

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI**



Mini Project Report on

“SPACE SHOOTER CODE”

Submitted in the partial fulfillment for the requirements of Computer Graphics & Visualization Laboratory of
6th semester CSE requirement in the form of the Mini Project work

Submitted By

KUMAR ARYAN

USN: 1BY20CS091

NISHANT KUMAR SINHA

USN: 1BY20CS125

Under the guidance of

Mr. SHANKAR R
Assistant Professor, CSE, BMSIT&M

Prof. CHETHANA C
Assistant Professor, CSE, , BMSIT&M



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**BMS INSTITUTE OF TECHNOLOGY &
MANAGEMENT**

YELAHANKA, BENGALURU - 560064.

2022-2023

INSTITUTE VISION

To emerge as one of the finest technical institutions of higher learning, to develop engineering professionals who are technically competent, ethical and environment friendly for betterment of the society.

INSTITUTE MISSION

Accomplish stimulating learning environment through high quality academic instruction, innovation and industry-institute interface.

DEPARTMENT VISION

To develop technical professionals acquainted with recent trends and technologies of computer science to serve as valuable resource for the nation/society.

DEPARTMENT MISSION

Facilitating and exposing the students to various learning opportunities through dedicated academic teaching, guidance and monitoring.

PROGRAM EDUCATIONAL OBJECTIVES

1. Lead a successful career by designing, analyzing and solving various problems in the field of Computer Science & Engineering.
2. Pursue higher studies for enduring edification.
3. Exhibit professional and team building attitude along with effective communication.
4. Identify and provide solutions for sustainable environmental development.

ACKNOWLEDGEMENT

We are happy to present this project after completing it successfully. This project would not have been possible without the guidance, assistance and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each and every one who has helped us make this project a success.

We heartily thank our **Principal, Dr. MOHAN BABU G N, BMS Institute of Technology &Management**, for his constant encouragement and inspiration in taking up this project.

We heartily thank our **Professor and Head of the Department, Dr. THIPPESWAMY G, Department of Computer Science and Engineering, BMS Institute of Technology &Management**, for his constant encouragement and inspiration in taking up this project.

We gracefully thank our Project Guide, **Professor. Shankar R, Assistant Professor, Prof. Chethana C, Assistant Professor, CSE, Department of Computer Science and Engineering** for their intangible support and for being constant backbone for our project.

Special thanks to all the staff members of Computer Science Department for their help and kind co-operation.

Lastly, we thank our parents and friends for the support and encouragement given throughout in completing this precious work successfully.

KUMAR ARYAN(1BY20CS091)
NISHANT KUMAR SINHA(1BY20CS125)

ABSTRACT

The main aim of the Story Simulation Computer Graphics Mini Project is to illustrate the concepts and usage of pre-built functions in OpenGL. Simulation of "Space war story" is being done using computer graphics. The development of the game has large scope to learn computer graphics and visualization from scratch. We will be using OpenGL utility toolkit to implement the algorithm, written in C++ language.

Space shooters have been around for quite some time. The game has existed for well over half a century. History has it that space shooters became a thing in the days of classic arcade games where it gradually replaced Pinball machines in the 60s. Space war debuted in 1962, thereby marking the beginning of a period. Space shooters are some of the earliest video games ever developed. Steve Russell designed the code, which enabled the game to run on DEC PDP-1 minicomputers. The game failed to rank up in terms of reputation with other subsequent titles due to the limited availability of these compatible machines that facilitates its running.

In the 70s, Space Invaders took over the global market, raking in over \$600 million in Japan alone in its early days. It was available on arcade machines, which directly translates to it being open to a larger pool of players. Space shooting games left a mark on the young people of the 70s era, and to this day, Space Invaders is regarded as one of the best games ever developed.

The 80s was a significant era for space shooting games because this period saw many innovations brought to the genre. The popular game, Contra, from 1987, widened the array of weapons available to the player and the enemies encountered during gameplay. In that decade, space shooters invariably ranked among the top deliveries in the industry market.

The 2D perspective is a popular aspect of the space shooter design element. The layout of the space shooting game is almost always simple. Peculiar to this subgenre of the arcade game is the movement of the player across a sci-fi map. A significant part of the game is playable in an outer space setting.

We have tried to implement the project making it as user-friendly and error free as possible. We regret any errors that may have inadvertently crept in.

TABLE OF CONTENTS

1. ACKNOWLEDGEMENT

2. ABSTRACT

3. TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO
CHAPTER 1	INTRODUCTION	1
	1.1 Computer Graphics	1
	1.2 OPEN GL	2
	1.3 GLUT	2
CHAPTER 2	LITERATURE SURVEY	3
	2.1 Non-interactive graphics	4
	2.2 Interactive graphics	4
CHAPTER 3	PROJECT IN DETAIL	5
	3.1 Problem Statement	5
	3.2 Brief Introduction	5
	3.3 Scope and Motivation	6
	3.4 Proposed System	6
	3.5 User Interface	7
CHAPTER 4	REQUIREMENTS SPECIFICATION	8
	4.1 Hardware Requirements	8
	4.2 Software Requirements	8
CHAPTER 5	DESIGN AND IMPLEMENTATION	9
	5.1 Design	9
	5.1.1 Open GL Functions	9
	5.1.2 User Defined Functions	13
	5.2 Implementation	14-16
	CONCLUSION	17
	FUTURE ENHANCEMENTS	17
	REFERENCES	20

CHAPTER 1

INTRODUCTION**1.1 COMPUTER GRAPHICS**

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer hardware and software. The development of computer graphics, or simply referred to as CG, has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. 2D computer graphics are digital images—mostly from two-dimensional models, such as 2D geometric models, text (vector array), and 2D data. 3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering images.

The user controls the contents, structure, and appearance of the objects and of their displayed images by using input devices, such as keyboard, mouse, or touch-screen. Due to close relationships between the input devices and the display, the handling of such devices is included in the study of computer graphics. The advantages of the interactive graphics are many in number. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process data rapidly and efficiently. In many design, implementation, and construction processes today, the information pictures can give is virtually indispensable. Scientific visualization became an important field in the 1980s when the scientists and engineers realized that they could not interpret the prodigious quantities of data produced in supercomputer runs without summarizing the data and highlighting trends and phenomena in various kinds of graphical representations.

1.2 OPEN GL

OpenGL is the most extensively documented 3D graphics API (Application Program Interface) to date. Information regarding OpenGL is all over the Web and in print. It is impossible to exhaustively list all sources of OpenGL information. OpenGL programs are typically written in C and C++. One can also program OpenGL from Delphi (a Pascal-like language), Basic, Fortran, Ada, and other languages. To compile and link OpenGL programs, one will need OpenGL header files. To run OpenGL programs, one may need shared or dynamically loaded OpenGL libraries, or a vendor-specific OpenGL Installable Client Driver (ICD).

OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn).

1.3 GLUT

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres, and cylinders. GLUT even has some limited support for creating pop-up menus. The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross-platform) and to make learning OpenGL easier. All GLUT functions start with the glut prefix (for example, glutPostRedisplay marks the current window as needing to be redrawn).

CHAPTER 2

LITERATURE SURVEY

CG (Computer graphics) started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computer themselves. It includes the creation, storage, and manipulation of models and images of objects. These models include physical, mathematical, engineering, architectural, and even conceptual or abstract structures, natural phenomena, and so on. Computer Graphics today is largely interactive- the user controls the contents, structure, and appearance of objects and their displayed images by using input devices, such as keyboard, mouse or touch sensitive panel on the screen. Bitmap graphics is used for user-computer interaction. A Bitmap is an ones and zeros representation of points (pixels, short for picture elements ‘) on the screen. Bitmap graphics provide easy-to-use and inexpensive graphics based applications.

The concept of desktop is a popular metaphor for organizing screen space. By means of a window manager, the user can create, position, and resize rectangular screen areas, called windows, that acted as virtual graphics terminals, each running an application. This allowed users to switch among multiple activities just by pointing at the desired window, typically with the mouse. Graphics provides one of the most natural means of communicating with the computer, since our highly developed 2D and 3D pattern – recognition abilities allow us to perceive and process pictorial data rapidly and efficiently. In many design, implementation, and construction processes, the information pictures can give is virtually indispensable.

Computer graphics is the creation and manipulation of pictures with the aid of computers. It is divided into two broad classes

- Non-Interactive Graphics.
- Interactive Graphics.

2.1 NON – INTERACTIVE GRAPHICS

This is a type of graphics where observer has no control over the pictures produced on the screen. It is also called as Passive graphics.

2.2 INTERACTIVE GRAPHICS

This is the type of computer graphics in which the user can control the pictures produced. It involves two-way communication between user and computer. The computer upon receiving signal from the input device can modify the displayed picture appropriately. To the user it appears that the picture changes instantaneously in response to his commands. The following fig. shows the basic graphics system:

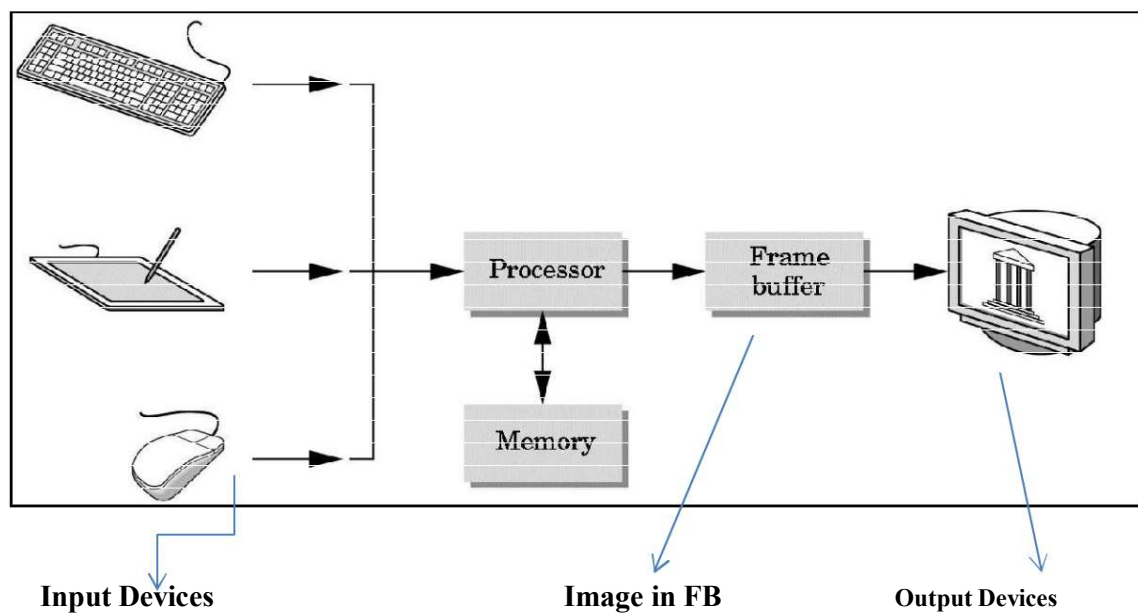


FIGURE 2.2 BASIC GRAPHICS SYSTEM

CHAPTER 3

PROJECT IN DETAIL**3.1 PROBLEM STATEMENT**

Computer graphics is no longer a rarity. It is an integral part of all computer user interfaces, and is indispensable for visualizing 2D; 3D and higher dimensional objects

. Creating 3D objects, rotations and any other manipulations are laborious process with graphics implementation using text editor. OpenGL provides more features for developing 3D objects with few lines by built in functions. The geometric objects are the building blocks of any individual. Thereby developing, manipulating, applying any transformation, rotation, scaling on them is the major task of any image development.

So in this project the main problem statement is to create a space shooter game with help of computer graphics that is using Open GL. We will be having two alien spaceships on a sci-fi map who will be fighting against each other. It is a multi-player game.

3.2 BRIEF INTRODUCTION

In this Project, we are making two graphically designed alien spaceships fight against each other in a sci-fi map. We make use of C++ in OpenGL in computer graphics.

The 2D perspective is a popular aspect of the space shooter design element. The layout of the space shooting game is almost always simple. Peculiar to this subgenre of the arcade game is the movement of the player across a sci-fi map. A significant part of the game is playable in an outer space setting.

This is a two player game. Here both the spaceships have 100 life's and a damage of 5 is given to the player who gets hit by the other player.

3.3 SCOPE AND MOTIVATION

The existence of a continually moving market orchestrates the need for different game genres to change with time. No single game genre stays the same forever. Space shooters don't work differently from the vast majority of games in the industry. As time passes, space shooters evolve with the market in different ways. Development of subdivisions within the arcade game genre, taking various appearances, and merging styles of play peculiar to other games are some of the aspects where these changes are evident.

Space shooters have been different over the years. The earliest model had static players where the movement was highly limited. Rail shooters are ones where the player stays on a static path. Others put the player in a fictitious canal. Some make use of the 3D engine to allow for movement in multiple directions.

In this project we are trying to develop an easy, efficient and cost-free method of space shooter game.

3.4 PROPOSED SYSTEM

We need something which is as interesting as a TV serial and at the same time filled with good stuffs.

Space shooters are somewhat different from a typical action game. Its conventional setting is often one in a sci-fi environment. The player usually appears like a spacecraft or something close to it. Players need no skill set or strategy to advance in the game. The levels surpassed or points amassed are the basis for judgment of players' progress.

With the use of different functionality like of making polygons, using various functions of mouse and creation of menu, the whole sci-fi map with the two players as alien spaceships. This project gives a user-friendly interface for the end user to interact with the frames. This project contains several different frames that are used in a particular sequence to help the user understand the story.

3.5 USER INTERFACE

The legend for user interaction is as follows:

For Player 1:

“W”	To move up.
“S”	To move down.
“D”	To move right.
“A”	To move left.
“C”	To shoot and “w” and “s” to change direction.

For Player 2:

“I”	To move up.
“K”	To move down.
“J”	To move right.
“L”	To move left.
“M”	To shoot and “i” and “k” to change direction.

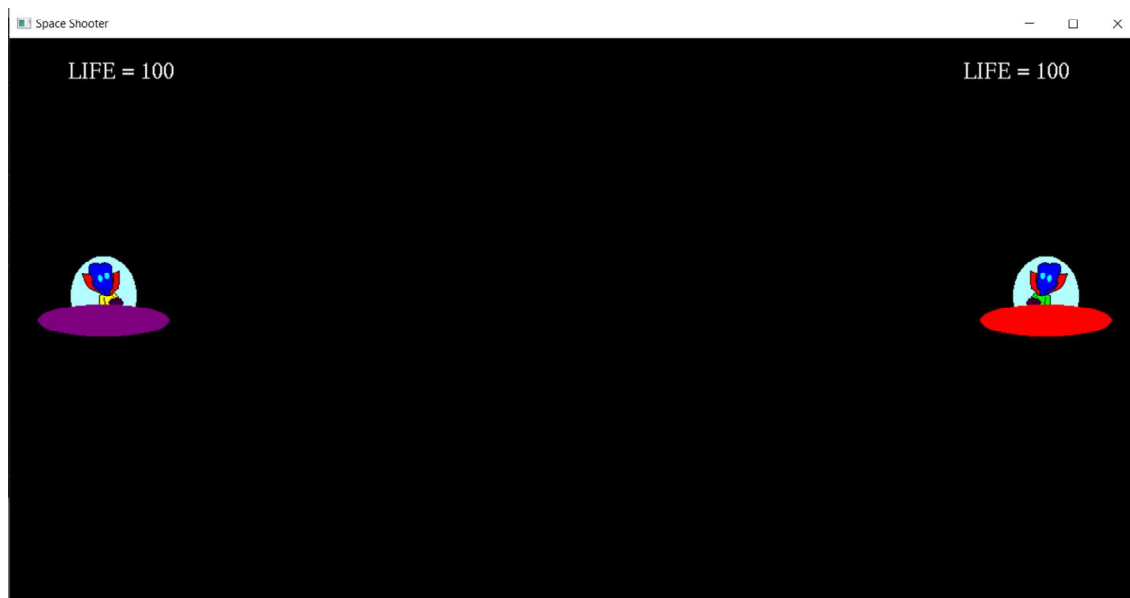


FIGURE 3.5.1: SPACE SHOOTER (A scenario)

CHAPTER 4**REQUIREMENT SPECIFICATIONN****4.1 HARDWARE REQUIREMENTS**

Processor: INTEL / AMD

Main memory: 2 GB RAM (Min.) Hard

Disk: Built-in is sufficient Keyboard:

QWERTY Keyboard Mouse: 2 or 3

Button mouse

Monitor: 1024 x 7668 display resolution

4.2 SOFTWARE REEQUIREMENTS

Programming language – C/C++ using OpenGL

Operating system – Windows/Linux

Compiler – C/C++ Compiler (GCC compiler)

IDE – Code blocks

Functional Requirement – <GL/glut.h>

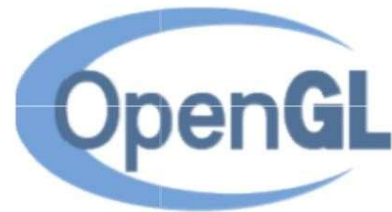


FIGURE 4.2.1: OpenGL API

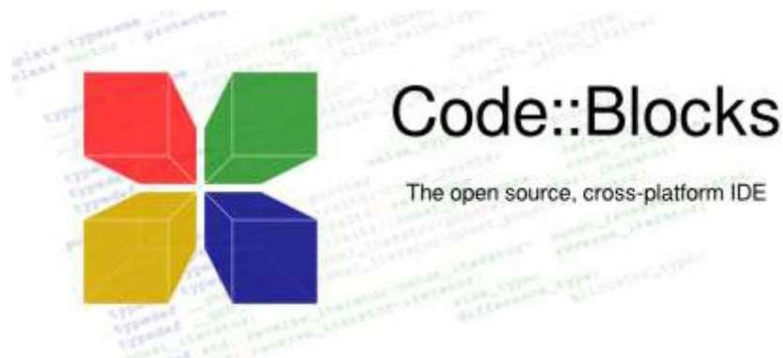


FIGURE 4.2.2: The Code Blocks IDE

CHAPTER 5**DESIGN AND IMPLEMENTATION****5.1 DESIGN**

The “Hare and Tortoise SPACE SHOOTER” is designed using some of the OpenGL inbuilt functions along with some user defined functions. So, this section comprises of the complete explanation of the design of the project using various opengl functions and user defined methods along with its explanation. This chapter is divided into two sections –

OPENGL Functions – This section throws light on the various OpenGL functions

- used and its uses.

USER – DEFINED Functions – This section demonstrates the various user defined

- functions and its purpose.

5.1.1 OPEN GL FUNCTIONS:

The different OpenGL functions that are used in the project is described below:

`glClearColor(0.1,0.8,0.1,1.0)` :- `glClearColor()` specifies the red, green, blue, and alpha values used by `glClear()` to clear the color buffers. Values specified by `glClearColor()` are clamped to the range 0, 1 `glMatrixMode(GL_MODELVIEW)` :- specify which matrix is the current matrix.

Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: `GL_MODELVIEW`, `GL_PROJECTION`, and `GL_TEXTURE`. The initial value is `GL_MODELVIEW`.

`gluOrtho2D(0,1000,0,500)` :- define a 2D orthographic projection matrix. left, right-

Specify the coordinates for the left and right vertical clipping planes.

bottom, top - Specify the coordinates for the bottom and top horizontal clipping planes.

`glRasterPos2f(x, y)` :- The `glRasterPos2` function uses the argument values for `x` and `y` while implicitly setting `z` and `w` to zero and one. The object coordinates presented by `glRasterPos` are treated just like those of a `glVertex` command. They are transformed by the current model view and projection matrices and passed to the clipping stage.

`glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,string[i])` :-
`glutBitmapCharacter` renders a bitmap character using OpenGL. Without using any display lists, `glutBitmapCharacter` renders the character in the named bitmap font.

`glutFullScreen()` :- `glutFullScreen` requests that the current window be made full screen. The exact semantics of what full screen means may vary by window system. The intent is to make the window as large as possible and disable any window decorations or borders added the window system.

`glFlush()` :- Force execution of GL commands in finite time. `glFlush` empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

`glPointSize(4)` :- specify the diameter of rasterized points. `glPointSize` specifies the rasterized diameter of points. The value written to the shading language built-in variable `gl_PointSize` will be used.

`glBegin(GL_POINTS)` :- delimit the vertices of a primitive or a group of like primitives. `glBegin` and `glEnd` delimit the vertices that define a primitive or a group of like primitives. `glBegin` accepts a single argument that specifies in which of ten ways the vertices are interpreted.

`glVertex2i(278,273)` :- specify a vertex. `glVertex` commands are used within `glBegin/glEnd` pairs to specify point, line, and polygon vertices. The current color, normal, texture coordinates, and fog coordinate are associated with the vertex when `glVertex` is called.

`glEnd()` :- delimit the vertices of a primitive or a group of like primitives. `glBegin` and `glEnd` delimit the vertices that define a primitive or a group of like primitives.

`glutSwapBuffers()` :- `glutSwapBuffers` swaps the buffers of the current window if double buffered. Performs a buffer swap on the layer in use for the current window. Specifically, `glutSwapBuffers` promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. An implicit `glFlush` is done by `glutSwapBuffers` before it returns. If the layer in use is not double buffered, `glutSwapBuffers` has no effect.

`glPushMatrix()` push and pop the current matrix stack. There is a stack of

matrices for each of the matrix modes. The current matrix in any mode is the matrix on the top of the stack for that mode. `glPushMatrix` pushes the current matrix stack down by one, duplicating the current matrix. That is, after a `glPushMatrix` call, the matrix on top of the stack is identical to the one below it.

`glPopMatrix()` :- push and pop the current matrix stack. `glPopMatrix` pops the current matrix stack, replacing the current matrix with the one below it on the stack.

`glTranslate(a2,0,0)` :- multiply the current matrix by a translation matrix. `glTranslate` produces a translation by x y z . The current matrix is multiplied by this translation matrix, with the product replacing the current matrix.

`glutInit(&argc,argv)` :- `glutInit` is used to initialize the GLUT library. `glutInit` will initialize the GLUT library and negotiate a session with the window system. During this process, `glutInit` may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. `glutInit` also processes command line options, but the specific options parse are window system dependent.

`glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB)` :- sets the initial display mode. The initial display mode is used when creating top-level windows, sub windows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

`glutInitWindowSize(1000,500)` :- set the initial window size . Windows created by `glutCreateWindow` will be requested to be created with the current initial window position and size. The initial value of the initial window size GLUT state is 300 by 300. The initial window size components must be greater than zero. The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. A GLUT program should use the windows reshape callback to determine the true size of the window. `glutCreateWindow("HARE AND TORTOISE")` :- creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name. Implicitly, the current window is set to the newly created window.

`glutPositionWindow(50,50)` :- requests a change to the position of the current window. . For top-level windows, the x and y parameters are pixel offsets from the screen origin. For subwindows, the x and y parameters are pixel offsets from the window's parent window origin.

`glutDisplayFunc(display1)` :- registers the callback function (or event handler) for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window re-paint request. In the example, we register the function `display()` as the handler.

`glutKeyboardFunc(NormalKey)` :- `glutKeyboardFunc` sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. `glutMainLoop()` :- `glutMainLoop` enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

5.1.2 USER DEFINED FUNCTIONS:

The whole project is divided into many small parts known as functions and these functions would take care of implementing particular parts of this project which makes the programming easy and effective. Each function takes the responsibility of designing the assigned parts hence we combined all the functions at a particular stage to get the final required design. The functions that are used to implement the “Space Shooters” are:

void gameScreenDisplay() :- This function is used to display the start of the game scene . It calls other function like DisplayHealthBar1(), DisplayHealthBar1() . It is also used to display the health bar of 1st player. In the end there is a gllush() to display everything in the screen.

void DisplayHealthBar2() :- This function is used to display the health bar of 2nd player. In the end there is a glflush() to display everything in the screen.

void checkLaserContact() :- This function is used to check the intersection point between the laser shot from one player to the position of another .

void displayGameOverMessage() :- This function is used to show the end of the game and display a Game Over message.

void keyOperations() :- It is the function used for player movement and to shoot lasers. void mouseClicked() :- Mouse click function allows the use of mouse . The starting menu is operated using mouse.

void passiveMotionFunc() :- v

5.2 IMPLEMENTATION

This section speaks about the implementation of the project via the snapshots. It gives the detailed description of the way in which various scenes and frames are implemented.



FIGURE 5.2.1

First Screen – This is the first scene when the project runs. It displays the welcome page and basic key-action instructions of this project.

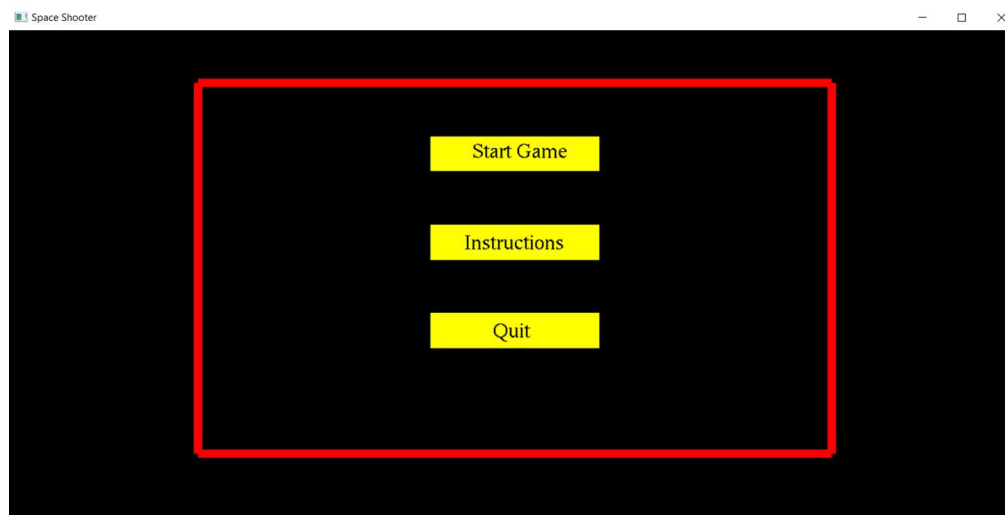
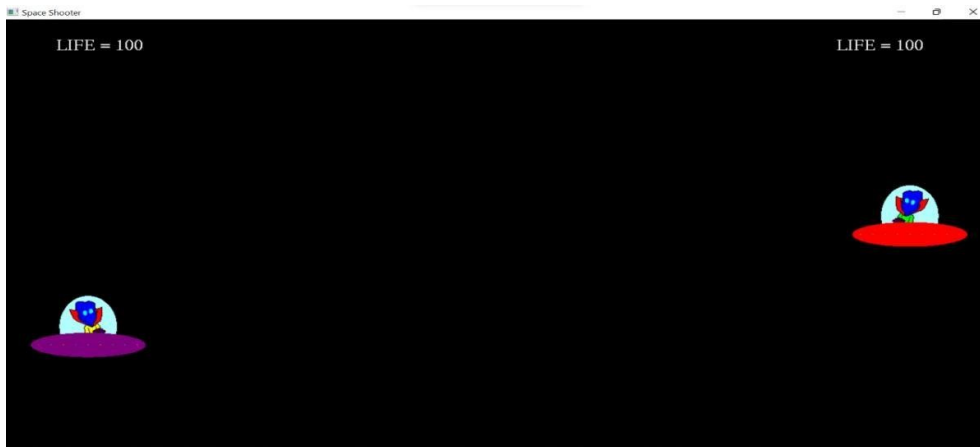
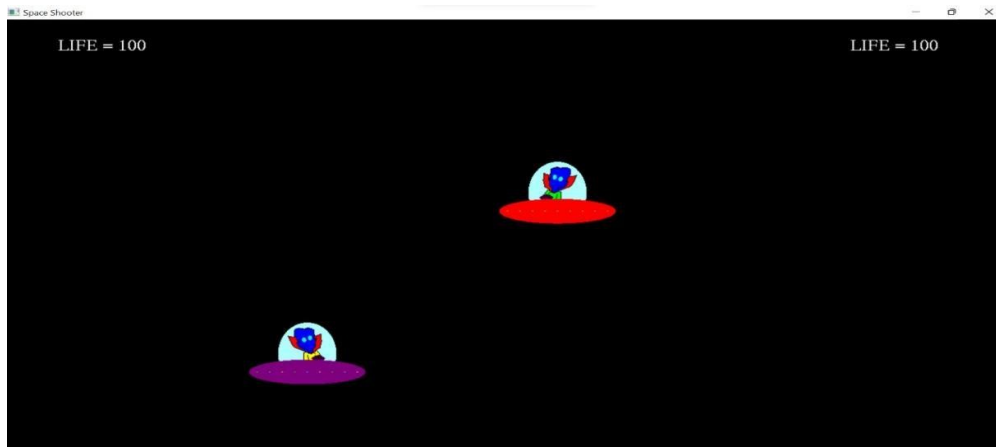


FIGURE 5.2.2

Second Screen – This is the menu, where we can either start the game or see the instructions to play the game or exit the game/program.

**FIGURE 5.2.3**

Third Screen – This is the third scene of the project. In this scene, both the players start the game at initial position.

**FIGURE 5.2.4**

This is the fourth scene of the project. In this scene players can move throughout the window and adjust their position.

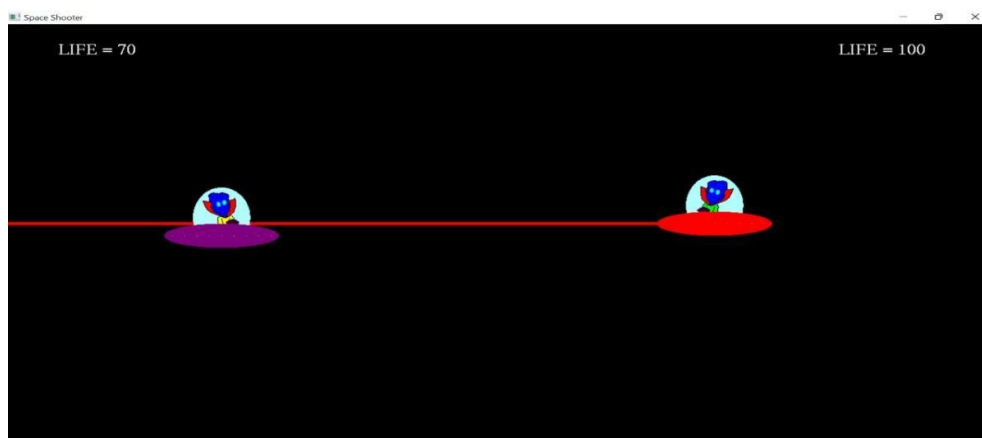


FIGURE 5.2.5

This is the fifth scene of the project. In this scene players can shoot each other. By hitting the other player with laser beam that player can decrease other player's life. When any of the player's life comes down to 0, other player gains victory.

**FIGURE 5.2.6**

This is the sixth scene of the project. This scene shows when gaining victory, it displays "Game OVER" and declares the winner.

CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

The very purpose of developing this project is to exploit the strength of OpenGL graphics capabilities for an interesting cause. The SPACE SHOOTER has been tested under Windows 10, and has been found to provide ease of use and manipulation to the user. The SPACE SHOOTER created for the Windows operating system can be used to play multiplayer game. It has a very simple and effective user interface.

We found designing and developing this project as a very interesting and learning experience. It helped us to learn about computer graphics, design of Graphical User Interfaces, interface to the user, user interaction handling and screen management. The graphics editor provides all and more than the features that have been detailed in the university syllabus.

FUTURE ENHANCEMENTS

Some of the future enhancements would be:

- Support for advanced 3D representation of the entire scenario.
- Support for transparency of layers and originality that is, simulating the objects in a more realistic way.
- Making the user interface of this project more user friendly which will
- certainly be more effectively and efficiently narrated.

REFERENCES

- [1] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version, 4th Edition, Pearson Education, 2011.
- [2] Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5th edition. Pearson Education, 2008.
- [3] Jackie.L.Neider,Mark Warhol,Tom.R.Davis,"OpenGL Red Book",Second Revised Edition,2005.
- [4] www.opengl.org
- [5] <https://learnopengl.com/>
- [6] [Welcome to My Blog \(shankarrajabopal.github.io\)](https://shankarrajabopal.github.io)