# Computer Graphics (UCS505) Project on :

ALIEN GAME USING OPENGL

# Submitted By:

|  |  |
| --- | --- |
| Sushant Vij | 102003759 |
| Sahil Chhabra | 102003766 |
| Srimon Ghosh | 101916128 |

**Group No.-1**

**B.E. Third Year – COE**

**Submitted To:**

**Ms. Jyoti Rani**



**Computer Science and Engineering Department Thapar Institute of Engineering and Technology Patiala – 147001**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Description** | **Page No.** |
| 1. | Introduction to Project | 3 |
| 2. | Computer Graphics concepts used | 4 |
| 3. | User Defined Functions | 5 |
| 4. | Code | 13 |
| 5. | Output/ Screen shots | 28 |

**Introduction :**

Game Space Shooter is an action-packed arcade-style video game set in space, where the player controls a spaceship and battles against waves of enemy spacecraft. The game involves navigating through space, dodging enemy fire, and shooting down enemy spaceships.

The game was first introduced in the early days of gaming and gained popularity in the 1980s and 1990s. The genre has since evolved with advancements in technology, providing a more immersive gaming experience with improved graphics, sound effects, and gameplay mechanics.

This project aims to replicate the classic space shooter game with simple OpenGL and C++ concepts. The player will have control of their spaceship's movements and weapons as they take on enemy spacecraft, aiming to survive each wave and progress through increasingly difficult levels. The game will be designed to provide a challenging and engaging experience for players, with a range of power-ups and obstacles to keep things interesting.

**Computer Graphics concepts used**

In our project, various computer graphics concepts are employed. The key concepts used in this project are:

1. 2D Graphics Rendering: In this project, 2D graphics rendering is used to create a

graphical representation of the aliens and spaceships.

2. OpenGL: OpenGL (Open Graphics Library) is a cross-platform graphics API used

for rendering 2D and 3D graphics. In this project, OpenGL functions are employed

to make the aliens and spaceships viewable.

3. GLFW/GLEW: GLFW provides a lightweight cross-platform API for creating

and managing OpenGL contexts and handling input, while GLEW simplifies the

process of loading and using OpenGL extensions in a cross-platform manner.

4. Coordinate System and Transformations: The project utilizes a Cartesian coordinate system to position the spaceships on the screen. The spaceships are represented by their (x, y) coordinates and their state is updated by modifying their coordinates based on the current state of the system. OpenGL transformation functions such as glMatrixMode, glLoadIdentity, and glOrtho are employed to establish the coordinate system and guarantee accurate display of the aliens and spaceships on the screen.

5. Animation and Game Loop: The game loop is an essential component of the

project, responsible for updating the game state and rendering the graphics. The

game loop is implemented using OpenGL's timer function, glutTimerFunc, which

ensures the smooth and consistent movement of the snake and other game objects.

The vehicle's animation is achieved by updating its position and redrawing the

game objects at regular intervals, creating the illusion of continuous motion.

# User Defined Functions :

# void displayRasterText(float x, float y, float z, const char\* stringToDisplay): The function displayRasterText() takes in the position (x, y, z) and a string of characters stringToDisplay, and displays the text on the screen using rasterization. The characters are iterated over and displayed one by one using glutBitmapCharacter() with the GLUT\_BITMAP\_TIMES\_ROMAN\_24 font.

# void init(): The init() function sets up the background color, the projection matrix using gluOrtho2D(), and the modelview matrix using glMatrixMode() and glLoadIdentity().

# void introScreen(): The introScreen() function is responsible for rendering the introductory screen of the game. First, it clears the color buffer using glClear(GL\_COLOR\_BUFFER\_BIT) to remove any previously rendered pixels on the screen. Next, it sets the color of the text that will be displayed using glColor3f() function. Then, it calls the displayRasterText() function to render the text on the screen. The text is positioned using the x, y, and z coordinates. The displayRasterText() function uses the glRasterPos3f() function to set the position of the text on the screen, and the glutBitmapCharacter() function to render each character of the string on the screen. Once all the text has been rendered, the function calls glFlush() to ensure that all commands are executed as soon as possible, and glutSwapBuffers() to swap the front and back buffers, making the newly rendered image visible on the screen.

# void startScreenDisplay(): This function is a start screen display function for a game that uses the OpenGL library. It draws a border and three colored polygons representing the Start Game, Instructions, and Quit options. The function also checks if the mouse cursor is hovering over any of the polygons and if the mouse button is pressed. If the Start Game polygon is clicked, the function sets the variables alienLife1 and alienLife2 to 100 and changes the viewPage to GAME. If the Instructions polygon is clicked, the viewPage is changed to INSTRUCTIONS, and if the Quit polygon is clicked, the program is exited. The function uses various OpenGL functions such as glBegin(), glEnd(), and glColor3f() to draw the polygons and set their color. The displayRasterText() function is used to display text on the screen. Overall, this function serves as the start screen for the game and allows the user to choose between different options to start or exit the game.

# void backButton(): This function is a back button function for a game that uses the OpenGL library. The function checks if the mouse cursor is hovering over the back button polygon. If the mouse cursor is hovering over the polygon and the mouse button is pressed, the function sets the viewPage to MENU, indicating that the user wants to go back to the main menu. The mButtonPressed flag is set to false to prevent the action from being repeated, and the display is updated using the glutPostRedisplay() function.

# If the mouse cursor is not hovering over the back button polygon, the color of the polygon is set to red using glColor3f() function, indicating that the button is inactive. The function uses the displayRasterText() function to display the word "Back" on the screen. Overall, this function serves as a back button that allows the user to return to the main menu from other screens or pages in the game.

# void instructionsScreenDisplay(): The `instructionsScreenDisplay()` function is responsible for displaying the instructions screen of a game. It clears the screen and sets the color to red. It then displays various text instructions for the players, including how to move their characters and how to shoot, as well as the objective of the game. The function also calls `backButton()` to display a button that allows the player to go back to the main menu.

# void DrawAlienBody(bool isPlayer1): This function is responsible for drawing the body of an alien in a game. The function takes a boolean parameter `isPlayer1`, which is used to determine the color of the alien body. If `isPlayer1` is true, the body color is set to green (0, 1, 0), otherwise, it is set to yellow (1, 1, 0). The function then uses `glBegin()` and `glEnd()` to define and draw a polygon using the `AlienBody` array, which contains the vertices of the body polygon. After that, it sets the color to black (0, 0, 0) and draws an outline of the polygon using `glLineWidth()` and `glBegin()`, and `glEnd()`. Finally, it draws a line effect using `glBegin()` and `glEnd()`.

# void DrawAlienCollar(): The function begins by setting the color of the collar to red using the glColor3f() function, which takes three floating-point values between 0 and 1 to specify the red, green, and blue components of the color. Then, the glBegin() function starts defining a polygon using the GL\_POLYGON primitive type. The for-loop iterates 20 times, and at each iteration, the glVertex2fv() function is called to specify a vertex of the polygon. The AlienCollar array is used to provide the coordinates for each vertex. After defining the polygon, the color is set to black, and another glBegin() function starts defining a line strip using the GL\_LINE\_STRIP primitive type. The same loop is used to specify each vertex of the line strip, again using the AlienCollar array to provide the coordinates. Finally, the glEnd() function is called to end the drawing operation.

# void DrawAlienFace(bool isPlayer1): This is a function called "DrawAlienFace" that takes a boolean parameter called "isPlayer1". The function is probably a part of a larger program using OpenGL graphics library, as it uses functions such as "glBegin", "glColor3f", and "glEnd" that are specific to OpenGL. The function draws an alien face using the vertex coordinates stored in the array "ALienFace". It first sets the color of the face to blue using "glColor3f(0, 0, 1)" if "isPlayer1" is true, or green using "glColor3f(0,1,0)" otherwise (the latter code is currently commented out). It then draws a filled polygon using the vertex coordinates stored in "ALienFace". After that, it draws the outline of the face using black color by reusing the same vertex coordinates. Finally, the function adds an "ear effect" by drawing a thin line strip using three additional vertices with specific coordinates. These vertices create a small curve that is supposed to represent an ear.

# void DrawAlienBeak(): This is a function called "DrawAlienBeak". It also uses OpenGL graphics library functions such as "glBegin", "glColor3f", and "glEnd". The function draws an alien beak using the vertex coordinates stored in the array "ALienBeak". It first sets the color of the beak to yellow using "glColor3f(1, 1, 0)". It then draws a filled polygon using the vertex coordinates stored in "ALienBeak". After that, it draws the outline of the beak using black color by reusing the same vertex coordinates. Overall, this function seems to be drawing an alien's beak as part of a larger alien figure or scene.

# void DrawAlienEyes(bool isPlayer1): This is a function called "DrawAlienEyes" that also uses OpenGL graphics library functions such as "glPushMatrix", "glRotated", "glTranslated", "glScalef", "glutSolidSphere", and "glPopMatrix". It takes a boolean parameter called "isPlayer1" but currently, it is not used. The function draws two alien eyes. It sets the color of the eyes to cyan using "glColor3f(0, 1, 1)". It then uses several transformations to position and shape the eyes. For the left eye, it first rotates it by -10 degrees around the z-axis using "glRotated(-10, 0, 0, 1)", then translates it to (-6, 32.5, 0) using "glTranslated(-6, 32.5, 0)". It then scales the eye in the x and y directions using "glScalef(2.5, 4, 0)" and finally draws a solid sphere with a radius of 1 using "glutSolidSphere(1, 20, 30)". For the right eye, it uses a similar set of transformations, but with different translation and rotation values. It first rotates it by -1 degrees around the z-axis using "glRotated(-1, 0, 0, 1)", then translates it to (-8, 36, 0) using "glTranslated(-8, 36, 0)". It then scales the eye in the x and y directions using "glScalef(2.5, 4, 0)" and finally draws a solid sphere with a radius of 1 using "glutSolidSphere(1, 100, 100)". Overall, this function seems to be drawing two alien eyes, positioned and shaped in a specific way, as part of a larger alien figure or scene.

# void DrawAlien(bool isPlayer1): This is a function called "DrawAlien" that also uses the previously defined functions "DrawAlienBody", "DrawAlienCollar", "DrawAlienFace", "DrawAlienBeak", and "DrawAlienEyes". It takes a boolean parameter called "isPlayer1". The function draws a complete alien figure by calling the previously defined functions for drawing different parts of the alien. Specifically, it first calls "DrawAlienBody" to draw the alien body, passing in the "isPlayer1" parameter. Then it calls "DrawAlienCollar" to draw the alien collar. After that, it calls "DrawAlienFace" to draw the alien face, passing in the "isPlayer1" parameter. Then it calls "DrawAlienBeak" to draw the alien beak. Finally, it calls "DrawAlienEyes" to draw the alien eyes, passing in the "isPlayer1" parameter. Overall, this function seems to be a high-level function that coordinates the drawing of all the parts required to make an alien figure. The "isPlayer1" parameter appears to be used to determine the color of some parts of the alien, but it is not used consistently across all the drawing functions.

# void DrawSpaceshipBody(bool isPlayer1): This function appears to draw the body of a spaceship using OpenGL. It takes a boolean parameter isPlayer1 which determines the color of the spaceship. If isPlayer1 is true, the spaceship is drawn in red, otherwise it is drawn in a shade of purple. The function first sets the color for the base of the spaceship using glColor3f(). It then uses glPushMatrix() to push the current matrix onto the stack and perform subsequent transformations on a copy of the matrix. It then scales the matrix using glScalef() to create a sphere with a radius of 1, which is then multiplied by the given values to create an ellipsoid that represents the body of the spaceship. The body is then drawn using glutSolidSphere(). The function then uses glPushMatrix() again to push the current matrix onto the stack and perform subsequent transformations on a copy of the matrix. It scales the matrix by a factor of 3 in the x and y directions, and 1 in the z direction to draw the lights on the spaceship. It then translates the matrix by -20 in the x direction to position the first light. The function uses a loop to draw nine lights, with each light being translated 5 units in the x direction from the previous one. The colors of the lights are determined using LightColor[] array and the value of CI variable. Finally, the function uses glPopMatrix() twice to pop the current matrix off the stack and restore the original matrix.

# void DrawSteeringWheel(): The function "DrawSteeringWheel" uses OpenGL to draw a wireframe sphere that represents a steering wheel. The sphere is scaled, translated, and colored, and the function assumes that the necessary OpenGL libraries and headers have been included.

# void DrawSpaceshipDoom(): The function "DrawSpaceshipDoom" draws a light blue sphere using OpenGL that represents a spaceship. The sphere is scaled and translated to give the appearance of a spaceship, and the function assumes that the necessary OpenGL libraries and headers have been included.

# void DrawLaser(int x, int y, bool dir[]): The function "DrawLaser" takes three parameters: an x-coordinate, a y-coordinate, and a boolean array representing the direction of the laser. The function uses OpenGL to draw a red line that represents a laser beam originating from the given coordinates and extending either vertically or horizontally to the edge of the screen, depending on the direction specified in the boolean array. The function assumes that the necessary OpenGL libraries and headers have been included. There are also some commented out lines that appear to be related to matrix manipulation, but they are not currently being used.

# void SpaceshipCreate(int x, int y, bool isPlayer1): The function "SpaceshipCreate" takes three parameters: an x-coordinate, a y-coordinate, and a boolean value indicating whether the spaceship belongs to player 1. The function uses OpenGL to create a spaceship by calling several other functions to draw different components of the spaceship, such as the steering wheel and the spaceship body. The function also calls a function to draw an alien character sitting inside the spaceship. There are some commented-out lines that suggest the presence of additional functionality related to checking for collisions and firing a laser beam, but they are not currently being used. The function assumes that the necessary OpenGL libraries and headers have been included.

# void DisplayHealthBar1(): The function "DisplayHealthBar1" uses OpenGL to display a health bar for the first player's alien character. The function first sets the color to white and displays the current value of the alien's life using sprintf\_s to format the text as a string. The function then sets the color to red, but it doesn't appear to draw anything in this color. The function assumes that the necessary OpenGL libraries and headers have been included.

# void DisplayHealthBar2(): The function "DisplayHealthBar2" uses OpenGL to display a health bar for the second player's alien character. The function first sets the color to white and displays the current value of the alien's life using sprintf\_s to format the text as a string. The function then sets the color to red, but it doesn't appear to draw anything in this color. The function assumes that the necessary OpenGL libraries and headers have been included.

# void checkLaserContact(int x, int y, bool dir[], int xp, int yp, bool player1): The `checkLaserContact` function checks if a laser shot fired from a spaceship intersects with another spaceship, and reduces the life of the corresponding alien by 5 if it does. It calculates the values of `m`, `k`, `a`, `b`, `c`, and `d` based on the equations for the line and circle, and checks if `d` is greater than or equal to 0 to determine if there is an intersection.

# void gameScreenDisplay(): The `gameScreenDisplay` function displays the game screen with two spaceships and their health bars. It first calls two functions to display the health bars for each spaceship, then scales up the entire screen by a factor of 2. If `alienLife1` is greater than 0, the function calls `SpaceshipCreate` to display the first spaceship at coordinates `xOne` and `yOne`, with the boolean `true` indicating that it belongs to player 1. If `laser1` is true, the function calls `DrawLaser` to draw the laser fired by the first spaceship in the direction stored in `laser1Dir`, and then calls `checkLaserContact` to check if the laser hits the second spaceship (at coordinates `-xTwo` and `yTwo`). If `alienLife2` is less than or equal to 0, the variable `viewPage` is set to `GAMEOVER`. If `alienLife2` is greater than 0, the function first uses `glPushMatrix` and `glPopMatrix` to flip the screen horizontally, and then calls `SpaceshipCreate` to display the second spaceship at coordinates `xTwo` and `yTwo`, with the boolean `false` indicating that it belongs to player 2. If `laser2` is true, the function calls `DrawLaser` to draw the laser fired by the second spaceship in the direction stored in `laser2Dir`, and then calls `checkLaserContact` to check if the laser hits the first spaceship (at coordinates `-xOne` and `yOne`). If `alienLife2` is less than or equal to 0, the variable `viewPage` is set to `GAMEOVER`. If `viewPage` is equal to `GAMEOVER`, the coordinates of both spaceships are reset to (500, 0).

# void displayGameOverMessage(): The `displayGameOverMessage` function displays the game over message on the screen. The function first sets the color to yellow using `glColor3f`, then declares a pointer variable `message` of type `const char\*`. If `alienLife1` is greater than 0, the message pointer is set to the string "Game Over! Player 1 won the game". Otherwise, it is set to the string "Game Over! Player 2 won the game". The function then calls `displayRasterText` to display the message on the screen at coordinates (-350, 600) with a font size of 0.4.

# void keyOperations(): This code defines a function called `keyOperations()` which is responsible for updating the game state based on user keyboard inputs. The function checks if the Enter key is pressed and if the current game view is the intro, then it changes the view to the menu. If the current view is the game, then the function updates the state of the game based on user inputs. It checks if the 'c' key is pressed, which indicates the second player is firing the laser. If so, it checks if the 'w' or 's' keys are also pressed to set the direction of the laser accordingly. If 'c' is not pressed, the function checks for the movement keys ('d', 'a', 'w', and 's') for the second player's spaceship movement. Similarly, the function also checks for the 'm' key, which indicates the first player is firing the laser. If so, it checks if the 'i' or 'k' keys are also pressed to set the direction of the laser accordingly. If 'm' is not pressed, the function checks for the movement keys ('l', 'j', 'i', and 'k') for the first player's spaceship movement.

# void display(): Within the function, glClearColor() is commented out, which sets the color used when clearing the screen. The keyOperations() function is called to handle keyboard input, and glClear() is called to clear the color buffer. A switch statement is used to determine which screen to display based on the value of the viewPage variable. If viewPage is set to INTRO, then the introScreen() function is called to display the introductory screen. If viewPage is set to MENU, then the startScreenDisplay() function is called to display the menu screen. If viewPage is set to INSTRUCTIONS, then the instructionsScreenDisplay() function is called to display the instructions screen. If viewPage is set to GAME, then the gameScreenDisplay() function is called to display the game screen, and the glScalef() function is called to reset the scaling values. If viewPage is set to GAMEOVER, then the displayGameOverMessage() function is called to display the game over message, and the startScreenDisplay() function is called to display the menu screen. Finally, glFlush() is called to ensure that all OpenGL commands are executed, glLoadIdentity() is called to reset the modelview matrix to the identity matrix, and glutSwapBuffers() is called to swap the front and back buffers.

# void passiveMotionFunc(int x, int y): This code defines a function called `passiveMotionFunc()`. The function takes two arguments, `x` and `y`, which represent the current position of the mouse cursor on the screen. The function calculates the position of the mouse in relation to the screen resolution and then converts it to orthographic 2D specifications. The calculation involves dividing the `x` coordinate by the width of the viewport (stored in `m\_viewport[2]`), multiplying by 1200 (the width of the orthographic view), and subtracting 600 (half the width of the orthographic view). The calculation for `y` is similar, but the value is negated to account for the difference in orientation between screen coordinates and OpenGL coordinates. After calculating the mouse position, the function likely performs additional calculations to determine the value of the `LaserAngle` variable. Finally, the function calls `glutPostRedisplay()` to request that the window be redrawn to reflect any changes made by the function. Overall, this function appears to be responsible for tracking the position of the mouse cursor and updating the graphics accordingly.

# void mouseClick(int buttonPressed, int state, int x, int y): The function takes four arguments, buttonPressed, state, x, and y. buttonPressed represents which mouse button was pressed (left, right, or middle). state represents whether the button was pressed down or released. x and y represent the current position of the mouse cursor on the screen. The function first checks if the left mouse button was pressed down (buttonPressed == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN). If so, it sets the boolean variable mButtonPressed to true. Otherwise, it sets mButtonPressed to false. After setting mButtonPressed, the function calls glutPostRedisplay() to request that the window be redrawn to reflect any changes made by the function.

# void keyPressed(unsigned char key, int x, int y): The function takes three arguments, key, x, and y. key represents the ASCII value of the key that was pressed. x and y represent the current position of the mouse cursor on the screen. The function sets the value of the keyStates array at the index corresponding to the pressed key to true. This allows other parts of the program to check whether a specific key is currently being pressed. After setting the keyStates array, the function calls glutPostRedisplay() to request that the window be redrawn to reflect any changes made by the function.

# void refresh(): The function does not take any arguments. It calls glutPostRedisplay() to request that the window be redrawn. This function might be used to refresh the display periodically.

# void keyReleased(unsigned char key, int x, int y): The function takes three arguments, key, x, and y. key represents the ASCII value of the key that was released. x and y represent the current position of the mouse cursor on the screen. The function sets the value of the keyStates array at the index corresponding to the released key to false. This allows other parts of the program to check whether a specific key is currently being pressed.

# int main(int argc, char\*\* argv): The program first calls glutInit() to initialize the GLUT library with the command-line arguments argc and argv. It then sets the display mode to single buffering and RGB color mode using glutInitDisplayMode(). The program sets the initial window position to (0, 0) using glutInitWindowPosition() and sets the window size to 1200 x 600 pixels using glutInitWindowSize(). A window is created with the title "Space Shooter" using glutCreateWindow(). The init() function is then called to set up the initial state of the program. glutIdleFunc(refresh) sets refresh() as the function to be called when the program is idle, which in turn calls glutPostRedisplay() to refresh the display. The program then sets up several callback functions to handle keyboard and mouse input using glutKeyboardFunc(), glutKeyboardUpFunc(), glutMouseFunc(), and glutPassiveMotionFunc(). These functions are responsible for handling events such as key presses, key releases, mouse clicks, and mouse movement. glGetIntegerv(GL\_VIEWPORT, m\_viewport) retrieves the viewport parameters of the window and stores them in the m\_viewport array. Finally, glutDisplayFunc(display) is called to specify the function display() as the function to be called when the window needs to be redrawn, and glutMainLoop() enters the main loop of the program, which handles events and updates the display as needed. Overall, this main function sets up the initial state of the program, sets up callback functions to handle user input, and enters the main event loop to handle events and update the display.

# Code:

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

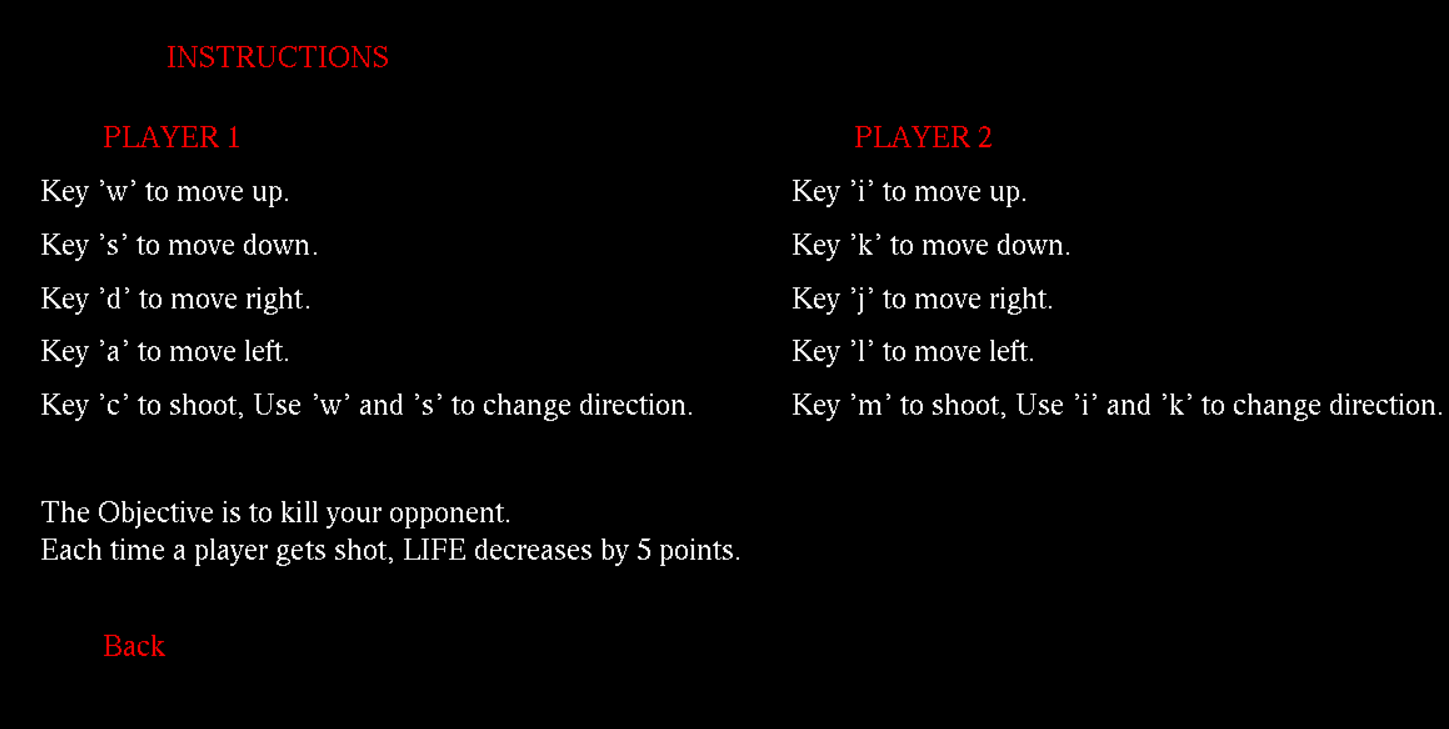
# Screenshots and Working :

# 

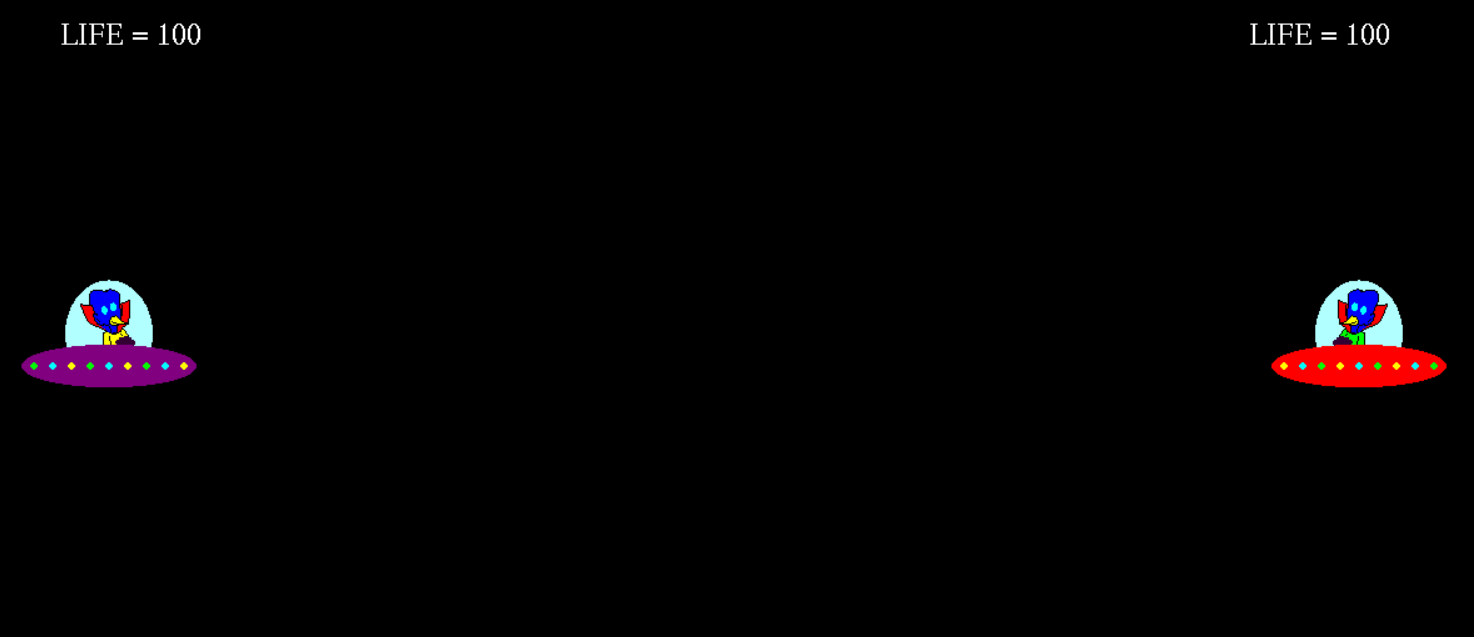
This is the front greeting page of the game. The user is given this page every time the code is run.

# 

This is the actual game page that opens after the user presses ‘Enter’ on the front greeting page. The game remains in idle state until the user chooses any assigned option of ‘Start Game’,’Instructions’,’Quit’.



This is the instruction page,from where a player can get information about his gameplay instructions.



This is the actual game page that opens after the user presses ‘Start Game’ on the front greeting page. The game remains in idle state until the user presses any assigned buttons on the keyboard.

