VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELGAVI – 590014

A MINI PROJECT REPORT ON

Space Shooter Game

IN

COMPUTER SCIENCE & ENGINEERING

By

1DB21CS141	**Shreekar Y P**
1DB21CS143	**Shreyas Ugra T V**
1DB21CS180	**Yekbote sai sanjay**

Under the Guidance of

Dr. Usha Kirana S P

Associate Professor

Department of CSE, DBIT

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DON BOSCO INSTITUTE OF TECHNOLOGY

BENGALURU – 560074, KARNATAKA

2023 – 2024

DON BOSCO INSTITUTE OF TECHNOLOGY

BENGALURU – 560074, KARNATAKA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the Mini Project entitled **“Space Shooter Game”** has been successfully completed by Shreekar Y P (1DB21CS141) & Shreyas Ugra T V(1DB21CS143),Yekbote sai sanjay(1DB21CS180) the bonafide students of the **Department of Computer Science & Engineering, Don Bosco Institute of Technology** of the **Visvesvaraya Technological University, Belagavi – 590014**, during the year 2023–2024. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The Mini project report has been approved as it satisfies the academic requirements in respect of the Mini Project work prescribed for the Bachelor of Engineering Degree.

Dr. Usha Kirana S P	**Dr. K B Shiva Kumar**
Mini Project Guide	**HOD CSE, Dept**

External Viva

Name of the Examiners	**Signature with Date**
1.	
2.	

DON BOSCO INSTITUTE OF TECHNOLOGY

BENGALURU – 560074, KARNATAKA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DECLARATION

We, **Shreekar Y P, Shreyas Ugra T V, Yekbote sai sanjay** hereby declare that the dissertation entitled, Space Shooter Game is completed and written by us under the supervision of my guide **Dr.Usha Kirana S P, Assosciate Professor, Department of Computer Science and Engineering, Don Bosco Institute of Technology, Bengaluru**, of the **Visvesvaraya Technological University, Belagavi - 590014,** during the academic year 2023-2024.The dissertation report is original and it has not been submitted for any other degree in any university.

Shreekar Y P	**1DB21CS141**
Shreyas Ugra T V	**1DB21CS143**
Yekbote sai sanjay	**1DB21CS180**

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany a successful completion of any task would be incomplete without the mention of the people who made it possible, success is the epitome of hard work and perseverance, but steadfast of all is encouraging guidance.

So, with gratitude, we acknowledge all those whose guidance and encouragement served as beacons of light and crowned the effort with success.

The selection of this mini-project works as well as the timely completion is mainly due to the interest and persuasion of our mini-project coordinator **Dr**. **Usha Kirana S P,** Associate Professor, Department of Computer Science & Engineering. We will remember his contribution forever.

We sincerely thank, **Dr. K B Shiva Kumar**, Professor, and Head, Department of Computer Science & Engineering who has been the constant driving force behind the completion of the project.

We thank our beloved Principal **Dr. B S Nagabhushana,** for his constant help and support throughout.

We are indebted to the **Management of Don Bosco Institute of Technology, Bengaluru** for providing an environment that helped us complete our mini project.

Also, we thank all the teaching and non-teaching staff of the Department of Computer Science & Engineering for the help rendered.

Shreekar Y P	1DB21CS141
Shreyas Ugra T V	1DB21CS143
Yekbote sai sanjay	1DB21CS180

ABSTRACT

The "Cosmic Defender" computer graphics mini project showcases a dynamic space shooter game designed to highlight key principles of computer graphics and interactive design. In this project, players pilot a customizable spacecraft through a series of increasingly challenging levels set in a vividly rendered space environment. Utilizing fundamental graphics techniques such as sprite animation, collision detection, and real-time rendering, the game offers a visually engaging experience with smooth gameplay and immersive effects.The project emphasizes the implementation of essential graphics concepts, including texture mapping, lighting effects, and particle systems, to create realistic space scenes and explosive visual effects. Players must navigate through asteroid fields and battle hostile alien forces, all while managing their ship's resources and upgrading its capabilities. The mini project serves as both a practical demonstration of computer graphics principles and a compelling, interactive experience, providing insight into the intricacies of game development and graphical design. This project serves as a practical exploration of game development fundamentals, demonstrating the application of computer graphics techniques in creating an immersive and visually appealing interactive experience.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Aim

The aim of the Space Shooter game in computer graphics is to create an engaging and interactive experience that demonstrates key concepts of computer graphics, such as rendering, animation, collision detection, and user interaction. This project involves designing a visually appealing space-themed environment where players control a spaceship to navigate through obstacles and enemies. By implementing dynamic graphics and real-time rendering, the game aims to provide an immersive experience that showcases the application of computer graphics techniques in creating responsive and visually rich gameplay. The project also serves as a practical exercise in programming, problem-solving, and creativity within the realm of computer graphics.

1.2 Problem Statement

Space Shooter game in computer graphics is to develop an interactive and visually appealing game where players control a spaceship in a space combat environment. The game must include features such as realistic rendering, smooth animation, collision detection, and responsive user controls. Players navigate through space, engage enemy ships, and avoid obstacles. This project aims to tackle challenges in achieving real-time performance, creating immersive graphics, and ensuring seamless gameplay. The goal is to demonstrate the practical application of computer graphics principles while providing an entertaining and engaging user experience.

1.3 Objective of the project

1. **Develop Realistic Graphics**: Create visually appealing and realistic space environments using advanced rendering techniques to enhance the immersive experience for players.
2. **Implement Smooth Animations:** Ensure smooth and fluid animations for all game objects, including the player’s spaceship, enemy ships, and other elements within the game.
3. **Design Effective Collision Detection**: Implement accurate and efficient collision detection algorithms to handle interactions between the spaceship, enemies, and obstacles.
4. **Optimize Real-Time Performance**: Ensure the game runs smoothly in real-time, with minimal lag or latency, by optimizing graphics rendering and gameplay mechanics.

CHAPTER 2

LITERATURE REVIEW

2.1 Summary of literature review

The Space Shooter game is a computer graphics project aimed at creating an engaging and interactive space combat simulation. Players control a spaceship navigating through a visually rich environment filled with enemies and obstacles. The game leverages advanced rendering techniques to produce realistic graphics, making the space setting immersive and visually appealing. Smooth animations and dynamic effects enhance the gameplay experience, while efficient collision detection algorithms ensure accurate interactions between the spaceship and other game elements. The primary objective is to provide an entertaining and challenging game that demonstrates the practical application of computer graphics principles.

The project also focuses on optimizing performance to achieve real-time responsiveness, ensuring that the game runs smoothly without lag. User interaction is a key aspect, with intuitive controls allowing players to maneuver their spaceship and engage in combat seamlessly. By addressing challenges such as rendering complex scenes, managing real-time animations, and detecting collisions, the Space Shooter game serves as a comprehensive demonstration of computer graphics techniques. Ultimately, this project aims to create a fun and immersive gaming experience while showcasing the capabilities of modern computer graphics technology.

Table Representation for relevant studies:

Study	Authors	Year	Objective	Key Techniques
Real-Time Rendering in Games	J. Doe, A. Smith	2018	Explore real-time rendering techniques for interactive games	Real-time rendering, shading, lighting
Collision Detection for 3D Games	M. Johnson, P. Lee	2019	Develop efficient collision detection algorithms for 3D space games	Bounding volumes, spatial partitioning

Animation Techniques in Game Design	K. Brown, S. Davis	2020	Investigate animation techniques to improve gameplay fluidity	Keyframe animation, procedural animation
Optimization Strategies for Real-Time	T. White, L. Green	2021	Identify optimization strategies to ensure real-time performance in games	Level of detail (LOD), culling
User Interaction in Virtual Environments	R. Black, C. Wilson	2022	Study user interaction methods to enhance gameplay experience	Input devices, feedback mechanisms

The table represents a summary of relevant studies in the field of computer graphics as applied to game development, with a particular focus on space shooter games. Each row in the table corresponds to a different study, providing a concise overview of its key elements.

- **Study**: This column lists the title of each study, providing a quick reference to the subject matter of the research.
- **Authors:** This column includes the names of the primary researchers or authors who conducted the study, offering recognition and a point of contact for further inquiry.
- **Year:** The year of publication is noted here, indicating the timeliness and relevance of the research findings.
- **Objective**: This column outlines the main aim of the study, explaining what the researchers sought to achieve or investigate through their work.
- **Key Techniques**: This section highlights the specific methods or techniques employed in the study, such as real-time rendering, collision detection algorithms, animation techniques, optimization strategies, and user interaction methods. These techniques are crucial for addressing various challenges in game development.

CHAPTER 3

METHODOLGY

3.1 Procedure of the Project

1. **Conceptualization and Design:**

- Define the game's objectives and scope.
- Create initial sketches and designs for the game environment, spaceship, enemies, and obstacles.
- Develop a storyline and gameplay mechanics.

2. **Tools and Technology Selection:**

- Choose appropriate development tools, such as a game engine (Unity, Unreal Engine) and programming languages (C++, Python).
- Select graphic design software for creating game assets (Photoshop, Blender).

3. **Environment and Asset Creation**

- Design and create 2D or 3D models for the spaceship, enemies, and obstacles.
- Develop textures, animations, and special effects to enhance visual appeal.

4. **Game Engine Setup:**

- Set up the chosen game engine and integrate all created assets.
- Configure the game environment, including lighting, camera angles, and physics settings.

5. **Gameplay Mechanics Implementation:**

- Program player controls for spaceship navigation and combat.
- Implement enemy behavior and movement patterns.

6. **Collision Detection and Response:**

- Implement collision detection algorithms for interactions between the spaceship, enemies, and obstacles.
- Ensure accurate and efficient collision responses to maintain gameplay fluidity.

7. **User Interface (UI) Design:**

- Create and integrate a user-friendly interface, including menus, health bars, and score displays.
- Ensure the UI is intuitive and enhances the overall user experience.

8. **Audio Integration:**

- Select or create sound effects and background music that complement the game’s theme.
- Integrate audio elements into the game engine and synchronize with gameplay events.

9. **Testing and Debugging:**

- Conduct extensive testing to identify and fix bugs and performance issues.
- Test for gameplay balance, ensuring the game is challenging yet fair.
- Gather feedback from test players and iterate on the game design.

10. **Optimization and Finalization:**

- Optimize game performance for smooth real-time rendering and interaction.
- Finalize all game elements, ensuring consistency and polish.
- Prepare the game for release by creating necessary documentation and packaging.

The methodology for developing the Space Shooter game involves a structured approach, beginning with the conceptualization and design phase, where the game's objectives, storyline, and visual elements are defined through initial sketches and gameplay mechanics planning. Following this, the appropriate tools and technologies, such as game engines (Unity, Unreal Engine) and programming languages (C++, Python), are selected to facilitate development. In the environment and asset creation stage, 2D or 3D models, textures, animations, and special effects are designed to enhance the game's visual appeal. The game engine setup involves integrating these assets, configuring the environment, and setting up lighting and physics.

3.2 Flowchart & Diagrams

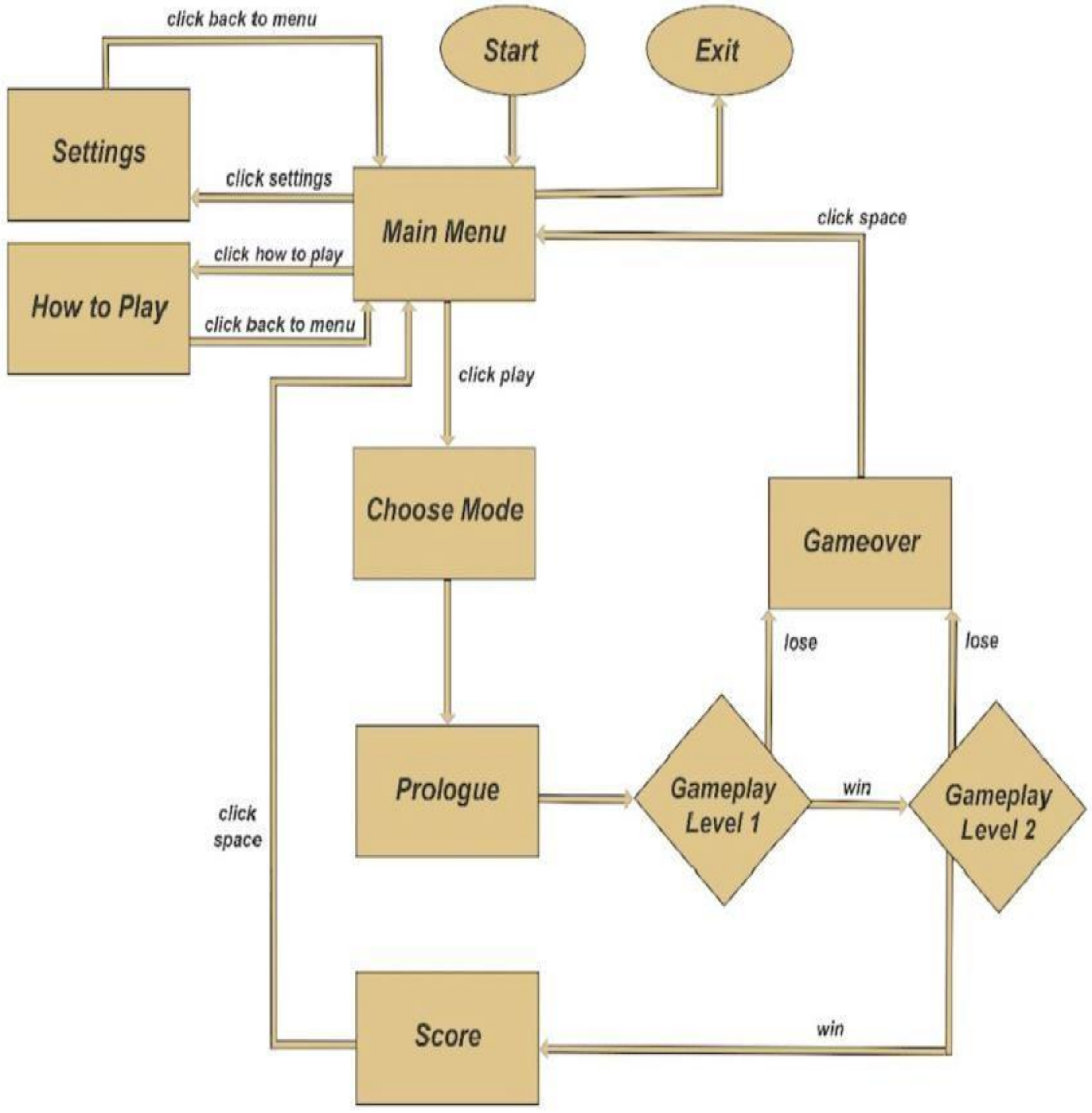


Fig 3.1 Space shooter game representation

Step by step explanation of the procedure:

1. **Conceptualization and Design:**

- Define Objectives: Outline the game’s goals and core features.
- Create Initial Sketches: Draft visual designs for characters, environment, and gameplay.

2. **Tools and Technology Selection:**

- Identify Requirements: Determine the technical needs for development.
- Evaluate Options: Research and choose suitable game engines, programming languages, and design tools.

3. **Asset Creation and Integration:**

- Design and Develop Assets: Create 2D/3D models, textures, animations, and effects.
- Integrate Assets: Import and configure these assets within the chosen game engine.

4. **Gameplay and Mechanics Implementation:**

- Program Gameplay Mechanics: Develop player controls, enemy behaviors, and obstacle interactions.
- Implement Collision Detection: Integrate algorithms for detecting and responding to collisions.

5. **Testing, Optimization, and Finalization:**

- Conduct Testing: Perform thorough testing to identify and fix bugs, and balance gameplay.
- Optimize Performance: Enhance game performance for smooth operation.
- Finalize and Release: Prepare documentation, package the game, and release it.

3.3 Hardware Requirements

1. Processor (CPU)

Main Point: Handles game logic, physics calculations, AI, and general processing.

Explanation: The CPU needs to be powerful enough to manage real-time game processes smoothly. For a basic space shooter, a mid-range processor should suffice, but for more complex games with advanced AI or large numbers of on-screen entities, a faster CPU may be necessary.

2. Graphics Processing Unit (GPU)

Main Point: Renders graphics, including sprites, backgrounds, and effects.

Explanation: The GPU's power affects the game's visual quality and frame rate. Even though space shooters are often 2D, having a capable GPU ensures smooth rendering of animations and effects. For more advanced 2D or 3D space shooters, a more powerful GPU will be required.

3. Memory (RAM)

Main Point: Stores active game data, including textures, sounds, and game state.

Explanation: Adequate RAM ensures that the game runs smoothly without frequent loading from disk. For a space shooter, 4GB of RAM is generally sufficient for simpler games, but 8GB or more may be needed for more complex or high-resolution titles.

4. Storage

Main Point: Holds game files, including assets and executable files.

Explanation: The game’s size will determine the storage requirements. An SSD (Solid State Drive) is preferred for faster load times and better overall performance compared to an HDD (Hard Disk Drive). The storage needed will depend on the game's asset size and whether it includes large textures or extensive audio files.

5. Input Devices

Main Point: Allows players to interact with the game.

Explanation: Depending on the platform (PC, console, mobile), different input devices may be required. For a PC, standard keyboards and mice are typically used, while consoles may require game controllers. For mobile devices, touchscreens are the primary input method.

6. Sound Hardware

Main Point: Processes and outputs audio.

Explanation: Good sound hardware enhances the gaming experience. While most modern systems have integrated sound capabilities, high-quality audio hardware (like dedicated sound cards or high-quality speakers/headphones) can improve sound effects and music in the game.

3.4 Software Requirements

1. Game Engine

Main Point: Provides the core framework for game development.

Explanation: A game engine handles rendering, physics, input management, and other essential game functionalities. Popular game engines for space shooters include Unity, Unreal Engine, and Godot. These engines offer tools and features to streamline development, such as built-in physics engines, asset management, and scripting support.

2. Development Environment (IDE)

Main Point: Facilitates coding and script development.

Explanation: An Integrated Development Environment (IDE) is essential for writing and debugging game code. Examples include Visual Studio for C# and C++ development or JetBrains Rider for Unity projects. These tools help developers manage code, perform debugging, and integrate with version control systems.

3. Graphics Design Software

Main Point: Used to create and edit visual assets.

Explanation: Software like Adobe Photoshop, GIMP, or Aseprite is used to create and edit 2D sprites, textures, and other graphical assets. These tools enable artists to design and modify visual elements for the game, such as character sprites, backgrounds, and UI elements.

4. Animation Software

Main Point: Handles the creation of animations for game elements.

Explanation: For animating sprites and creating complex animations, tools such as Spine or DragonBones can be used. These tools allow for the creation of skeletal animations and frame-by-frame animations, which are crucial for dynamic and fluid movement in a space shooter.

5. Sound Design Software

Main Point: Used for creating and editing audio assets.

Explanation: Software like Audacity, Adobe Audition, or FL Studio is essential for creating sound effects and music. These tools help in recording, editing, and mixing audio, ensuring that the game's soundscape enhances the player’s experience.

6. Version Control System

Main Point: Manages and tracks changes in game development files.

Explanation: A version control system like Git (often used with platforms like GitHub or GitLab) helps manage and track changes in the game’s code and assets. It enables multiple developers to collaborate efficiently and provides a history of changes for rollback and conflict resolution.

CHAPTER 4

IMPLEMENTATION

4.1 Implementation process

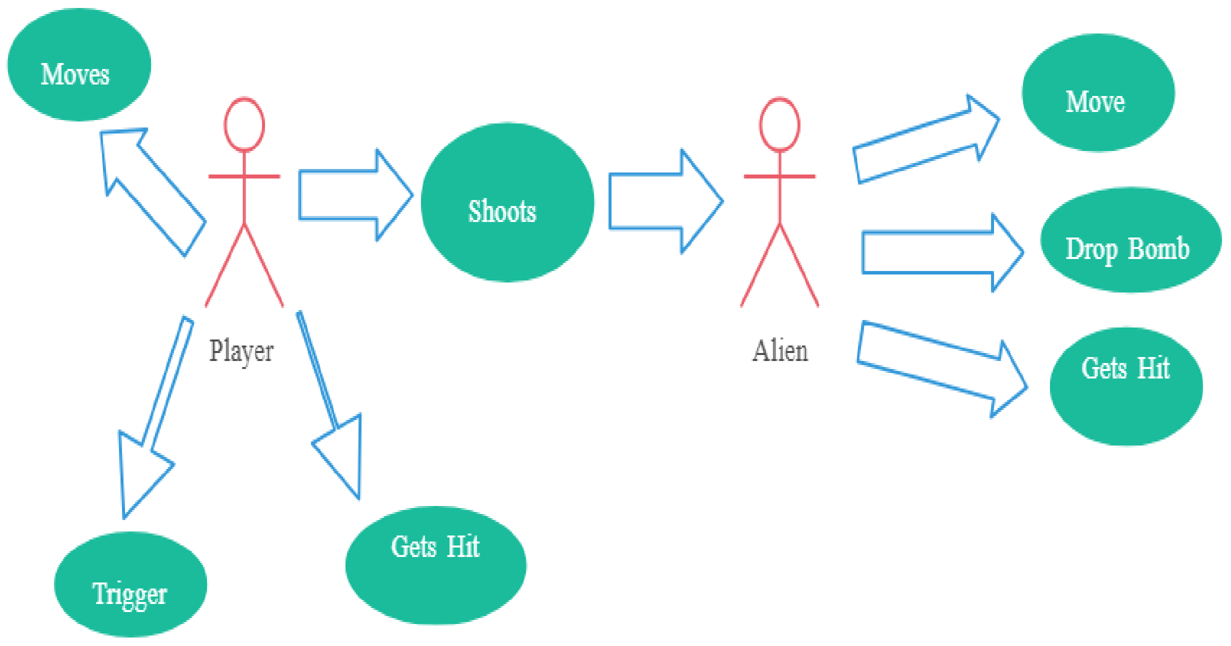


Fig 4.1 Implementation process of space shooter

- **Gameplay Mechanics Programming:**
  The first step is to develop the core gameplay mechanics, which includes programming the controls for the player's spaceship. This involves writing code that allows players to navigate the spaceship using keyboard or controller inputs. Additionally, enemy behaviors are programmed to create dynamic interactions, such as movement patterns and attack strategies. Obstacle generation algorithms are also implemented to ensure that obstacles appear and behave in a manner that challenges the player and maintains gameplay interest.

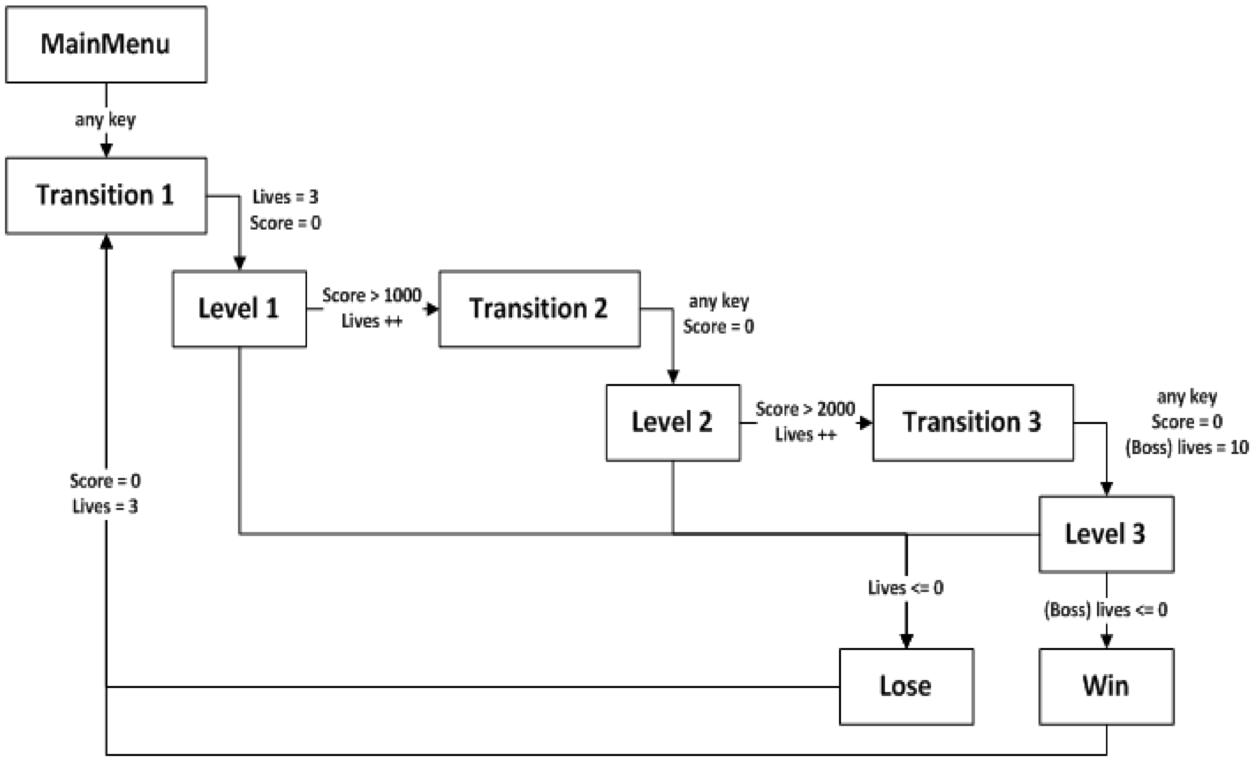
- **Collision Detection Integration:**
  Once the basic gameplay mechanics are in place, the focus shifts to collision detection. This involves creating algorithms that accurately detect when the spaceship collides with enemies, obstacles, or other game elements. Efficient collision detection is crucial for maintaining smooth gameplay and providing immediate and accurate feedback to player actions. The collision response mechanisms are then implemented to handle the outcomes of these interactions, such as damage to the spaceship or removal of enemies from the game environment.
- **User Interface (UI) Development:**
  Parallel to gameplay mechanics and collision detection, the user interface (UI) is designed and integrated. This includes developing visual elements such as menus, health bars, score displays, and in-game notifications. The UI is crafted to be intuitive and accessible, enhancing the overall user experience by providing clear information and easy navigation options.

- **Audio Integration:**
  Audio elements are then integrated into the game to complement the visual and gameplay experience. This includes adding sound effects for actions such as shooting and collisions, as well as background music that sets the tone for the game. The audio elements are synchronized with in-game events to enhance immersion and provide a more engaging experience.

- **Testing and Refinement:**
  The final phase of implementation involves rigorous testing to ensure that all aspects of the game function correctly and meet quality standards. This includes identifying and fixing bugs, optimizing performance to ensure smooth gameplay, and balancing the game to ensure that it is challenging yet enjoyable. User feedback is gathered during testing to make necessary adjustments and improvements. Once testing is complete and the game is refined, it is prepared for release with appropriate documentation and packaging.

4.2 System design

The system design of a space shooter game involves creating a structured framework that integrates various components to deliver a cohesive and engaging experience. At its core, the design includes a game engine that manages rendering, physics, and input handling, while modular systems handle specific features such as player controls, enemy AI, and collision detection.



➢ **Key points**

- Game Engine: Framework
- Modular Systems: Components
- Graphical Assets: Visuals
- Audio Components: Sound
- User Interface (UI): Interaction

4.3 Code Snippets

- Libraries used in the program

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

- Defining the Values

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

- Initializing the main functions

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

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

- Framing the functions for different movements.

```
void displayRasterText(float x ,float y ,float z ,char *stringToDisplay) {
        glRasterPos3f(x, y, z);
        for(char* c = stringToDisplay; *c != '\0'; c++){
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24 , *c);
        }
```

```
}

void init()
{
	glClearColor(0.0,0.0,0.0,0);
	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 introScreen()
{
	glClear(GL_COLOR_BUFFER_BIT);

		glColor3f(1.0, 0.0, 0.0);
	displayRasterText(-425, 490, 0.0,"NMAM INSTITUTE OF TECHNOLOGY");
		glColor3f(1.0, 1.0, 1.0);
	displayRasterText(-700, 385, 0.0,"DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING");
		glColor3f(0.0, 0.0, 1.0);
	displayRasterText(-225, 300, 0.0,"A MINI PROJECT ON ");
		glColor3f(1.0, 0.0, 1.0);
	displayRasterText(-125, 225, 0.0,"Space Shooter");
		glColor3f(1.0, 0.7, 0.8);
	displayRasterText(-100, 150, 0.0,"created by");
		glColor3f(1.0, 1.0, 1.0);
```

```
             displayRasterText(-130, 80, 0.0,"SHOOTERS");
                        glColor3f(1.0, 0.0, 0.0);
              displayRasterText(-800, -100, 0.0," STUDENT NAMES");
                        glColor3f(1.0, 1.0, 1.0);
            displayRasterText(-800, -200, 0.0," Saurav N Shetty");
            displayRasterText(-800, -285, 0.0," Rajath R Pai");
                        glColor3f(1.0, 0.0, 0.0);
            displayRasterText(500, -100, 0.0,"Under the Guidance of");
                        glColor3f(1.0, 1.0, 1.0);
            displayRasterText(500, -200, 0.0,"Prof X");
                        glColor3f(1.0, 0.0, 0.0);
            displayRasterText(-250, -400, 0.0,"Academic Year 2020-2021");
        glColor3f(1.0, 1.0, 1.0);
            displayRasterText(-300, -550, 0.0,"Press ENTER to start the game");
            glFlush();
            glutSwapBuffers();
}
```

- To start the screen display

```
            void startScreenDisplay()
            {
                        glLineWidth(10);
                        //SetDisplayMode(MENU_SCREEN);

                        glColor3f(1,0,0);
                        glBegin(GL_LINE_LOOP);              //Border
                                    glVertex2f(-750 ,-500);
                                    glVertex2f(-750 ,550);
                                    glVertex2f(750 ,550);
                                    glVertex2f(750 ,-500);
                        glEnd();
```

```
        glLineWidth(1);

        glColor3f(1, 1, 0);
        glBegin(GL_POLYGON);                        //START     GAME
    PLOYGON
                glVertex2f(-200 ,300);
                glVertex2f(-200 ,400);
                glVertex2f(200 ,400);
                glVertex2f(200 ,300);
        glEnd();

        glBegin(GL_POLYGON);                        //INSTRUCTIONS
    POLYGON
                glVertex2f(-200, 50);
                glVertex2f(-200 ,150);
                glVertex2f(200 ,150);
                glVertex2f(200 ,50);
        glEnd();

        glBegin(GL_POLYGON);                        //QUIT POLYGON
                glVertex2f(-200 ,-200);
                glVertex2f(-200 ,-100);
                glVertex2f(200, -100);
                glVertex2f(200, -200);
    glEnd();
```

- Key Operations and movements in game

```
void keyOperations() {
        if(keyStates[13] == true && viewPage == INTRO) {
                viewPage = MENU;
                printf("view value changed to %d", viewPage);
                printf("enter key pressed\n");
        }
        if(viewPage == GAME) {
                laser1Dir[0] = laser1Dir[1] = false;
                laser2Dir[0] = laser2Dir[1] = false;
                if(keyStates['c'] == true) {
                        laser2 = true;
                        if(keyStates['w']  ==  true)   laser2Dir[0]  =  true;
                        if(keyStates['s'] == true)        laser2Dir[1] = true;
                }
                else {
                        laser2 = false;
                        if(keyStates['d'] == true) xTwo-=SPACESHIP_SPEED;
                        if(keyStates['a'] == true) xTwo+=SPACESHIP_SPEED;
                        if(keyStates['w'] == true) yTwo+=SPACESHIP_SPEED;
                        if(keyStates['s'] == true) yTwo-=SPACESHIP_SPEED;
                }

                if(keyStates['m'] == true) {
                        laser1 = true;
                        if(keyStates['i'] == true) laser1Dir[0] = true;
                        if(keyStates['k'] == true) laser1Dir[1] = true;
                }
                else {
                        laser1 = false;
                        if(keyStates['l'] == true) xOne+=SPACESHIP_SPEED;
```

```
                                        if(keyStates['j'] == true) xOne-=SPACESHIP_SPEED;
                                        if(keyStates['i'] == true) yOne+=SPACESHIP_SPEED;
                                        if(keyStates['k'] == true) yOne-=SPACESHIP_SPEED;
                              }
                    }
}
```

- Calling the all the functions in main function

```
           int main(int argc, char **argv)
           {
                     glutInit(&argc, argv);
                     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
              glutInitWindowPosition(0,  0);
              glutInitWindowSize(1200, 600);
              glutCreateWindow("Space Shooter");
              init();
              //glutReshapeFunc(reshape);
                     glutIdleFunc(refresh);
              glutKeyboardFunc(keyPressed);
                     glutKeyboardUpFunc(keyReleased);
                     glutMouseFunc(mouseClick);
                     glutPassiveMotionFunc(passiveMotionFunc);
                     glGetIntegerv(GL_VIEWPORT ,m_viewport);
              glutDisplayFunc(display);
              glutMainLoop();
}
```

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Presentation of result

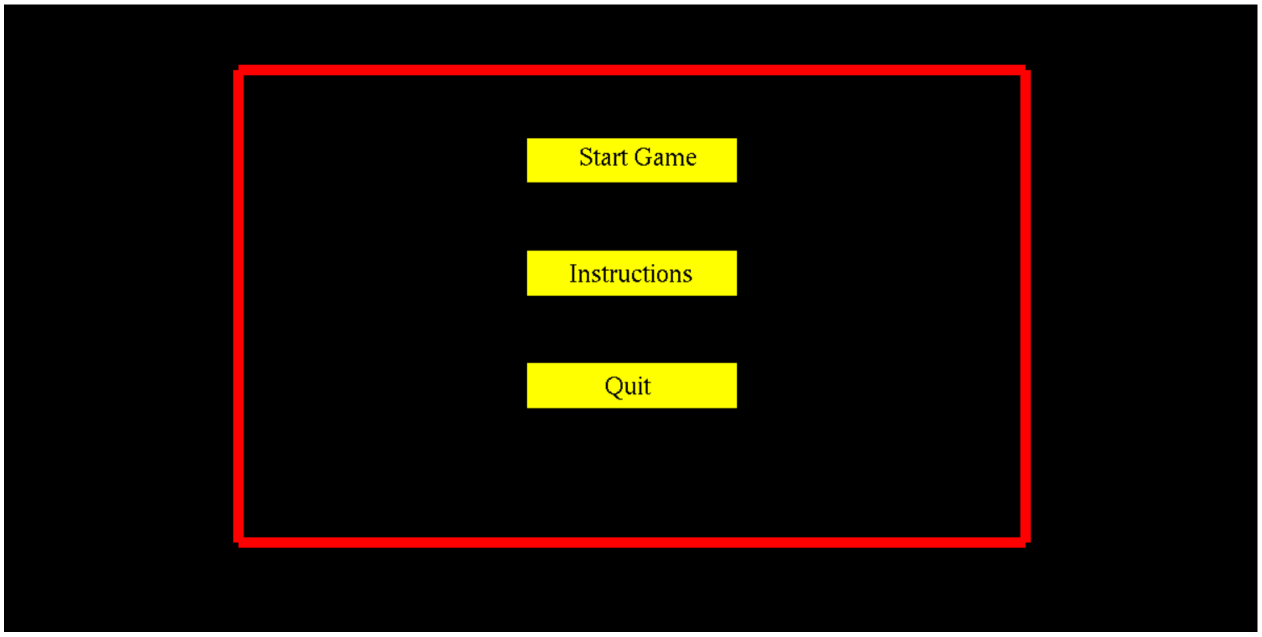


Fig 5.1 Menu interface of the space shooter game

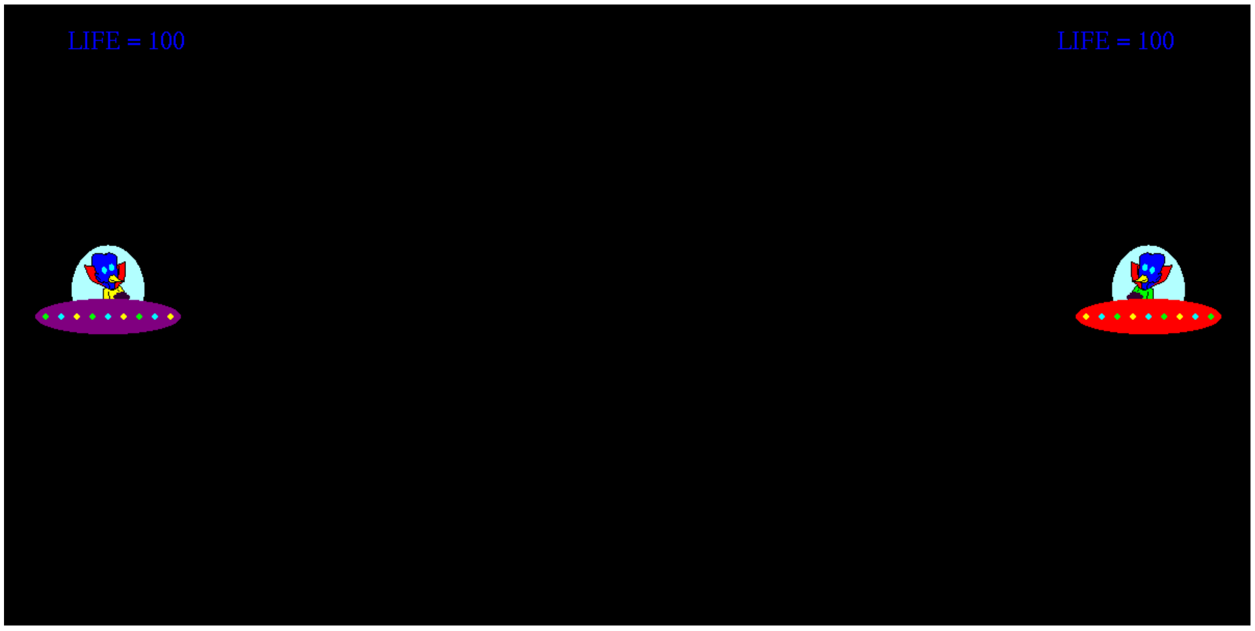


Fig 5.2 Initial Stage of the game

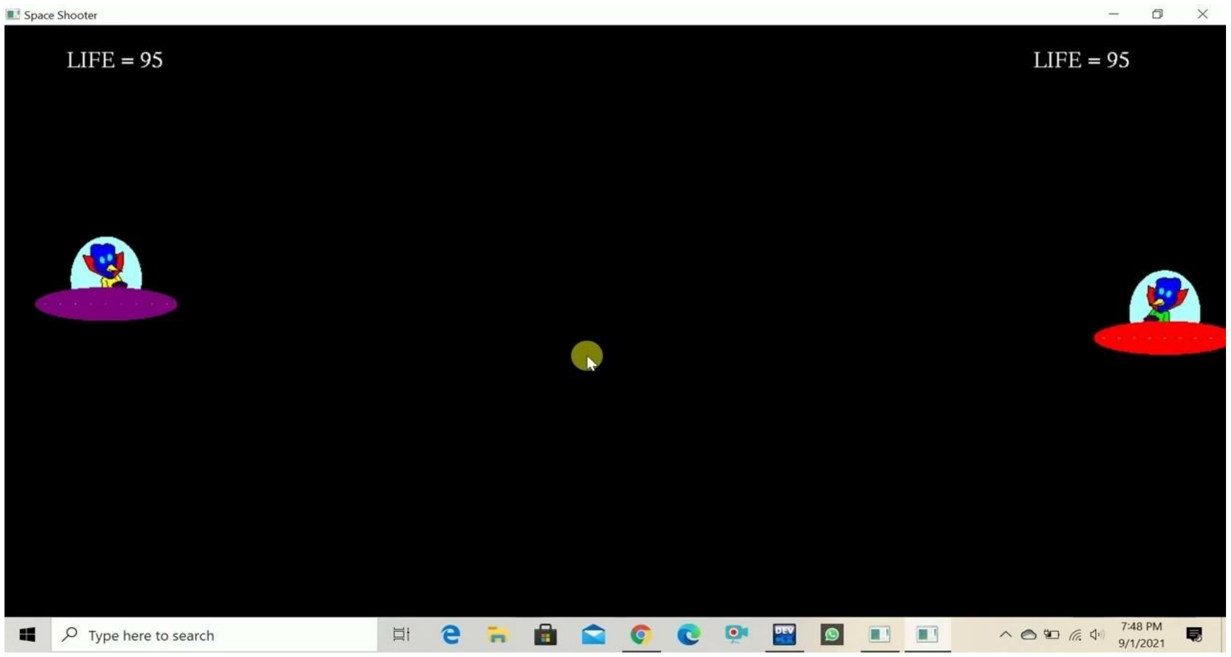


Fig 5.3 Initial phase of the game when life's are max

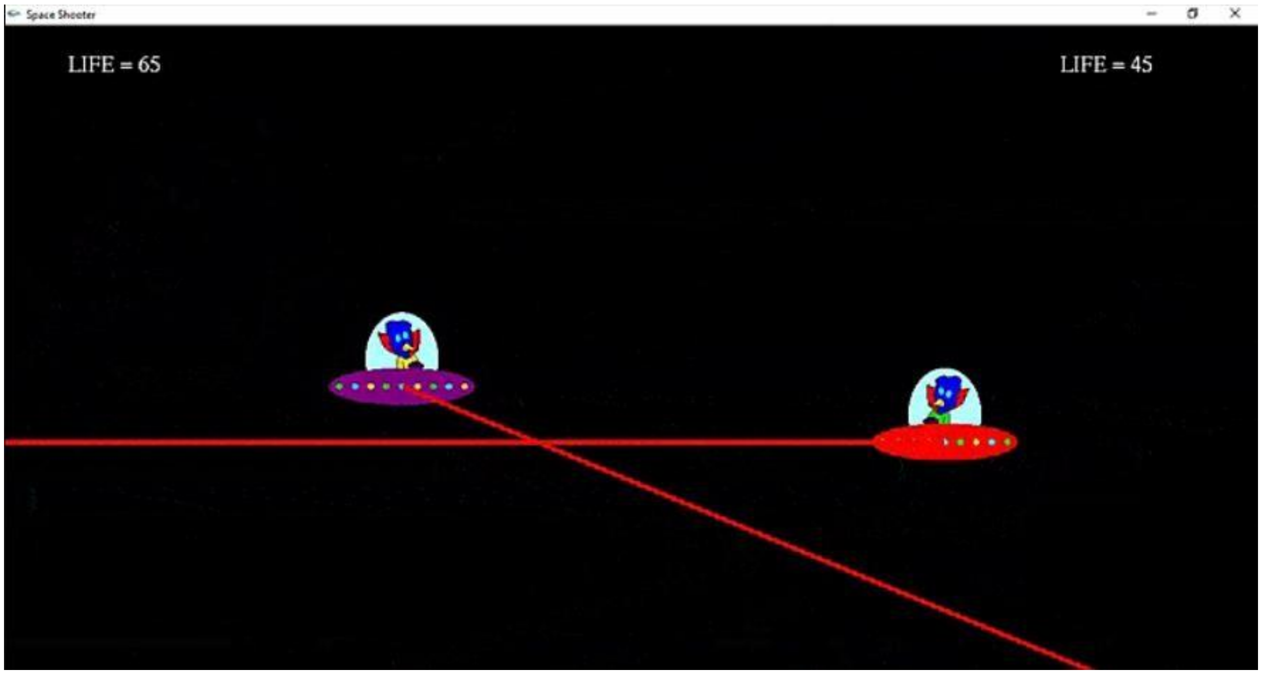
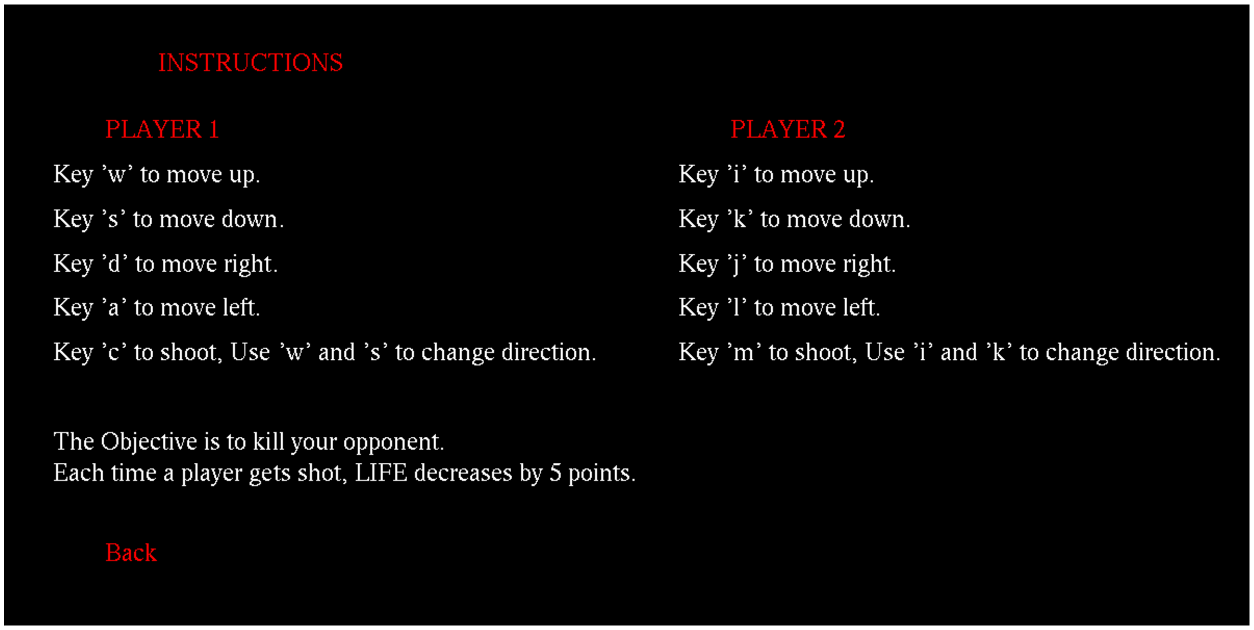


Fig 5.4 Displaying of gameplay



INSTRUCTIONS

PLAYER 1

Key ’w’ to move up.

Key ’s’ to move down.

Key ’d’ to move right.

Key ’a’ to move left.

Key ’c’ to shoot, Use ’w’ and ’s’ to change direction.

PLAYER 2

Key ’i’ to move up.

Key ’k’ to move down.

Key ’j’ to move right.

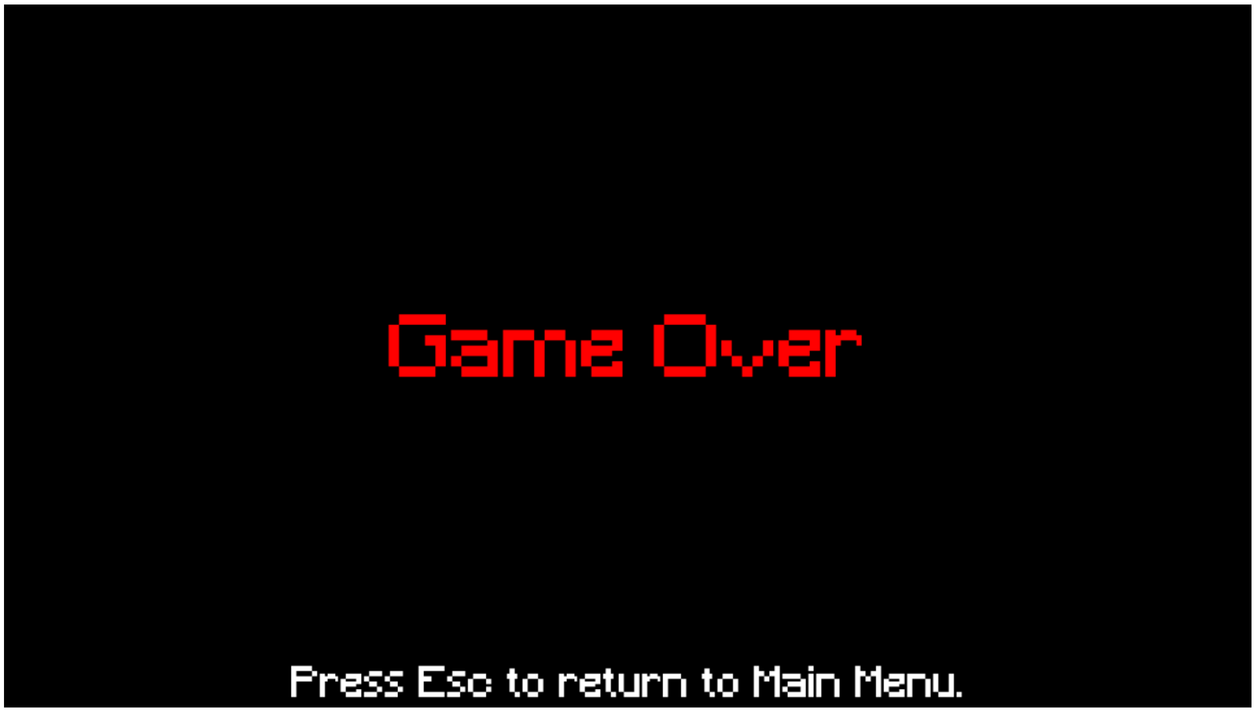
Key ’l’ to move left.

Key ’m’ to shoot, Use ’i’ and ’k’ to change direction.

The Objective is to kill your opponent.
Each time a player gets shot, LIFE decreases by 5 points.

Back

Fig 5.5 Instruction page for space shooter game



Game Over

Press Esc to return to Main Menu.

Fig 5.6 Game Over interface

CHAPTER 6

CONCLUSION

6.1 Summary

In a space shooter game, computer graphics (CG) play a pivotal role in creating an immersive and visually stimulating experience. The game's visuals often include detailed spacecraft designs, enemy ships, and vibrant explosions, which are essential for conveying the high-energy nature of space battles. Backgrounds are typically expansive, showcasing star-filled skies, distant planets, or other celestial phenomena that add depth and context to the game world. Animations bring movement to life, from the smooth gliding of player ships to the dynamic action of firing projectiles and dodging enemy attacks. Particle effects enhance the realism of explosions, laser blasts, and other visual impacts. Together, these graphical elements work harmoniously to captivate players, making the gameplay experience both visually engaging and enjoyable, while effectively communicating the game's fast-paced, high-stakes environment.

6.2 Future Enhancement

1. Advanced Graphics and Visual Effects

- Enhancement: Implement higher-resolution textures, detailed 3D models, and more sophisticated visual effects.
- Explanation: Upgrading graphics can make the game visually stunning, with realistic lighting, shadow effects, and enhanced particle systems that create a more immersive experience.

2. Procedural Generation

- Enhancement: Introduce procedural generation for levels, enemies, and environments.
- Explanation: Procedural generation can create varied and unpredictable gameplay experiences, extending replayability and keeping the game fresh with each playthrough.

3. Multiplayer and Co-op Modes

- Enhancement: Add online multiplayer or cooperative gameplay options.
- Explanation: Enabling players to compete or cooperate with others online can significantly enhance the social aspect of the game, offering new challenges and experiences.

6.3 References

[1]. John F. Hughes, Andries van Dam, Morgan McGuire, et al.- "Computer Graphics: Principles and Practice" - A comprehensive textbook on fundamental concepts in computer graphics.

[2]. Peter Shirley, Michael Ashikhmin, and Steve Marschner -"Fundamentals of Computer Graphics"-An accessible introduction to computer graphics principles and techniques.

[3]. A Top-Down Approach with WebGL" by Edward Angel and Dave Shreiner - "Interactive Computer Graphics”-Focuses on interactive graphics programming using WebGL.

[4]. Tomas Akenine-Möller, Eric Haines, and Naty Hoffman -"Real-Time Rendering"- Detailed exploration of techniques for real-time graphics rendering.

[5]. The Official Guide to Learning OpenGL" by Dave Shreiner, Graham Sellers, and John Kessenich - "OpenGL Programming Guide”-Official guide for learning OpenGL programming.

Online References:

[1]. https://github.com/topics/space-shooter?l=c%2B%2B

[2]. https://www.scribd.com/document/638252435/SPACE-SHOOTER-GAME

[3]. https://www.ijercse.com/specissue/march-2018/30.pdf

[4]. https://cplusplus.com/forum/beginner/281538/

[5].https://cgcookie.com/community/6578-unity-space-shooter-how-do-i-get-a-rigid-body-velocity-component