Computer Graphics (UCS505)

Project on

OpenGL: Guardian of Galaxy

Submitted By

Ishita Suchdeva	102103493
Selina Varshney	102103496

3CO18

B.E. Third Year – COE

Submitted To:

Jasmine Kaur

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

1. Introduction to Project

OpenGL Game: Guardian of Galaxy

- In "Guardian of Galaxy," players take on the role of an alien tasked with defending their home planet from a relentless barrage of asteroids hurtling through space. Armed with a powerful laser beam, players must skillfully blast these cosmic hazards.
- Initially equipped with a life force of 100, players blast their way through waves of asteroids and earn 1 point for each successful hit for surviving the onslaught. With each passing level, the asteroids come faster and in greater numbers, testing the player's reflexes and precision. However, danger lurks in every collision, as each impact on the player's spaceship results in a decrement of 10 points from their life force, which when reaches 0, the game ends.
- "Guardian of Galaxy" challenges players with its fast-paced action, strategic decision-making, and relentless pace. With every hit and miss, the stakes escalate, pushing players to the limits of their abilities. Will you emerge victorious and claim the title of the ultimate guardian of the galaxy? The fate of your alien homeland hangs in the balance.

2. Computer Graphics concepts used

OpenGL Setup and Initialization:

glutInit() initializes GLUT, glutInitDisplayMode() sets display parameters, glutInitWindowSize() and glutInitWindowPosition() configure window size and position. glutCreateWindow() creates the window, glClearColor() sets the clear color. glMatrixMode() and glLoadIdentity() handle transformation matrices, gluOrtho2D() defines the 2D viewing region. Callback functions like glutDisplayFunc(), glutKeyboardFunc(), etc., manage user interactions. glutMainLoop() maintains continuous event processing

OpenGL Primitives: The code utilizes different OpenGL primitives like GL_POLYGON, GL_LINE_LOOP, GL_LINES, and GL_POINTS to define and draw shapes such as polygons, lines, and points.

Primitive Drawing and 2D Transormations:

These functions are crucial for defining and drawing geometric primitives in OpenGL. glBegin() and glEnd() are used to delineate the vertices of shapes, while glVertex2f() specifies 2D vertex coordinates. Setting colors is done with glColor3f() and glColor4f(), and line width with glLineWidth(). glRasterPos3f() determines the position for 3D drawing. Transformation functions like glTranslated(), glRotated(), and glScalef() manipulate objects on the current matrix stack, with glPushMatrix() and glPopMatrix() used to save and restore matrix states. Lastly, glutBitmapCharacter() renders bitmap characters using specified fonts.

Event Handling:

- Mouse button and motion callback functions (glutMouseFunc(), glutMotionFunc(), glutPassiveMotionFunc()) handle mouse events.
- Keyboard callback function (glutKeyboardFunc()) handles keyboard events.

File Handling:

- fopen_s() opens a file stream securely.
- fread() reads data from a file stream.
- fwrite() writes data to a file stream.
- fclose() closes a file stream.

3. User Defined Functions

- **displayRasterText(float x, float y, float z, char* stringToDisplay):**

  This function is used to display raster text on the OpenGL window at the specified coordinates (x, y, z). It takes a string (stringToDisplay) as input and displays it at the specified location.
- **SetDisplayMode(int modeToDisplay):**

  This function sets the display mode based on the value of modeToDisplay. It is used to switch between different display modes such as the game screen and menu screen. It changes the background color accordingly.
- **initializeStoneArray():**

  This function initializes the array of stones used in the game. It generates random indices for different types of stones and initializes their positions. It also sets the status of each stone to alive.
- **Drawing Functions**

  DrawAlienBody(), DrawAlienCollar(), DrawAlienFace(), DrawAlienBeak(), DrawAlienEyes(), DrawAlien(), DrawSpaceshipBody(), DrawSteeringWheel(), DrawSpaceshipDoom(), DrawSpaceShipLazer(), DrawLazerBeam() : All these are used for drawing graphics using drawing primitives and 2D transformations.

  DrawStone(int StoneIndex):

This function draws different types of stones based on the StoneIndex. It specifies the vertices and colors to draw the stones.

- **checkIfSpaceShipIsSafe():**

  This function checks if the spaceship is safe from collision with stones. It iterates through all the stones and checks if the spaceship collides with any of them.

- **SpaceshipCreate():**

  This function combines different spaceship drawing functions to draw a complete spaceship on the screen.

- **DisplayHealthBar():**

  This function displays the health bar, score, and level information on the screen using OpenGL raster text.

- **startScreenDisplay():**

  This function displays the start screen of the game. It provides options for starting the game, viewing instructions, or quitting the game.

- **GameScreenDisplay():**

  This function displays the game screen. It includes the spaceship, stones, health bar, score, and level information.

- **readFromFile():**

  This function reads the highest score from a file named "HighScoreFile.txt".

- **writeIntoFile():**

  This function writes the current score into a file named "HighScoreFile.txt" if it is higher than the previously stored high score.

- **GameOverScreen():**

  This function displays the game over screen. It shows the final score, highest score, and options to restart the game or quit.

- **StoneGenerate():**

  This function generates stones on the game screen. It updates the position of stones and checks for collisions with the spaceship. If a stone is clicked, it increments the score.

- **backButton():**

  If clicked, it resets the instructionsGame flag, enabling the player to return to the start screen.

- **InstructionsScreenDisplay():**

  This function displays the instructions screen of the game. It clears the color and depth buffers, sets the display mode to MENU_SCREEN, and then displays instructions for playing the game, such as how to move the spaceship and shoot lasers. It also includes a back button to return to the previous screen.

- **display():**

  This function is the main display function for rendering graphics. It first clears the color buffer, sets the viewport size, and then checks the game state to determine which screen to display: start screen, game screen, instructions screen, or game over screen. It calls the corresponding display functions based on the game state.

- **somethingMovedRecalculateLaserAngle():**

  This function recalculates the angle of the laser beam based on the movement of the mouse and updates the LaserAngle variable. It calculates the angle using trigonometric functions (atan) based on the difference in mouse position and spaceship position.

- **keys(unsigned char key, int x, int y):**

  This function handles keyboard input from the user. It checks for specific key presses ('w', 'a', 's', 'd') to move the spaceship in different directions. It also calls somethingMovedRecalculateLaserAngle() when any movement key is pressed to recalculate the angle of the laser beam. After processing the key presses, it calls the display() function to update the display accordingly.

- **myinit():**

  This function initializes OpenGL parameters, setting the clear and drawing colors and defining an orthographic projection. Adjusting gluOrtho2D() parameters can expand the viewport for extra content.

- **passiveMotionFunc(int x, int y):**

  This function is called when the mouse moves within the window without any mouse buttons being pressed. It calculates the position of the mouse in the window coordinates and then converts it to the orthographic 2D specifications used in the game. After calculating the mouse position, it calls somethingMovedRecalculateLaserAngle() to update the angle of the laser beam based on the mouse position and then updates the display using the display() function.

- **mouseClick(int buttonPressed, int state, int x, int y):**

  This function is called when a mouse button is clicked or released. It takes parameters indicating which button was pressed, the state of the button (pressed or released), and the mouse coordinates. In this case, it checks if the left mouse button was pressed and sets a flag (mButtonPressed) accordingly. It then updates the display using the display() function.

- **UpdateColorIndexForSpaceshipLights(int value):**

  This function updates the color index for the spaceship lights. It increments the color index (CI) and performs modulo 3 operation to ensure that the index loops back to 0 after reaching 2, creating a rotation effect of colors. After updating the color index, it updates the display using the display() function. Additionally, it sets a timer using glutTimerFunc() to call itself again after a specified interval (250 milliseconds in this case).

4. Code:

```
/***********************************************************************************
**                              G A M E    D E S C R I P T I O N S
**
**                                          keybord controls=   w,a,s,d
**                                          mouse left click to fire
**
**          INSTRUCTIONS:
**                                          Dodge the objects and shoot them down
**
**          OBJECTIVE :   Beat the high score.
**                                          Score +1 point for shooting each object
**                                          Score +50 for lvl up
************************************************************************************
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <gl/glut.h>
#include <math.h>
#include<string.h>

#define PI 3.14159
#define GAME_SCREEN 0                       //Constant to identify background color
#define MENU_SCREEN 4
#define MAX_STONES  100
#define MAX_STONE_TYPES 5
#define stoneRotationSpeed 20
#define SPACESHIP_SPEED 20
int stoneTranslationSpeed = 5;

GLint m_viewport[4];
GLint CI = 0;
int x, y;
int i;
int randomStoneIndices[100];
int index;
int Score = 0;
int alienLife = 100;
int GameLvl = 1;
float mouseX, mouseY;                       //Cursor coordinates;
float LaserAngle = 0, stoneAngle = 0, lineWidth = 1;
float xOne = 0, yOne = 0;              //Spaceship coordinates
float xStone[MAX_STONES], yStone[MAX_STONES];  //coordinates of stones
float xStart = 1200;                        //Health bar starting coodinate
GLint stoneAlive[MAX_STONES];               //check to see if stone is killed
```

```
bool mButtonPressed = false, startGame = false, gameOver = false; //boolean values to check state of the game
bool startScreen = true, nextScreen = false, previousScreen = false;
bool gameQuit = false, instructionsGame = false, optionsGame = false;

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} };
GLfloat ALienBeak[][2] = { {-6,21.5}, {-6.5,22}, {-9,21}, {-11,20.5}, {-20,20}, {-14,23}, {-9.5,28}, {-7,27},      {-6,26.5},{-
4.5,23}, {-4,21}, {-6,19.5}, {-8.5,19}, {-10,19.5}, {-11,20.5} };

char highScore[100], ch;
void display();
void StoneGenerate();
void displayRasterText(float x, float y, float z, char* stringToDisplay) {
        int length;
        glRasterPos3f(x, y, z);
        length = strlen(stringToDisplay);
        for (int i = 0; i < length; i++) {
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, stringToDisplay[i]);
        }
}
void SetDisplayMode(int modeToDisplay) {
        switch (modeToDisplay) {
        case GAME_SCREEN: glClearColor(0, 0, 0, 1); break;
        case MENU_SCREEN: glClearColor(1, 0, 0, 1); break;
        }
}
void initializeStoneArray() {
        //random stones index
        for (int i = 0; i < MAX_STONES; i++) {
                randomStoneIndices[i] = rand() % MAX_STONE_TYPES;
                stoneAlive[i] = true;
        }
        xStone[0] = -(200 * MAX_STONES) - 600;          //START LINE for stone appearance

        for (int i = 0; i < MAX_STONES; i++) {     //random appearance yIndex for each stone
                yStone[i] = rand() % 600;
                if (int(yStone[i]) % 2)
                        yStone[i] *= -1;
                xStone[i + 1] = xStone[i] + 200;                //xIndex of stone aligned with 200 units gap
        }
}
```

```
void DrawAlienBody()
{
        glColor3f(0, 1, 0);                                     //BODY color
        glBegin(GL_POLYGON);
        for (i = 0; i <= 8; i++)
                glVertex2fv(AlienBody[i]);
        glEnd();
        glColor3f(0, 0, 0);                             //BODY Outline
        glLineWidth(1);
        glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 8; i++)
                glVertex2fv(AlienBody[i]);
        glEnd();
        glBegin(GL_LINES);              //BODY effect
        glVertex2f(-13, 11);
        glVertex2f(-15, 9);
        glEnd();
}
void DrawAlienCollar()
{
        glColor3f(1, 0, 0);                                     //COLLAR
        glBegin(GL_POLYGON);
        for (i = 0; i <= 20; i++)
                glVertex2fv(AlienCollar[i]);
        glEnd();
        glColor3f(0, 0, 0);                                     //COLLAR outline
        glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 20; i++)
                glVertex2fv(AlienCollar[i]);
        glEnd();
}
void DrawAlienFace()
{
        //glColor3f(0.6,0.0,0.286);                             //FACE
        //glColor3f(0.8,0.2,0.1);
        //glColor3f(0,0.5,1);
        glColor3f(0, 0, 1);
        glBegin(GL_POLYGON);
        for (i = 0; i <= 42; i++)
                glVertex2fv(ALienFace[i]);
        glEnd();
        glColor3f(0, 0, 0);                                     //FACE outline
        glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 42; i++)
                glVertex2fv(ALienFace[i]);
        glEnd();
        glBegin(GL_LINE_STRIP);     //EAR effect
        glVertex2f(3.3, 22);
        glVertex2f(4.4, 23.5);
```

```
	glVertex2f(6.3, 26);
	glEnd();
}
void DrawAlienBeak()
{
	glColor3f(1, 1, 0);				//BEAK color
	glBegin(GL_POLYGON);
	for (i = 0; i <= 14; i++)
		glVertex2fv(ALienBeak[i]);
	glEnd();

	glColor3f(0, 0, 0);				//BEAK outline
	glBegin(GL_LINE_STRIP);
	for (i = 0; i <= 14; i++)
		glVertex2fv(ALienBeak[i]);
	glEnd();
}
void DrawAlienEyes()
{
	glColor3f(0, 1, 1);
	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 DrawAlien()
{
	DrawAlienBody();
	DrawAlienCollar();
	DrawAlienFace();
	DrawAlienBeak();
	DrawAlienEyes();
}
void DrawSpaceshipBody()
{
	glColor3f(1, 0, 0);				//BASE
	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 DrawSpaceshipDoom()
{
        glColor4f(0.7, 1, 1, 0.0011);
        glPushMatrix();
        glTranslated(0, 30, 0);
        glScalef(35, 50, 1);
        glutSolidSphere(1, 50, 50);
        glPopMatrix();
}
void DrawSpaceShipLazer() {

        glColor3f(1, 0, 0);
        glPushMatrix();
        glBegin(GL_POLYGON);        //Lazer stem
        glVertex2f(-55, 10);
        glVertex2f(-55, 30);
        glVertex2f(-50, 30);
        glVertex2f(-50, 10);
        glEnd();

        float xMid = 0, yMid = 0;
        //Mid point of the lazer horizontal
        xMid = (55 + 50) / 2.0;
        yMid = (25 + 35) / 2.0;

        //Rotating about the point ,20
        glTranslated(-xMid, yMid, 0);
        glRotated(LaserAngle, 0, 0, 1);
        glTranslated(xMid, -yMid, 0);

        //find mid point of top of lazer stem
        float midPoint = -(55 + 50) / 2.0;

        glBegin(GL_POLYGON);        //Lazer horizontal stem
        glVertex2f(midPoint + 10, 25);
        glVertex2f(midPoint + 10, 35);
        glVertex2f(midPoint - 10, 35);
        glVertex2f(midPoint - 10, 25);
        glEnd();

        glPopMatrix();
}
void DrawLazerBeam() {

        float xMid = -(55 + 50) / 2.0;
        float yMid = (25 + 35) / 2.0;
        float mouseXEnd = -((-mouseX) + xOne);
        float mouseYEnd = -((-mouseY) + yOne);
        glLineWidth(5);  //----Laser beam width
        glColor3f(1, 0, 0);
        glBegin(GL_LINES);
        glVertex2f(xMid, yMid);
        glVertex2f(mouseXEnd, mouseYEnd);
        glEnd();
        glLineWidth(1);
}
```

```
void DrawStone(int StoneIndex)
{
        glPushMatrix();
        glLoadIdentity();
        switch (StoneIndex)             //CHANGE INDEX VALUE FOR DIFFERENT STONE VARIETY;
        {
        case 0:
                glTranslated(xStone[index], yStone[index], 0);
                glRotatef(stoneAngle, 0, 0, 1);
                glTranslated(0, 0, 0);
                glColor3f(0.4f, 0.0f, 0.0f);
                glScalef(35, 35, 1);
                glutSolidSphere(1, 9, 50);
                glLoadIdentity();
                glTranslated(xStone[index], yStone[index], 0);
                glRotatef(stoneAngle, 0, 0, 1);
                glTranslated(0, 0, 0);
                glScalef(60, 10, 1);
                glutSolidSphere(1, 5, 50);
                glLoadIdentity();
                glTranslated(xStone[index], yStone[index], 0);
                glRotatef(stoneAngle, 0, 0, 1);
                glTranslated(0, 0, 0);
                glScalef(10, 60, 1);
                glutSolidSphere(1, 5, 50);
                break;

        case 1:
                glColor3f(1.0f, 0.8f, 0.8f);
                glTranslated(xStone[index], yStone[index], 0);
                glRotatef(stoneAngle, 0, 0, 1);
                glTranslated(0, 0, 0);
                glScalef(15, 20, 1);
                glutSolidSphere(1, 9, 50);
                glLoadIdentity();
                glTranslated(xStone[index], yStone[index], 0);
                glRotatef(stoneAngle, 0, 0, 1);
                glTranslated(0, 0, 0);
                glScalef(40, 5, 1);
                glutSolidSphere(1, 5, 50);
                break;

        case 2:
                glColor3f(0.2f, 0.2f, 0.0f);
                glTranslated(xStone[index], yStone[index], 0);
                glRotatef(stoneAngle, 0, 0, 1);
                glTranslated(0, 0, 0);
                glScalef(60, 25, 1);
                glutSolidSphere(1, 9, 50);
```

```
			glLoadIdentity();
			glTranslated(xStone[index], yStone[index], 0);
			glRotatef(stoneAngle, 0, 0, 1);
			glTranslated(0, 0, 0);
			glScalef(25, 60, 1);
			glutSolidSphere(1, 9, 50);
			break;
		case 3:
			glColor3f(0.8f, 0.8f, 0.1f);
			glTranslated(xStone[index], yStone[index], 0);
			glRotatef(stoneAngle, 0, 0, 1);
			glTranslated(0, 0, 0);
			glScalef(35, 10, 1);
			glutSolidSphere(1, 10, 7);
			glLoadIdentity();
			glTranslated(xStone[index], yStone[index], 0);
			glRotatef(stoneAngle, 0, 0, 1);
			glTranslated(0, 0, 0);
			glScalef(50, 20, 1);
			glutSolidSphere(1, 5, 50);
			break;
		 case 4:
			glColor3f(0.26f, 0.26f, 0.26f);
			glTranslated(xStone[index], yStone[index], 0);
			glRotatef(stoneAngle, 0, 0, 1);
			glTranslated(0, 0, 0);
			glScalef(10, 55, 1);
			glutSolidSphere(1, 9, 50);
			glLoadIdentity();
			glTranslated(xStone[index], yStone[index], 0);
			glRotatef(stoneAngle, 0, 0, 1);
			glTranslated(0, 0, 0);
			glScalef(20, 10, 1);
			glutSolidSphere(1, 9, 50);
			glLoadIdentity();

			glTranslated(xStone[index], yStone[index], 0);
			glRotatef(stoneAngle + 45, 0, 0, 1);
			glTranslated(0, 0, 0);
			glScalef(25, 10, 1);
			glutSolidSphere(1, 9, 50);
			break;
		}
		glPopMatrix();
}
bool checkIfSpaceShipIsSafe() {
		for (int i = 0; i < MAX_STONES; i++) {
			if (stoneAlive[i] & ((xOne >= (xStone[i] / 2 - 70) && xOne <= (xStone[i] / 2 + 70) && yOne >= (yStone[i] /
2 - 18) && yOne <= (yStone[i] / 2 + 53)) || (yOne <= (yStone[i] / 2 - 20) && yOne >= (yStone[i] / 2 - 90) && xOne >=
(xStone[i] / 2 - 40) && xOne <= (xStone[i] / 2 + 40))))
                {
                        stoneAlive[i] = 0;
                        return false;
                }
        }
        return true;
}
void SpaceshipCreate() {
        glPushMatrix();
        glTranslated(xOne, yOne, 0);
        if (!checkIfSpaceShipIsSafe() && alienLife) {
                alienLife -= 10;
                xStart -= 23;
        }
        DrawSpaceshipDoom();
        glPushMatrix();
        glTranslated(4, 19, 0);
        DrawAlien();
        glPopMatrix();
        DrawSteeringWheel();
        DrawSpaceshipBody();
        DrawSpaceShipLazer();
        if (mButtonPressed) {
                DrawLazerBeam();
        }
        glEnd();
        glPopMatrix();
}
void DisplayHealthBar() {

        glColor3f(1, 0, 0);
        glBegin(GL_POLYGON);
        glVertex2f(-xStart, 700);
        glVertex2f(1200, 700);
        glVertex2f(1200, 670);
        glVertex2f(-xStart, 670);
        glEnd();
        char temp[40];
        glColor3f(0, 0, 1);
        sprintf_s(temp, sizeof(temp), "SCORE = %d", Score);

        displayRasterText(-1100, 600, 0.4, temp);//<---display variable score ?
        sprintf_s(temp, sizeof(temp), "  LIFE = %d", alienLife);
        displayRasterText(800, 600, 0.4, temp);
        sprintf_s(temp, sizeof(temp), "  LEVEL : %d", GameLvl);
        displayRasterText(-100, 600, 0.4, temp);
        glColor3f(1, 0, 0);
}
```

```
void startScreenDisplay()
{
	glLineWidth(50);
	SetDisplayMode(MENU_SCREEN);

	glColor3f(0, 0, 0);
	glBegin(GL_LINE_LOOP);		//Border
	glVertex3f(-750, -500, 0.5);
	glVertex3f(-750, 550, 0.5);
	glVertex3f(750, 550, 0.5);
	glVertex3f(750, -500, 0.5);
	glEnd();

	glLineWidth(1);

	glColor3f(1, 1, 0);
	glBegin(GL_POLYGON);				//START GAME PLOYGON
	glVertex3f(-200, 300, 0.5);
	glVertex3f(-200, 400, 0.5);
	glVertex3f(200, 400, 0.5);
	glVertex3f(200, 300, 0.5);
	glEnd();

	glBegin(GL_POLYGON);				//INSTRUCTIONS POLYGON
	glVertex3f(-200, 50, 0.5);
	glVertex3f(-200, 150, 0.5);
	glVertex3f(200, 150, 0.5);
	glVertex3f(200, 50, 0.5);
	glEnd();

	glBegin(GL_POLYGON);				//QUIT POLYGON
	glVertex3f(-200, -200, 0.5);
	glVertex3f(-200, -100, 0.5);
	glVertex3f(200, -100, 0.5);
	glVertex3f(200, -200, 0.5);
	glEnd();

	if (mouseX >= -100 && mouseX <= 100 && mouseY >= 150 && mouseY <= 200) {
		glColor3f(0, 0, 1);
		if (mButtonPressed) {
			startGame = true;
			gameOver = false;
			mButtonPressed = false;
		}
	}
	else
		glColor3f(0, 0, 0);

	displayRasterText(-100, 340, 0.4, (char*)"Start Game");
	if (mouseX >= -100 && mouseX <= 100 && mouseY >= 30 && mouseY <= 80) {
		glColor3f(0, 0, 1);
		if (mButtonPressed) {
			instructionsGame = true;
			mButtonPressed = false;
		}
	}
	else
		glColor3f(0, 0, 0);
	displayRasterText(-120, 80, 0.4, (char*)"Instructions");

	if (mouseX >= -100 && mouseX <= 100 && mouseY >= -90 && mouseY <= -40) {
		glColor3f(0, 0, 1);
		if (mButtonPressed) {
			gameQuit = true;
			mButtonPressed = false;
		}
	}
	else
		glColor3f(0, 0, 0);
	displayRasterText(-100, -170, 0.4, (char*)"    Quit");

}
void GameScreenDisplay()
{
	SetDisplayMode(GAME_SCREEN);
	DisplayHealthBar();
	glScalef(2, 2, 0);
	if (alienLife) {
		SpaceshipCreate();
	}
	else {
		gameOver = true;
		instructionsGame = false;
		startScreen = false;
	}								//<----------------------gameover screen

	StoneGenerate();

}
void readFromFile() {

	FILE* fp;
	if (fopen_s(&fp, "HighScoreFile.txt", "r") != 0) {
		// Handle error opening file
	}

	int i = 0;
```

```
        if (fp != NULL) {
                while (fread(&ch, sizeof(char), 1, fp)) {
                        highScore[i++] = ch;
                }
                highScore[i] = '\0';
        }
        fclose(fp);
}
void writeIntoFile() {                                          //To write high score on to file
        FILE* fp;
        if (fopen_s(&fp, "HighScoreFile.txt", "w") != 0) {
                // Handle error opening file
        }

        int i = 0;
        char temp[40];
        if (fp != NULL) {
                int n = Score;
                while (n) {
                        ch = (n % 10) + '0';
                        n /= 10;
                        temp[i++] = ch;
                }
                temp[i] = '\0';
                _strrev(temp);

                puts(temp);
                if (temp[0] == '\0')
                        temp[i++] = '0', temp[i++] = '\0';
                fwrite(temp, sizeof(char) * i, i, fp);
        }
        fclose(fp);
}
void GameOverScreen()
{
        SetDisplayMode(MENU_SCREEN);
        glColor3f(0, 0, 0);
        glLineWidth(50);
        glBegin(GL_LINE_LOOP);          //Border
        glVertex3f(-650, -500, 0.5);
        glVertex3f(-650, 520, 0.5);
        glVertex3f(650, 520, 0.5);
        glVertex3f(650, -500, 0.5);
        glEnd();

        glLineWidth(1);
        stoneTranslationSpeed = 5;
        glColor3f(0, 1, 0);
        glBegin(GL_POLYGON);                            //GAME OVER
```

```
glVertex3f(-550, 810, 0.5);
glVertex3f(-550, 610, 0.5);
glVertex3f(550, 610, 0.5);
glVertex3f(550, 810, 0.5);
glEnd();

glColor3f(1, 1, 0);
glBegin(GL_POLYGON);                              //RESTART POLYGON
glVertex3f(-200, 50, 0.5);
glVertex3f(-200, 150, 0.5);
glVertex3f(200, 150, 0.5);
glVertex3f(200, 50, 0.5);
glEnd();

glBegin(GL_POLYGON);                              //QUIT POLYGON
glVertex3f(-200, -200, 0.5);
glVertex3f(-200, -100, 0.5);
glVertex3f(200, -100, 0.5);
glVertex3f(200, -200, 0.5);
glEnd();


displayRasterText(-300, 640, 0.4, (char*)"G A M E    O V E R ! ! !");
glColor3f(0, 0, 0);
char temp[40];

sprintf_s(temp,sizeof(temp) ,"Score : %d", Score);
displayRasterText(-100, 340, 0.4, temp);
readFromFile();
char temp2[40];
if (atoi(highScore) < Score) {
        writeIntoFile();
        sprintf_s(temp2,sizeof(temp2), "Highest Score :%d", Score);
}
else
        sprintf_s(temp2,sizeof(temp2), "Highest Score :%s", highScore);

displayRasterText(-250, 400, 0.4, temp2);

if (mouseX >= -100 && mouseX <= 100 && mouseY >= 25 && mouseY <= 75) {
        glColor3f(0, 0, 1);
        if (mButtonPressed) {                              //Reset game default values
                startGame = true;
                gameOver = false;
                mButtonPressed = false;
                initializeStoneArray();
                alienLife = 100;
                xStart = 1200;
                Score = 0;
```

```
				GameLvl = 1;
				GameScreenDisplay();
			}
		}
		else
			glColor3f(0, 0, 0);
		displayRasterText(-70, 80, 0.4, (char*)"Restart");

		if (mouseX >= -100 && mouseX <= 100 && mouseY >= -100 && mouseY <= -50) {
			glColor3f(0, 0, 1);
			if (mButtonPressed) {
				exit(0);
				mButtonPressed = false;
			}
		}
		else
			glColor3f(0, 0, 0);
		displayRasterText(-100, -170, 0.4, (char*)"    Quit");

}
void StoneGenerate() {
		if (xStone[0] >= 1200) {      // If the last screen hits the end of the screen, then go to the next level
			GameLvl++;
			stoneTranslationSpeed += 3;
			Score += 50;
			initializeStoneArray();
			GameScreenDisplay();
		}

		for (int i = 0; i < MAX_STONES; i++) {
			index = i;

			if (mouseX <= (xStone[i] / 2 + 20) && mouseX >= (xStone[i] / 2 - 20) && mouseY >= (yStone[i] / 2 - 20)
&& mouseY <= (yStone[i] / 2 + 20) && mButtonPressed) {
				if (stoneAlive[i]) {   // If alive, kill the stone
					stoneAlive[i] = 0;
					Score++;
					if (Score % 10 == 0) {    // Check if the score is a multiple of 10
						GameLvl++;            // Increment the level
						stoneTranslationSpeed += 3;    // Adjust any game parameters related to level
increase
					}
					if (Score % 3 == 0) {
						stoneTranslationSpeed += 1;     // Rate of increase of game speed
					}
				}
			}
			xStone[i] += stoneTranslationSpeed;
			if (stoneAlive[i])             // Stone is alive
```

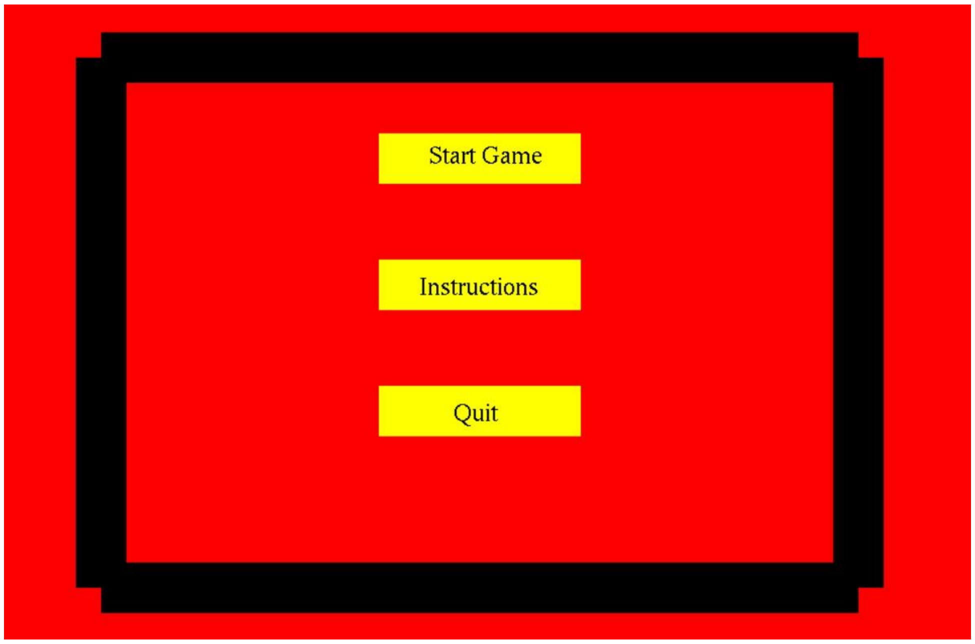
```
				DrawStone(randomStoneIndices[i]);
		}
		stoneAngle += stoneRotationSpeed;
		if (stoneAngle > 360) stoneAngle = 0;
}

void backButton() {
		if (mouseX <= -450 && mouseX >= -500 && mouseY >= -275 && mouseY <= -250) {
			glColor3f(0, 0, 1);
			if (mButtonPressed) {
				mButtonPressed = false;
				instructionsGame = false;
				startScreenDisplay();
			}
		}
		else glColor3f(0, 0, 0);
		displayRasterText(-1000, -550, 0, (char*)"Back");
}
void InstructionsScreenDisplay()
{

		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
		SetDisplayMode(MENU_SCREEN);
		//colorBackground();
		glColor3f(0, 0, 0);
		displayRasterText(-900, 400, 0.4, (char*)"Key 'w' to move up.");
		displayRasterText(-900, 300, 0.4, (char*)"Key 's' to move down.");
		displayRasterText(-900, 200, 0.4, (char*)"Key 'd' to move right.");
		displayRasterText(-900, 100, 0.4, (char*)"Key 'a' to move left.");
		displayRasterText(-900, 0.0, 0.4, (char*)"Left mouse click to shoot laser");
		//displayRasterText(-900 ,-100 ,0.4 ,"The packet can be placed only when 's' is pressed before.");
		displayRasterText(-900, -200, 0.4, (char*)"You Get 1 point for shooting each objet and 50 points for completing each
lvl ");
		displayRasterText(-900, -270, 0.4, (char*)"The Objective is to score maximum points");
		backButton();
		if (previousScreen)
			nextScreen = false, previousScreen = false; //as set by backButton()


}
void display() {

		glClear(GL_COLOR_BUFFER_BIT);
		glViewport(0, 0, 1200, 700);

		if (startGame && !gameOver)
			GameScreenDisplay();

		else if (instructionsGame)
```

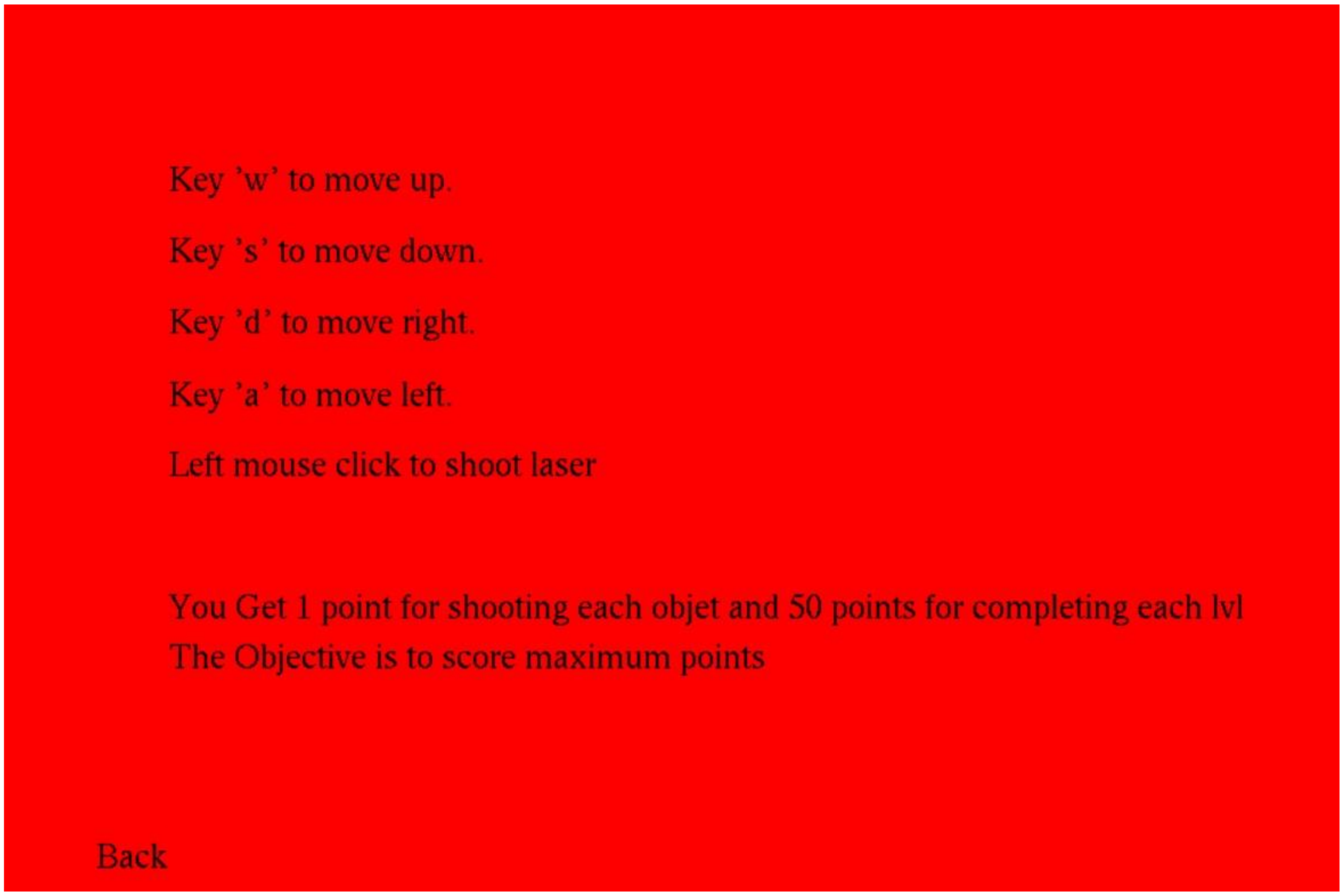
```
			InstructionsScreenDisplay();

		else if (gameOver)
			GameOverScreen();

		//Make spaceship bigger
		else if (startScreen) {

			startScreenDisplay();
			if (gameQuit || startGame || optionsGame || instructionsGame) {
				//startScreen = false;

				if (startGame) {
					SetDisplayMode(GAME_SCREEN);
					startScreen = false;

				}
				else if (gameQuit)
					exit(0);

			}
			else if (instructionsGame) {
				SetDisplayMode(GAME_SCREEN);
				InstructionsScreenDisplay();
			}
		}

		//Reset Scaling values
		glScalef(1 / 2, 1 / 2, 0);
		glFlush();
		glLoadIdentity();
		glutSwapBuffers();
	}
	void somethingMovedRecalculateLaserAngle() {

		float mouseXForTan = (-50 - mouseX) + xOne;
		float mouseYForTan = (35 - mouseY) + yOne;
		float LaserAngleInRadian = atan(mouseYForTan / mouseXForTan);
		LaserAngle = (180 / PI) * LaserAngleInRadian;
	}
	void keys(unsigned char key, int x, int y)
	{
		//if(key=='w' && key=='d' ){xOne+=0.5;yOne+=0.5;}
		if (key == 'd') xOne += SPACESHIP_SPEED;
		if (key == 'a') xOne -= SPACESHIP_SPEED;
		if (key == 'w') { yOne += SPACESHIP_SPEED; }
		if (key == 's') { yOne -= SPACESHIP_SPEED; }
		if (key == 'd' || key == 'a' || key == 'w' || key == 's')
			somethingMovedRecalculateLaserAngle()
```

```
        display();

}
void myinit()
{
        glClearColor(0.5, 0.5, 0.5, 0);
        glColor3f(1.0, 0.0, 0.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        gluOrtho2D(-1200, 1200, -700, 700);             //<-----CHANGE THIS TO GET EXTRA SPACE
        //  gluOrtho2D(-200,200,-200,200);
        glMatrixMode(GL_MODELVIEW);
}
void passiveMotionFunc(int x, int y) {

        //when mouse not clicked
        mouseX = float(x) / (m_viewport[2] / 1200.0) - 600.0;  //converting screen resolution to ortho 2d spec
        mouseY = -(float(y) / (m_viewport[3] / 700.0) - 350.0);

        //Do calculations to find value of LaserAngle
        somethingMovedRecalculateLaserAngle();
        display();
}
void mouseClick(int buttonPressed, int state, int x, int y) {

        if (buttonPressed == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
                mButtonPressed = true;
        else
                mButtonPressed = false;
        display();
}
void UpdateColorIndexForSpaceshipLights(int value)
{
        CI = (CI + 1) % 3;                              //Color Index swapping to have rotation effect
        display();
        glutTimerFunc(250, UpdateColorIndexForSpaceshipLights, 0);
}
void idleCallBack() {                           //when no mouse or keybord pressed
        display();
}
```

```
void main(int argc, char** argv) {

        FILE* fp;
        if (fopen_s(&fp, "HighScoreFile.txt", "r") != 0) {
                // Handle error opening file
        }
        //check if HighScoreFile.txt exist if not create
        if (fp != NULL)
                fclose(fp);
        else
                writeIntoFile();

        glutInit(&argc, argv);
        glutInitWindowSize(1200, 700);
        glutInitWindowPosition(90, 0);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutTimerFunc(50, UpdateColorIndexForSpaceshipLights, 0);
        glutCreateWindow("GUARDIAN OF GALAXY");
        glutDisplayFunc(display);
        glutKeyboardFunc(keys);
        glutPassiveMotionFunc(passiveMotionFunc);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glutIdleFunc(idleCallBack);
        glutMouseFunc(mouseClick);
        glGetIntegerv(GL_VIEWPORT, m_viewport);
        myinit();
        SetDisplayMode(GAME_SCREEN);
        initializeStoneArray();
        glutMainLoop();
}
```

5. Output/ Screen shots:




Key ’w’ to move up.

Key ’s’ to move down.

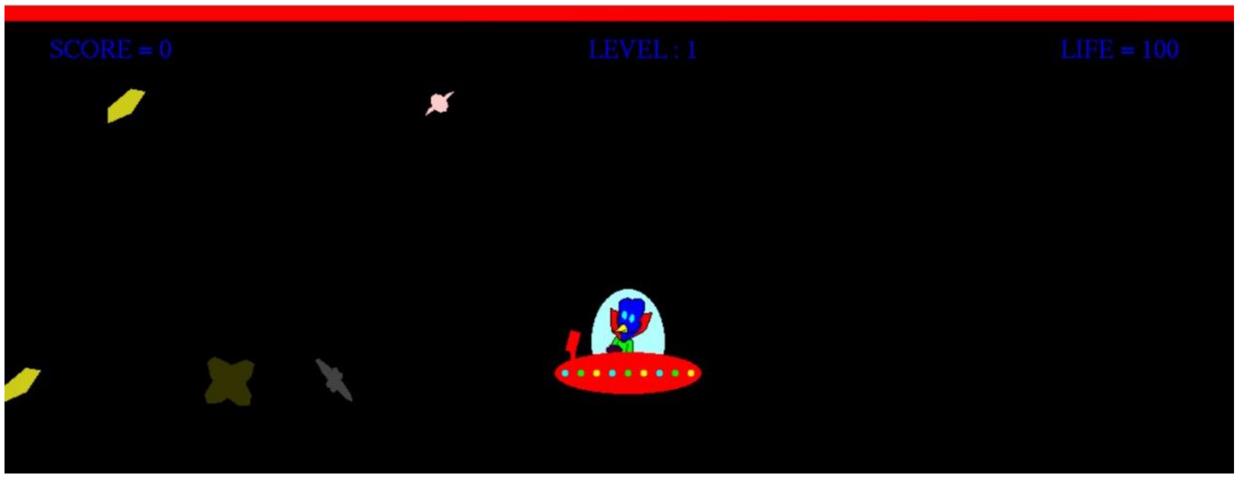
Key ’d’ to move right.

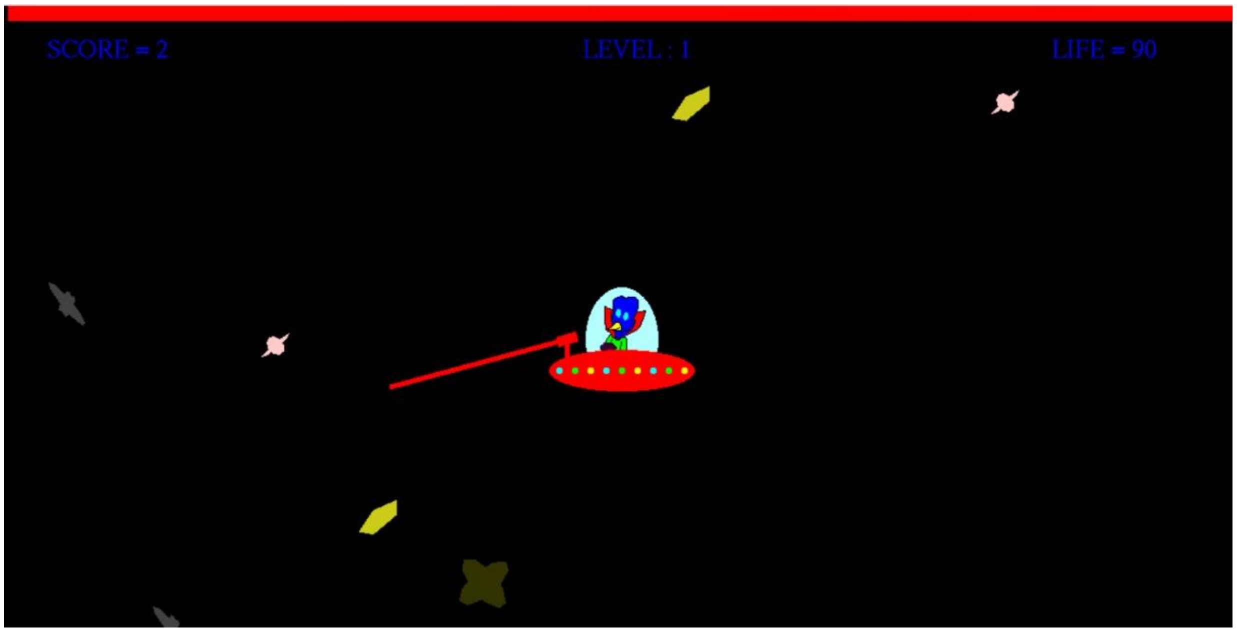
Key ’a’ to move left.

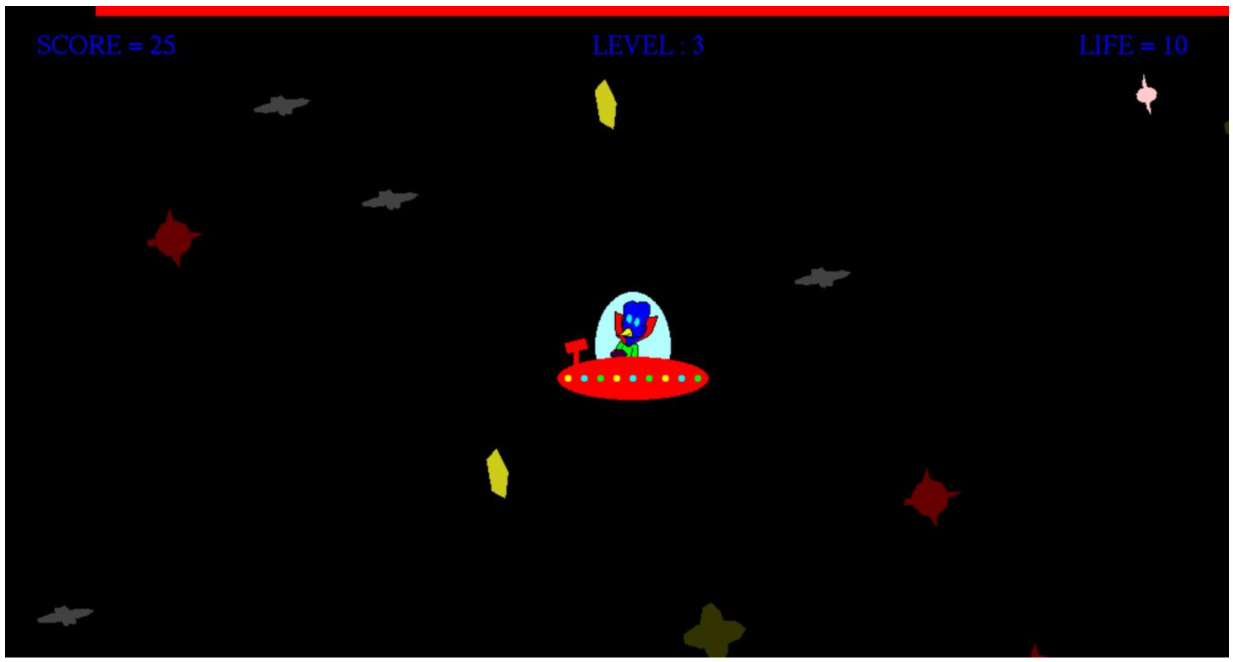
Left mouse click to shoot laser

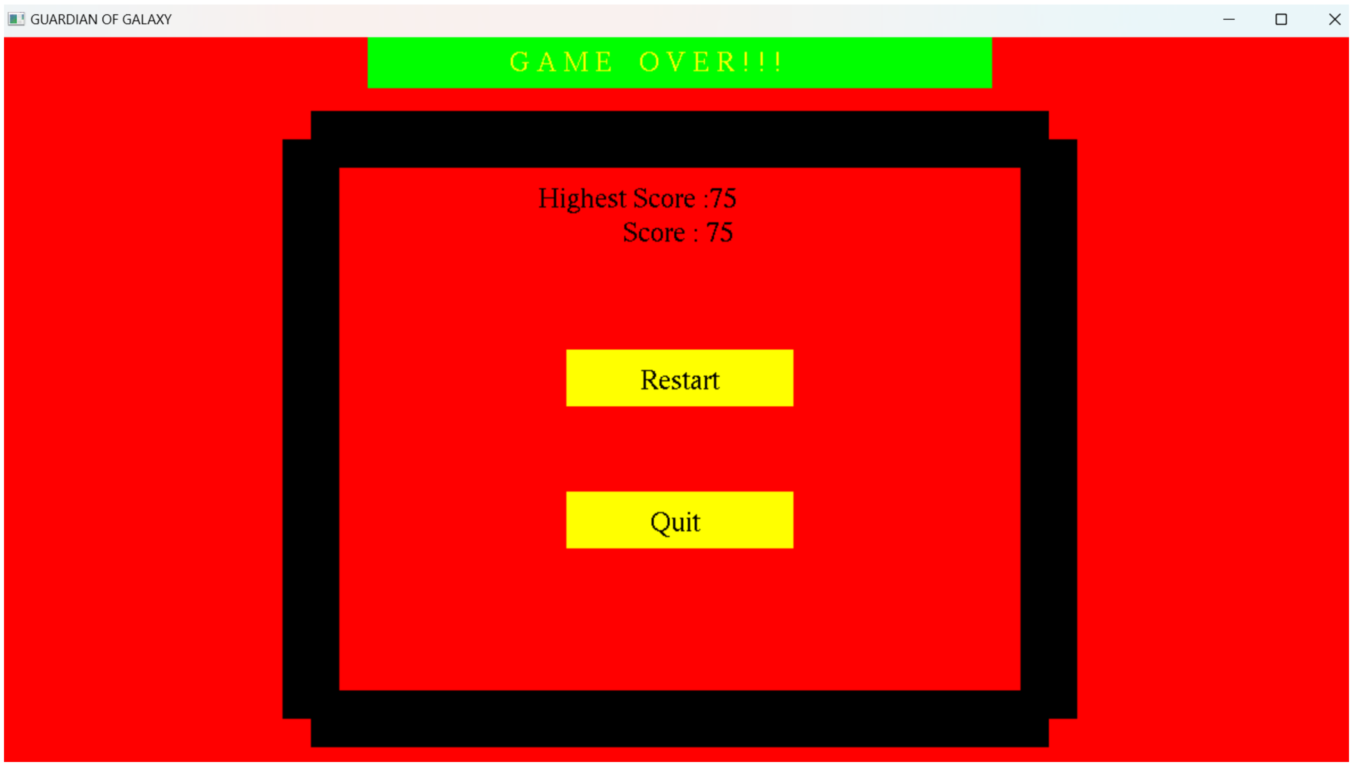
You Get 1 point for shooting each objet and 50 points for completing each lvl
The Objective is to score maximum points

Back


SCORE = 0
LEVEL : 1
LIFE = 100


SCORE = 2
LEVEL : 1
LIFE = 90


SCORE = 25
LEVEL : 3
LIFE = 10



Highest Score :75
Score : 75