options.

16. drawScene(): Renders the game scene based on the current game state and user interactions. This function combines all visual elements of the game, including background scenery, game objects, UI elements, and special effects, to create the complete game experience. It is called repeatedly during gameplay to update the display in real-time.

17. main(): Initializes the OpenGL environment, sets up the window, registers

callback functions, and starts the main event loop. This function serves as the entry point for the game, initializing all necessary components and resources before entering the main game loop. It sets up the OpenGL context, creates the game window, and configures event handlers to respond to user input and system events

Code Snippets

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <algorithm>
#include <fstream>
using namespace std;
#define GL_SILENCE_DEPRECATION
#define SZ 100000
#define FOR(I,J,K) for (I = J; I < K; I++)
double enemy = 0.1f;
double enemy_x[SZ];
double enemy_y[SZ];
double enemy_vel[SZ];
double enemy_stat[SZ] = { 0 };
double enemy_col[SZ];
double shooter_x = 0.0f;
double shooter_y = -1.9f;
double shooter_min_x = -1.9f;
double shooter_max_x = 1.9f;
double angle = 0.0f;
double bullet_x[SZ];
double bullet_y[SZ];
double bullet_vel = 0.05f;
int bullet_stat[SZ] = { 0 };
double bullet_vel_x[SZ];
double bullet_vel_y[SZ];
double tip_x[SZ];
double tip_y[SZ];
int bullet = 0;
int points = 0;
int on = 1000;
int frontCount = -1, cnt = 0, high = -1;
int game_over = 0;
int game_play = 0;
int level = 1;
int game_level = 0;
double mousex;
double mousey;
int mainmenu = 1;
int high_score = 0;
int howtoplay = 0;
int about = 0;
int life = 3;
int flag = 0;
int a = 0;
void renderBitmapString(float x, float y, void* font, const char* string)
{
      const char* c;
      glRasterPos2f(x, y);
```

```
	for (c = string; *c != '\0'; c++) {
		glutBitmapCharacter(font, *c);
	}
}
void MainMenu() {
	glLineWidth(4);
	glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
	glColor3f(1.0, 1.0, 1.0);
	renderBitmapString(-0.5f, 1.3f, GLUT_BITMAP_HELVETICA_18, "ENEMY
RUSH");
	renderBitmapString(-0.2f, 0.8f, GLUT_BITMAP_HELVETICA_18, "By -");
	renderBitmapString(-0.6f, 0.6f, GLUT_BITMAP_HELVETICA_18, "Bhavya
(102203638)");
	renderBitmapString(-0.6f, 0.4f, GLUT_BITMAP_HELVETICA_18, "Shambhavi
(102383058)");
	renderBitmapString(-0.6f, 0.2f, GLUT_BITMAP_HELVETICA_18, "Shreyansh
(102203642)");
	glBegin(GL_LINES);
	glVertex2f(-0.4f, -0.2f);
	glVertex2f(0.4f, -0.2f);
	glVertex2f(-0.4f, -0.5f);
	glVertex2f(0.4f, -0.5f);
	glVertex2f(-0.4f, -0.2f);
	glVertex2f(-0.4f, -0.5f);
	glVertex2f(0.4f, -0.2f);
	glVertex2f(0.4f, -0.5f);
	glEnd();
	renderBitmapString(-0.33f, -0.38f, GLUT_BITMAP_HELVETICA_18, "NEW
GAME");
	glBegin(GL_LINES);
	glVertex2f(-0.4f, -0.7f);
	glVertex2f(0.4f, -0.7f);
	glVertex2f(-0.4f, -1.0f);
	glVertex2f(0.4f, -1.0f);
	glVertex2f(-0.4f, -0.7f);
	glVertex2f(-0.4f, -1.0f);
	glVertex2f(0.4f, -0.7f);
	glVertex2f(0.4f, -1.0f);
	glEnd();
	renderBitmapString(-0.42f, -0.88f, GLUT_BITMAP_HELVETICA_18, "HOW TO
PLAY");


}
void drawBullet() {
	glPushMatrix();
	glColor3f(1.0, 1.0, 1.0);
	glTranslatef(-170 / 1200, 220 / 800, 0);
	glutSolidSphere(0.03, 50, 50);
	glPopMatrix();
}
void drawenemy() {
	int i;
	if (frontCount == -1) {
```

```
			cnt = 0;
		}
		if (frontCount == 0) {
			cnt = 1;
		}
		if (frontCount < 100000) {
			for (i = 0;cnt <= 2;i++) {
				if (i > frontCount || enemy_y[i] <= -5.0f) {
					enemy_x[i] = ((rand() % 100) * 1.0f / 100.0) +
((rand() % 4) * 1.0f) - 2;
					enemy_y[i] = 2.3f;
					enemy_vel[i] = .003f + 3 * ((rand() % 10 * 1.0f) /
10000);
					enemy_stat[i] = 0;
					enemy_col[i] = rand() % 3;
					frontCount++;
					cnt++;
				}
			}
		}
	}
	void drawenemy(int i) {
		glColor3f(0.0, 1.0, 0.0);
		glPushMatrix();
		glTranslatef(0.0, 0.07, 0.0);
		glutSolidSphere(0.04, 50, 50);
		glPopMatrix();
		glLineWidth(5);
		glColor3f(1.0, 0.0, 0.0);
		glBegin(GL_LINES);
		glVertex2f(-0.02, -0.05);
		glVertex2f(-0.09, -0.14);
		glVertex2f(0.02, -0.05);
		glVertex2f(0.09, -0.14);
		glEnd();
		glColor3f(1.0, 0.0, 0.0);
		glLineWidth(5);
		glBegin(GL_LINES);
		glVertex2f(-0.04, 0.01);
		glVertex2f(-0.1, -0.04);
		glVertex2f(0.04, 0.01);
		glVertex2f(0.1, -0.04);
		glEnd();
		glColor3f(1.0, 1.0, 0.0);
		glPointSize(15.0);
		glBegin(GL_POINTS);
		glVertex2f(-0.000, -0.01);
		glEnd();
	}
	void gameOver() {
		glColor3f(1.0, 1.0, 1.0);
		renderBitmapString(-0.5f, 0.7f, GLUT_BITMAP_HELVETICA_18, "GAME
	OVER");
		renderBitmapString(-0.5f, 0.5f, GLUT_BITMAP_HELVETICA_18, "SCORE :
```

```
");
	char str[80];
	sprintf_s(str, "%d", points);
	for (int i = 0;i < (int)(strlen(str));i++)
		glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, str[i]);
	glBegin(GL_LINES);
	glVertex2f(-0.6f, -0.5f);
	glVertex2f(-0.6f, -0.2f);
	glVertex2f(-0.6f, -0.5f);
	glVertex2f(0.6f, -0.5f);
	glVertex2f(-0.6f, -0.2f);
	glVertex2f(0.6f, -0.2f);
	glVertex2f(0.6f, -0.5f);
	glVertex2f(0.6f, -0.2f);
	glEnd();
	renderBitmapString(-0.2f, -0.4f, GLUT_BITMAP_HELVETICA_18, "MENU");
	frontCount = -1;
	cnt = 0;
	high = -1;
}
void drawShooter() {
	glColor3f(0.0, 0.5, 1.0);
	glBegin(GL_POLYGON);
	glVertex2f(-0.2f, -0.2f);
	glVertex2f(0.2f, -0.2f);
	glVertex2f(0.2f, -0.15f);
	glVertex2f(-0.2f, -0.15f);
	glEnd();
	glBegin(GL_POLYGON);
	glVertex2f(-0.1f, -0.5f);
	glVertex2f(0.1f, -0.5f);
	glVertex2f(0.1f, 0.16f);
	glVertex2f(-0.1f, 0.16f);
	glEnd();
	glBegin(GL_POLYGON);
	glVertex2f(0.1f, -0.15f);
	glVertex2f(0.4f, -0.15f);
	glVertex2f(0.4f, -0.05f);
	glVertex2f(0.1f, -0.05f);
	glEnd();
	glBegin(GL_POLYGON);
	glVertex2f(-0.1f, -0.15f);
	glVertex2f(-0.4f, -0.15f);
	glVertex2f(-0.4f, -0.05f);
	glVertex2f(-0.1f, -0.05f);
	glEnd();
	glBegin(GL_POLYGON);
	glVertex2f(-0.3f, -0.07f);
	glVertex2f(-0.4f, -0.07f);
	glVertex2f(-0.4f, 0.15f);
	glVertex2f(-0.3f, 0.1f);
	glEnd();
	glBegin(GL_POLYGON);
	glVertex2f(0.3f, -0.07f);
```

```
	glVertex2f(0.4f, -0.07f);
	glVertex2f(0.4f, 0.15f);
	glVertex2f(0.3f, 0.1f);
	glEnd();
	glBegin(GL_POLYGON);
	glVertex2f(-0.1f, 0.16f);
	glVertex2f(0.1f, 0.16f);
	glVertex2f(0.005f, 0.22f);
	glEnd();
}
void new_update(int value) {
	for (int i = 0;i <= frontCount;i++) {
		if (enemy_y[i] <= -1.6f) {
			enemy_vel[i] = 0;
			enemy_y[i] = -5.0f;
			enemy_stat[i] = 0;
			frontCount--;
			points--;
		}
		if (shooter_x > enemy_x[i] - 0.46f && shooter_x < enemy_x[i] +
0.46f &&
			enemy_y[i] - 0.05f <= -1.7f) {
			life--;
		}
		if (life == 0) {
			game_over = 1;
			game_play = 0;
			mainmenu = 0;
		}
	}
	for (int i = 0;i <= high;i++) {
		if (bullet_stat[i] == 1) {
			bullet_x[i] -= bullet_vel_x[i];
			bullet_y[i] += bullet_vel_y[i];
			tip_x[i] -= bullet_vel_x[i];
			tip_y[i] += bullet_vel_y[i];
			if (bullet_y[i] >= 2.0f) {
				bullet_stat[i] = 0;
			}
		}
	}
	for (int i = 0;i <= frontCount;i++) {
		for (int j = 0;j <= high;j++) {
			if (tip_x[j] > enemy_x[i] - 0.05f && tip_x[j]<enemy_x[i]
+ 0.05f &&
				tip_y[j]>enemy_y[i] - 0.05f && tip_y[j] <
enemy_y[i] + 0.05f && bullet_stat[j] == 1) {
				enemy_vel[i] = 0;
				enemy_y[i] = -5.0f;
				enemy_stat[i] = 0;
				frontCount--;
				bullet_stat[j] = 0;
				points++;
			}
```

```
			}
		}
		for (int i = 0;i <= frontCount;i++) {
			if (enemy_y[i] < 0.0f && enemy_stat[i] == 0) {
				cnt--;
				enemy_stat[i] = 1;
			}
			if (enemy_y[i] >= -1.85f) {
				enemy_y[i] -= enemy_vel[i];
			}
		}
		on += 100;
		game_level += 10;
		if (game_level >= 10000) {
			level++;
			game_level = 1;
		}
		glutTimerFunc(10, new_update, 0);
	}
	void score() {
		glColor3f(1.0f, 1.0f, 1.0f);
		renderBitmapString(2.2f, 1.8f, GLUT_BITMAP_HELVETICA_18, "SCORE :
	");
		char str[80];
		sprintf_s(str, "%d", points);
		for (int i = 0;i < (int)(strlen(str));i++)
			glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, str[i]);
	}
	void showLife() {
		glColor3f(1.0f, 1.0f, 1.0f);
		renderBitmapString(-2.6f, 1.8f, GLUT_BITMAP_HELVETICA_18, "LIFE :
	");
		char str[80];
		sprintf_s(str, "%d", life);
		for (int i = 0;i < (int)(strlen(str));i++)
			glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, str[i]);
	}

	void howToPlay() {
		glColor3f(1.0f, 1.0f, 1.0f);

		// Title
		renderBitmapString(-0.3f, 1.5f, GLUT_BITMAP_HELVETICA_18, "HOW TO
	PLAY");

		// Instructions (centered horizontally, spaced vertically)
		renderBitmapString(-1.8f, 1.2f, GLUT_BITMAP_HELVETICA_18, "1. Use
	Left arrow key to move the shooter Left");
		renderBitmapString(-1.8f, 1.0f, GLUT_BITMAP_HELVETICA_18, "2. Use
	Right arrow key to move the shooter Right");
		renderBitmapString(-1.8f, 0.8f, GLUT_BITMAP_HELVETICA_18, "3. Press
	Space key to Shoot");
		renderBitmapString(-1.8f, 0.6f, GLUT_BITMAP_HELVETICA_18, "4. Kill
	the object to earn 1 point");
```

```
	renderBitmapString(-1.8f, 0.4f, GLUT_BITMAP_HELVETICA_18, "5. If
objects are missed and cross the shooter, 1 life is lost");
	renderBitmapString(-1.8f, 0.2f, GLUT_BITMAP_HELVETICA_18, "6. Press
ESC to exit the game");

	// BACK button centered at the bottom
	glBegin(GL_LINES);
	glVertex2f(-0.4f, -1.2f);
	glVertex2f(0.4f, -1.2f);
	glVertex2f(-0.4f, -1.5f);
	glVertex2f(0.4f, -1.5f);
	glVertex2f(-0.4f, -1.2f);
	glVertex2f(-0.4f, -1.5f);
	glVertex2f(0.4f, -1.2f);
	glVertex2f(0.4f, -1.5f);
	glEnd();
	renderBitmapString(-0.1f, -1.38f, GLUT_BITMAP_HELVETICA_18, "BACK");
}

void handleResize(int w, int h) {
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(45.0f, (double)w / (double)h, 0.1f, 200.0f);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}
void handleKeypress1(unsigned char key, int x, int y) {
	if (key == 27) {
		exit(0);
	}
	else if (key == ' ' && on >= 1000) {
		int i;
		for (i = 0;i < high;i++) {
			if (bullet_stat[i] == 0) {
				bullet_x[i] = shooter_x;
				bullet_y[i] = shooter_y;
				tip_x[i] = bullet_x[i];
				tip_y[i] = bullet_y[i] + 0.2;
				bullet_vel_x[i] = 0;
				bullet_vel_y[i] = bullet_vel;
				bullet_stat[i] = 1;
				bullet = 1;
				break;
			}
		}
		if (bullet == 0) {
			high++;
			bullet_x[high] = shooter_x;
			bullet_y[high] = shooter_y;
			tip_x[high] = shooter_x;
			tip_y[high] = shooter_y + 0.2;
			bullet_vel_x[high] = 0;
			bullet_vel_y[high] = bullet_vel;
```

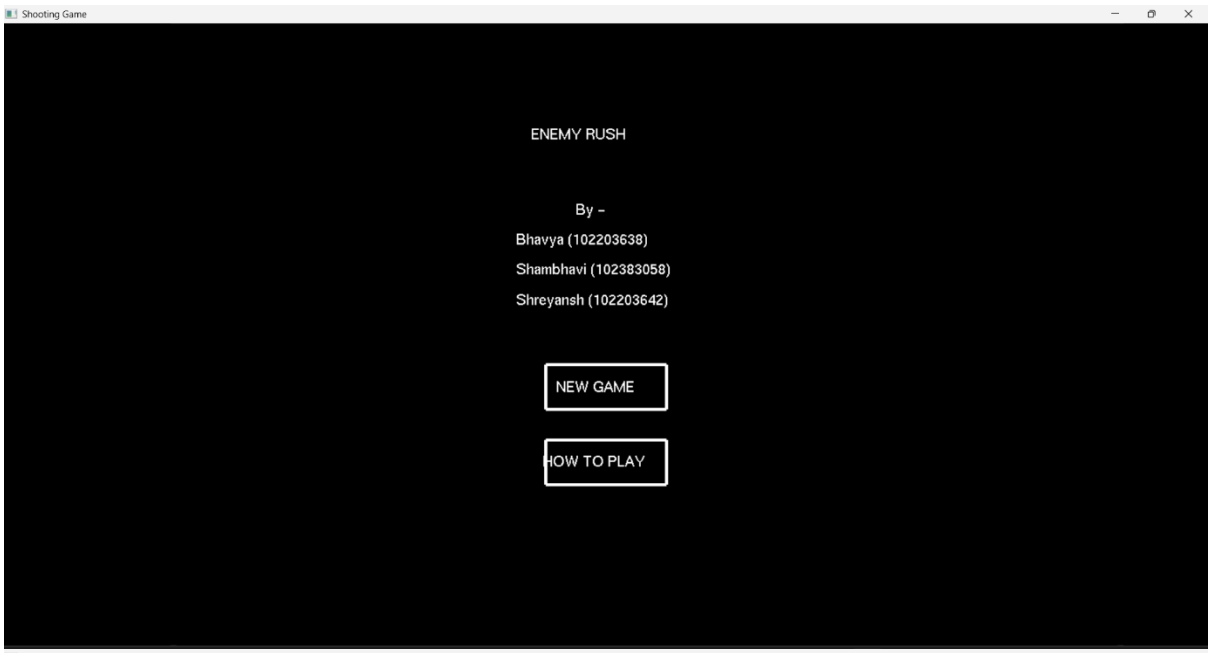
```
                  bullet_stat[high] = 1;
            }
            on = 0;
            bullet = 0;
      }
}
void getCoord(int x, int y) {
      GLint viewport[4];
      GLdouble modelview[16];
      GLdouble projection[16];
      GLfloat winX, winY, winZ;
      GLdouble posX, posY, posZ;
      glGetIntegerv(GL_VIEWPORT, viewport);
      glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
      glGetDoublev(GL_PROJECTION_MATRIX, projection);
      winX = (float)x;
      winY = (float)viewport[3] - (float)y;
      glReadPixels(x, int(winY), 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT,
&winZ);
      winZ = 0;
      gluUnProject(winX, winY, winZ, modelview, projection, viewport,
&posX, &posY,
            &posZ);
      mousex = posX * 100 / 2;
      mousey = (posY * 100) / 2;
      glutPostRedisplay();
}
void handleKeypress2(int key, int x, int y) {
      if (key == GLUT_KEY_LEFT) {
            if (shooter_x > shooter_min_x) {
                  shooter_x -= 0.05;
            }
      }
      else if (key == GLUT_KEY_RIGHT) {
            if (shooter_x < shooter_max_x) {
                  shooter_x += 0.05;
            }
      }
}
void handleMouseclick(int button, int state, int x, int y) {
      if (button == GLUT_LEFT_BUTTON && game_play == 0 && mainmenu == 1) {
            getCoord(x, y);

            // NEW GAME button area
            if (mousex >= -0.4f && mousex <= 0.4f && mousey >= -0.5f &&
mousey <= -0.2f) {
                  // Only start the game if it's a new game, prevent
immediate start when returning to main menu
                  if (game_over == 0) {
                        game_play = 1;
                        game_over = 0;
                  }
            }
            // HOW TO PLAY button area
```

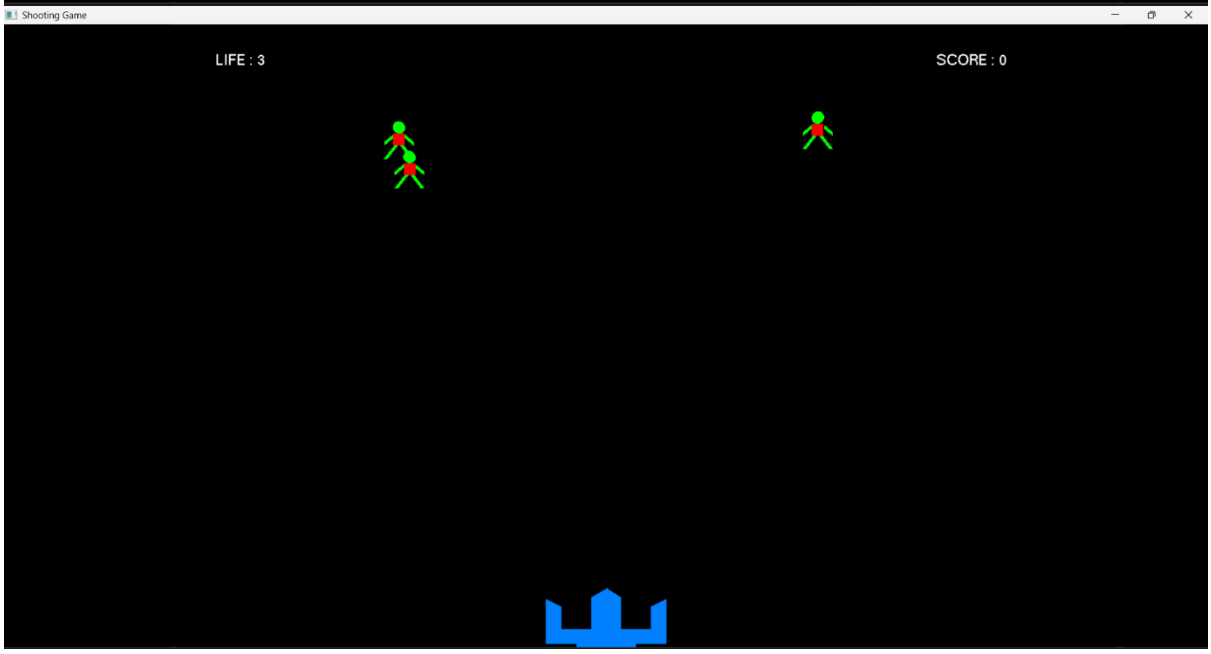
```
            else if (mousex >= -0.4f && mousex <= 0.4f && mousey >= -1.0f
&& mousey <= -0.7f) {
                  mainmenu = 0;
                  howtoplay = 1;
            }
      }

      // BACK button from HOW TO PLAY
      else if (button == GLUT_LEFT_BUTTON && game_play == 0 && mainmenu ==
0 && howtoplay == 1) {
            getCoord(x, y);
            if (mousex >= -0.4f && mousex <= 0.4f && mousey >= -1.5f &&
mousey <= -1.2f) {
                  mainmenu = 1;
                  howtoplay = 0;
            }
      }

      // BACK button from GAME OVER screen
      else if (button == GLUT_LEFT_BUTTON && game_play == 0 && mainmenu ==
0 && game_over == 1) {
            getCoord(x, y);
            if (mousex >= -0.4f && mousex <= 0.4f && mousey >= -0.5f &&
mousey <= -0.2f) {
                  mainmenu = 1;
                  game_over = 0;
                  life = 3;          // Reset the number of lives
                  points = 0;        // Reset points
            }
      }
}


void drawScene() {
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      if (game_play == 0 && mainmenu == 1) {
            glPushMatrix();
            glTranslatef(0.0f, 0.0f, -5.0f);
            glColor3f(0.0f, 0.0f, 0.0f);
            MainMenu();
            glPopMatrix();
      }
      else if (game_play == 0 && mainmenu == 0 && game_over == 1) {
            glPushMatrix();
            glTranslatef(0.0f, 0.0f, -5.0f);
            glColor3f(0.0f, 0.0f, 0.0f);
            gameOver();
            glPopMatrix();
      }
      else if (mainmenu == 0 && howtoplay == 1 && game_play == 0) {
            glPushMatrix();
            glTranslatef(0.0f, 0.0f, -5.0f);
```

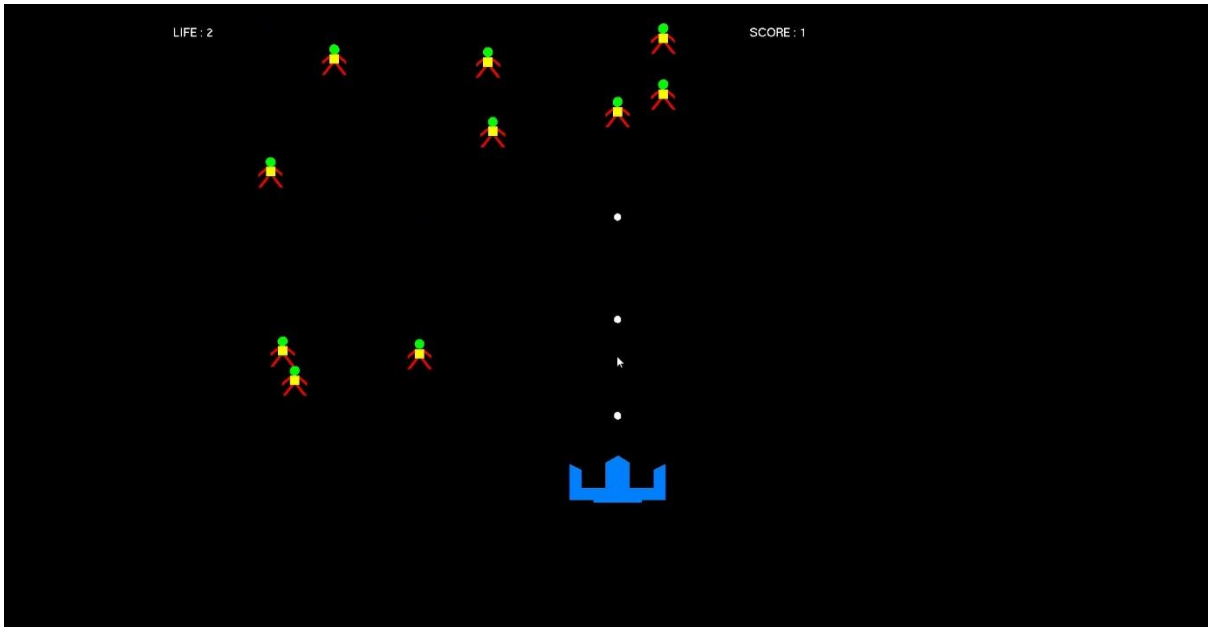
```
		glColor3f(0.0f, 0.0f, 0.0f);
		howToPlay();
		glPopMatrix();
	}
	else if (mainmenu == 0 && game_play == 0 && game_over == 1) {
		glPushMatrix();
		glTranslatef(0.0f, 0.0f, -5.0f);
		glColor3f(0.0f, 0.0f, 0.0f);
		gameOver();
		glPopMatrix();
	}
	else if (game_over == 0 && game_play == 1) {
		drawenemy();
		glPushMatrix();
		glTranslatef(0.0f, 0.0f, -5.0f);
		glColor3f(0.0f, 0.0f, 0.0f);
		for (int i = 0;i <= high;i++) {
			if (bullet_stat[i] == 1) {
				glPushMatrix();
				glTranslatef(bullet_x[i], bullet_y[i], 0.0f);
				glRotatef(0.0f, 0.0f, 0.0f, 1.0f);
				glColor3f(0.0f, 0.0f, 0.0f);
				drawBullet();
				glPopMatrix();
			}
		}
		glPushMatrix();
		glTranslatef(shooter_x, shooter_y, 0.0f);
		drawShooter();
		glPopMatrix();
		for (int i = 0;i <= frontCount;i++) {
			glPushMatrix();
			glTranslatef(enemy_x[i], enemy_y[i], 0.0f);
			if (enemy_vel[i] != 0)
				drawenemy(i);
			glPopMatrix();
		}
		glPushMatrix();
		glColor3f(1.0f, 0.0f, 0.0f);
		score();
		showLife();
		glPopMatrix();
	}
	else {
		glPushMatrix();
		glTranslatef(0.0f, 0.0f, -5.0f);
		glColor3f(0.0f, 0.0f, 0.0f);
		glPushMatrix();
		glPushMatrix();
		glColor3f(1.0f, 0.0f, 0.0f);
		glPopMatrix();
	}
	glPopMatrix();
	glutSwapBuffers();
```

```
}
int main(int argc, char** argv) {
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowSize(1200, 800);
	glutInitWindowPosition(220, 50);
	glutCreateWindow("Shooting Game");
	glutDisplayFunc(drawScene);
	glutIdleFunc(drawScene);
	glutKeyboardFunc(handleKeypress1);
	glutSpecialFunc(handleKeypress2);
	glutMouseFunc(handleMouseclick);
	glutReshapeFunc(handleResize);
	glutTimerFunc(5, new_update, 0);
	glutMainLoop();
	return 0;
}
```

Screenshots






LIFE : 2
SCORE : 1


GAME OVER
SCORE : 6
MENU


Shooting Game
HOW TO PLAY
1. Use Left arrow key to move the shooter Left
2. Use Right arrow key to move the shooter Right
3. Press Space key to Shoot
4. Kill the object to earn 1 point
5. If objects are missed and cross the shooter, 1 life is lost
6. Press ESC to exit the game
BACK