RV COLLEGE OF ENGINEERING®
BENGALURU – 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

"Shooting Game"

COMPUTER GRAPHICS LAB (16CS73)

OPEN ENDED EXPERIMENT REPORT

VII SEMESTER

2020-2021

Submitted by

RITESH M	1RV17CS124
SAADHVI RAYASAM	1RV17CS131

Under the Guidance of

Mamatha T Jayaprakash
Department of CSE, R.V.C.E.,
Bengaluru - 560059

RV COLLEGE OF ENGINEERING®, BENGALURU - 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the **Open-Ended Experiment** titled " Shooting Game " has been carried out by Ritesh M (1RV17CS124) and Saadhvi Rayasam (1RV17CS131), bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73)** during the year 2020-2021. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report.

Mamatha T Jayaprakash
Faculty Incharge,
Department of CSE,
R.V.C.E., Bengaluru –59

Dr. Ramakanth Kumar P
Head of Department,
Department of CSE,
R.V.C.E., Bengaluru–59

RV COLLEGE OF ENGINEERING® , BENGALURU - 560059
(Autonomous Institution Affiliated to VTU)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **Ritesh M (1RV17CS124) and Saadhvi Rayasam (1RV17CS131),** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled **"Shooting Game"** has been carried out by us and submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73) - Open-Ended Experiment** during the year 2020-2021. We do declare that the matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

Place: Bengaluru **Ritesh M**

Date: 18 December 2020 **Saadhvi Rayasam**

Table of Contents

1. Introduction	5
1.1 Computer Graphics	5
1.2 OpenGL	5
1.2.1 OpenGL Graphics Architecture	5
1.2.2 Primitives and Attributes	6
1.2.3 Color, Viewing and Control Functions	7
1.3 Proposed System	9
1.3.1 Objective of the project	9
1.3.2 Methodology	9
1.3.3 Scope	10
2. Requirement Specifications	11
2.1 Hardware Requirements	11
2.2 Software Requirements	11
3. System Design and Implementation	12
3.1 Data Flow Diagram	12
3.2 Structure Chart	13
3.3 Modular Description	14
4. Results and Snapshots	16
5. Conclusion	18
6. Bibliography	19
7. Appendix	20

1. Introduction

This section gives a brief introduction on Computer Graphics and the Open source graphics library OpenGL along with its architecture. An overview of the fundamentals of OpenGL such as its primitives, attributes, color, control and viewing functions has been illustrated. A clear understanding of these basic concepts is essential to fully grasp the project.

1.1 Computer Graphics

Computer graphics studies the manipulation of visual and geometric information using computational techniques. It focuses on the mathematical and computational foundations of image generation and processing rather than purely aesthetic issues. Computer graphics is often differentiated from the field of visualization, although the two fields have many similarities.

1.2 OpenGL

OpenGL (Open Graphics Library) is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit, to achieve hardware-accelerated rendering.

1.2.1 OpenGL Graphics Architecture

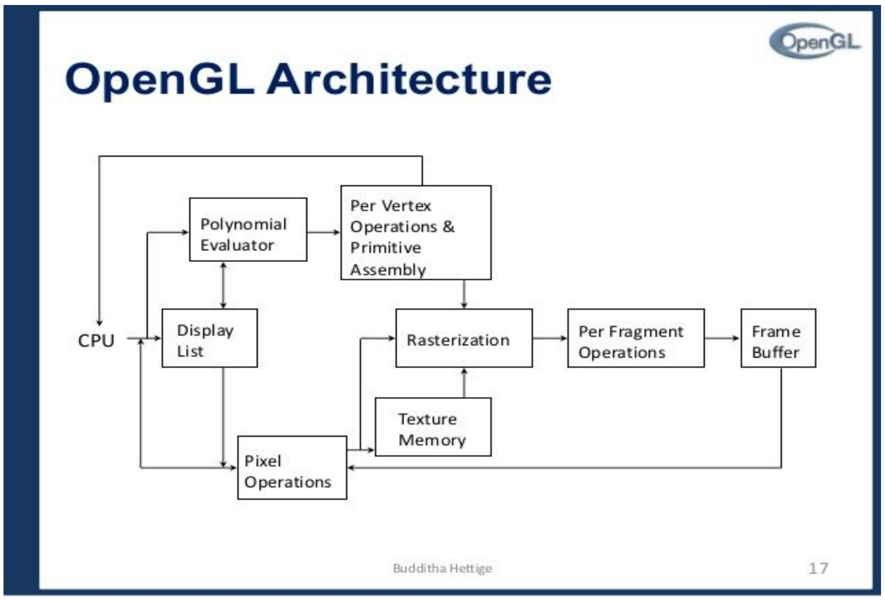


Fig1: OpenGL Architecture

The OpenGL architecture is structured as a state-based pipeline. Below is a simplified diagram of this pipeline. Commands enter the pipeline from the left.Commands may either be accumulated in display lists, or processed immediately through the pipeline. Display lists allow for greater optimization and command reuse, but not all commands can be put in display lists.

The first stage in the pipeline is the evaluator. This stage effectively takes any polynomial evaluator commands and evaluates them into their corresponding vertex and attribute commands.

The second stage is the per-vertex operations, including transformations, lighting, primitive assembly, clipping, projection, and viewport mapping.

The third stage is rasterization. This stage produces fragments, which are a series of framebuffer addresses and values, from the viewport-mapped primitives as well as bitmaps and pixel rectangles.

The fourth stage is the per-fragment operations. Before fragments go to the framebuffer, they may be subjected to a series of conditional tests and modifications, such as blending or z-buffering.

Parts of the framebuffer may be fed back into the pipeline as pixel rectangles. Texture memory may be used in the rasterization process when texture mapping is enabled.

1.2.1 Primitives and Attributes

OpenGL supports several basic primitive types, including points, lines, quadrilaterals, and general polygons. All of these primitives are specified using a sequence of vertices. The diagram below shows the basic primitive types, where the numbers indicate the order in which the vertices have been specified. All geometric primitives are eventually described in terms of their vertices - coordinates that define the points themselves, the endpoints of line segments, or the corners of polygons.

An attribute is any property that determines how a geometric primitive is to be rendered. Each geometric primitive has a set of attributes . Point has Color, line segments have color, thickness, and pattern and polygon has pattern.

1.2.3 Color, Viewing and Control Functions

Coloring, Viewing and Control Functions form an integral part of OpenGL. Coloring functions give us the tools necessary to set the background color, color the primitives and diagrams. Viewing helps us to change the point of view in which we see the output of an OpenGL program. Control Functions gives the coder options to react to a particular input, like the click of a mouse or on pressing a key.

i. Color

There are many ways to specify a color in computer graphics, but one of the simplest and most widely used methods of describing a color is the RGB color model. RGB stands for the colors red, green and blue: the additive primary colors. Each of these colors is given a value, in OpenGL usually a value between 0 and 1. 1 means as much of that color as possible, and 0 means none of that color. We can mix these three colors together to give us a complete range of colors, as shown to on the left.

For instance, pure red is represented as (1, 0, 0) and full blue is (0, 0, 1). White is the combination of all three, denoted (1, 1, 1), while black is the absence of all three, (0, 0, 0). Yellow is the combination of red and green, as in (1, 1, 0). Orange is yellow with slightly less green, represented as (1, 0.5, 0).

ii. Viewing

As far as OpenGL is concerned, there is no camera. More specifically, the camera is always located at the eye space coordinate (0.0, 0.0, 0.0). To give the appearance of moving the camera, your OpenGL application must move the scene with the inverse of the camera transformation by placing it on the MODELVIEW matrix. This is commonly referred to as the viewing transformation.

In practice this is mathematically equivalent to a camera transformation but more efficient because model transformations and camera transformations are concatenated to a single matrix. As a result though, certain operations must be performed when the camera and only the camera is on the MODELVIEW matrix. For example to position a light source in world space it must be positioned while the viewing transformation and only the viewing transformation is applied to the MODELVIEW matrix.

iii. Control Functions

Control functions are used to communicate with the window system, to initialize programs, deal with execution errors etc. Some of these functions are listed below.

Interaction between windowing system and OpenGL is initialized by the GLUT function glutInit(int *argcp, char **argv). The two arguments allow the user to pass command-line arguments, as in the standard C main function.

Opening an OpenGL window can be accomplished by glutCreateWindow(char *title) where the string title is displayed on the top of the window. The window has a default size, a position on the screen, and characteristics such as use of RGB color. GLUT functions allow specification of these parameters.

Another function is glutInitDisplayMode(GLUT RGB | GLUT DEPTH | GLUT DOUBLE). Here, RGB color (default) instead of indexed (GLUT INDEX) color Depth buffer (not default) for hidden-surface removal double buffering rather than the default single (GLUT SINGLE) buffering.

1.3 Proposed System

The PC gaming industry alone is worth $37 billion USD worldwide. Some make a great career as gamers while others get addicted and go down a bad lane. While there are some pros and cons, focussing on the positives here, certain games are said to increase memory, improve dexterity and some games help dyslexic kids in reading. This project particularly focuses on improving dexterity, wise decision making. This project involves creating a shooting game where a player can shoot bullets at the press of a button at bots that are carrying the score they'd get if the user pops that particular bot. It is a low graphics game whose target users are, for the most part, kids and kids with special needs.

1.3.1 Objective of the project

The objective of this project is to create a shooting game using OpenGL library functions. The project should be pleasing to the eyes, it should also have a smooth interface that's also easy to use. The functions should work as specified in the instructions without latency.

1.3.2 Methodology

The sequence for the game is as follows :

- Welcome screen with a sign is briefly visible, followed by a menu for proceeding
- Menu for the game has options to enter the game, view game credits and to exit
- The view credits screen displays our names and USNs.

On entering the game :

- A window displays a message stating that the game is being loaded.
- Once the shooting game is loaded, the user observes a gun that is placed at its default position (at a height from the bottom of the screen and to the extreme left, that is, the muzzle of the gun points to the right) and the gun can be adjusted to move up or down throught the game as needed. The timer and number of bullets is initialized and bots start appearing on the opposite end, each with upward movement.
- Bullets can be fired from the gun on clicking the spacebar button on the keyboard, and the timer begins on the first click.

- Bots (rectangular bodied figures with a face) are the target objects in the shooting game. Each carries a random number on the body indicating its score which can be secured on successfully firing a bullet at it. The bots appear at certain random heights on the screen and continuously move upwards from the initial position.
- There can be 5 bots at any time instance and bullets must be fired from the gun in order to hit the bot(s). A headshot earns more points than a body shot.
- A total of 30 bullets are available and the total game time is 20 seconds. If either of the conditions are exhausted, that is, if 30 bullets are used, or the timer exceeds 20 seconds from the start of the game, the game terminates.
- Once the game ends, a score card is displayed with information such as the total score obtained, number of bots hit, number of headshots and the percentage of accuracy with respect to the number of bullets fired.

1.3.3 Scope

The shooting game is graphically attractive and smooth without latency, which can be utilised to train kids and kids with special needs in cognition and dexterity. Some advantages of playing this game include improvement in hand-eye coordination, and an augmented attention span/concentration level, along with enhanced decision making skill. This project gives a digital version of the shooting game famously played in arcades. The game mimics the real arcade shooting game by making it visually vibrant and also allowing the user to adjust the position of the gun as desired throughout the duration of the game. The game is solely built using OpenGL, and hence, it can be thought of as a beginner's guide to developing games for the gaming community and for kids or adults by and large.

2. Requirement Specifications

Here are some hardware and software requirements that should be met by the machine that the project will be run on. The requirements listed are related to the hardware that is expected to be present on the system that it is played on.

2.1 Hardware Requirements

- 4GB/8GB RAM
- 256 GB SSD/512 GB HDD
- 1536x864 Screen resolution

2.2 Software Requirements

- OpenGL Graphics Library
- Visual Studio
- Windows 10 Operating System

3. System Design and Implementation

System design is the process of designing the elements of a system such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system. Here in this section the architecture and information flow of the project is discussed and illustrated thoroughly.

3.1 Data Flow Diagram

The complete design of the proposed system is designed in terms of data flow diagrams.

A Data Flow Diagram (DFD) is a graphical representation of the 'flow' of data through a system. This is used to model process aspects and gives a preliminary view for the data. Additionally, using this method makes it easier to bridge the gap between the user and the developer. It converts a descriptive design of the system to an implementable sequence of events.
Depending on the methodology (Gane and Sarson vs. Yourdon and Coad), DFD symbols vary slightly. However, the basic ideas remain the same. There are four basic elements of a data flow diagram: processes (transforms incoming data flow into outgoing data flow), data stores (repositories of data in the system), external entities (objects outside the system, with which the system communicates; sources and destinations of the system's inputs and outputs) and data flows (pipelines through which packets of
Data flow diagrams are also categorized by level. Fig 2 shows the final DFD.

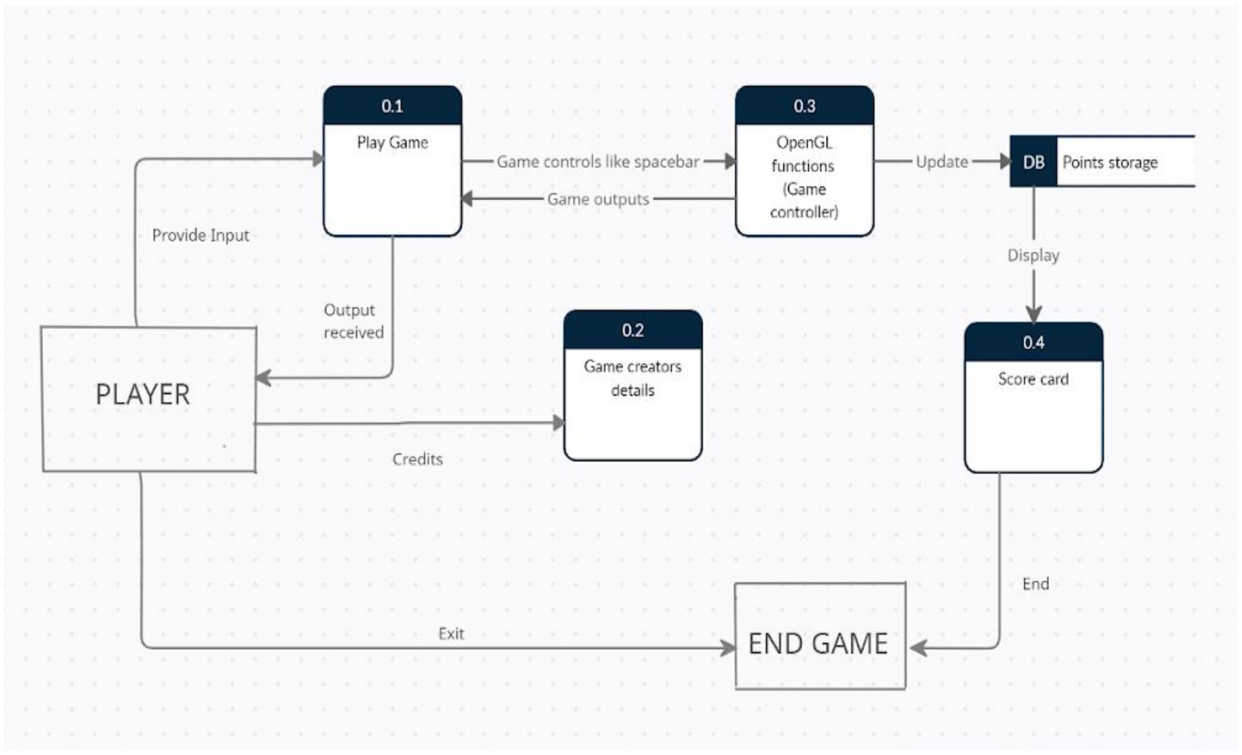


Fig2 : Data Flow Diagram

3.2 Structure Chart

A structure chart (SC) in software engineering and organizational theory is a chart which shows the breakdown of a system to its lowest manageable levels. They are used in structured programming to arrange program modules into a tree. Each module is represented by a box, which contains the module's name. The tree structure visualizes the relationships between modules.

In structured analysis structure charts, according to Wolber (2009), "are used to specify the high-level design, or architecture, of a computer program. As a design tool, they aid the programmer in dividing and conquering a large software problem, that is, recursively breaking a problem down into parts that are small enough to be understood by a human brain. The process is called top-down design, or functional decomposition. The structure of the project is illustrated in the figure below (Fig3).

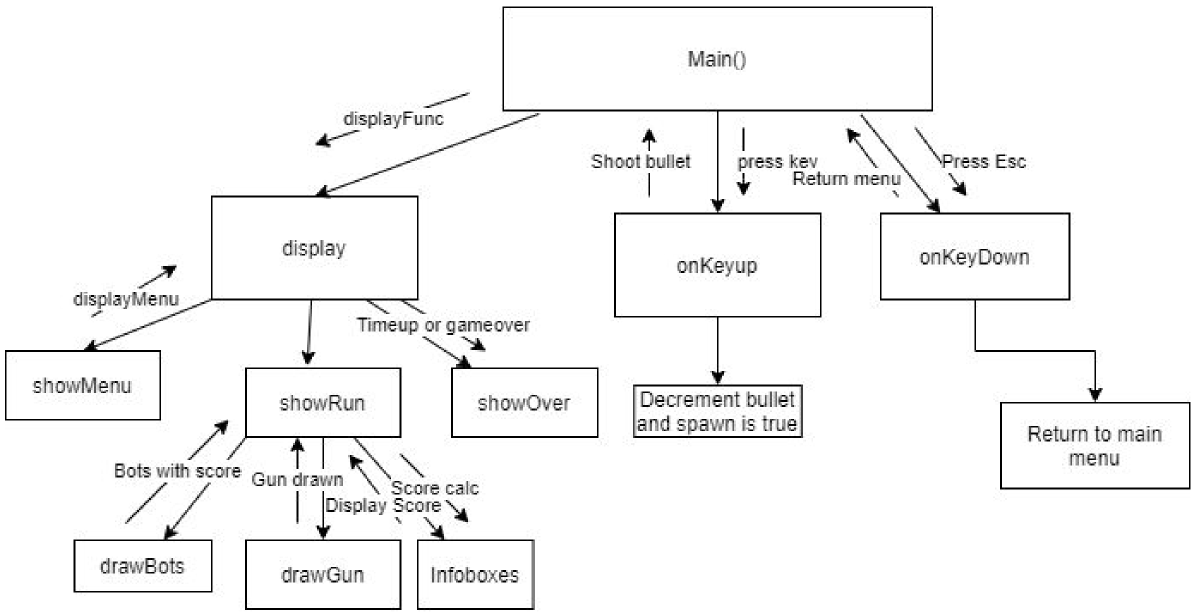


Fig3: Structure Chart

3.3 Modular Description

In this section the main modules are listed and explained in more detail. The modules are described and their specific functions are highlighted. The project is divided into various parts for modularity and control, this way the project can be altered with ease as it would be clear which module handles a specific task.

1. **Displaying menu**

A concise menu is displayed at the center of the window with a few options for a player to choose from. The player can enter and start the game, or choose to look at the credits of the game developers, or exit from the game. The bottom of the screen has a help tip to aid in the selection of each of the menu options.

2. **Drawing gun**

This module is used to draw the shooting gun, the "AK47". The gun is drawn with the help of GL linestrip and quadrilaterals. The gun can be moved up or down, and positioned as per convenience to shoot the moving bots and secure more points.

3. Drawing bots

This module is responsible for drawing the bots. It should release a random number of bots at an instance with the upper limit being 5 bots per instance. It should also assign colors to each bots and points should be allotted to each bot at random and their positioning relative to each other should be changing as well to disrupt any regular pattern that may arise during the gameplay with respect to the bots being released from the bottom of the screen. The speed with which the bots appear can be increased or decreased during the game with the help of left and right keyboard controls.

4. Shooting bullets

This module has functionality to shoot bullets from the gun, aimed at the moving bots. A bullet is fired from the gun when the "spacebar" button on the keyboard is pressed. The bullet travels across the screen tracing a horizontal trajectory. If one or more bot(s) is (are) encountered, the bot is considered to be shot and its corresponding points are added to the total score. The game can be paused and resumed once started with the help of F1 function key.

5. Keeping count of score

This module is associated with keeping count of score during the gameplay so that it can act as both a live tracker and can be called later to display the score at the end of the game. This module also has to take care of the "HeadShots" in the game as a HeadShot carries more points than the score that appears on the body of the bot. The game can be restarted or one can go back to the menu screen.

4. Results and Snapshots

This section contains the results and screenshots of the project in execution. This section gives a basic overview of how the project/game looks while it is being played.

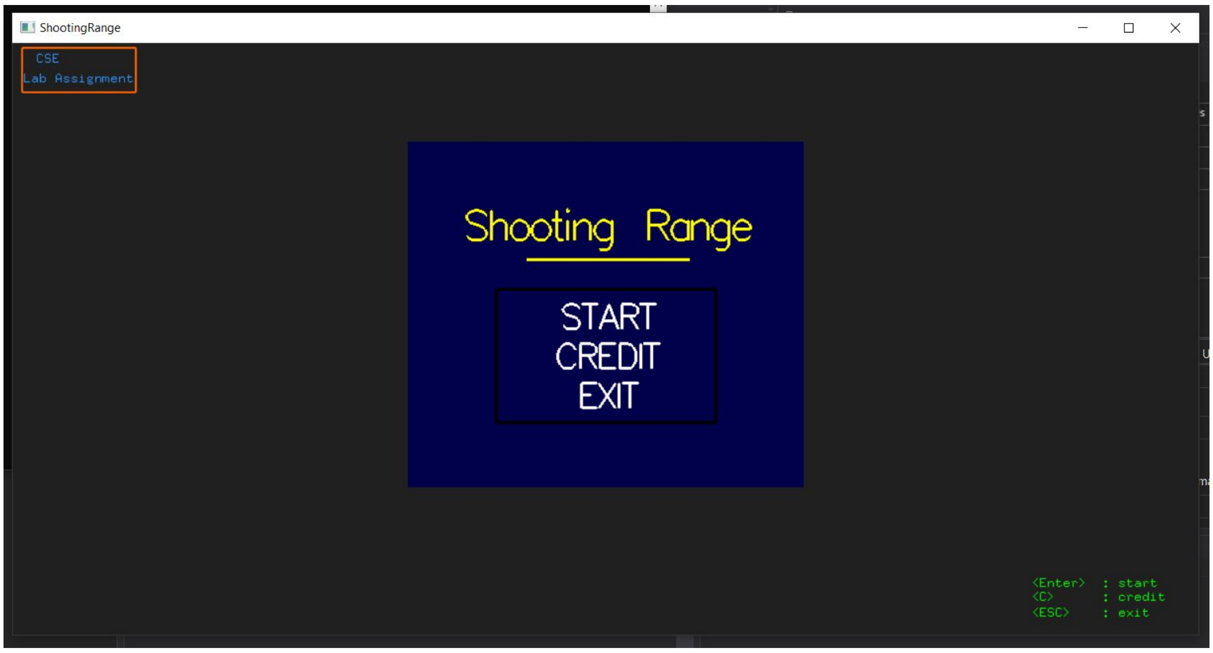


Fig4: MenuPage

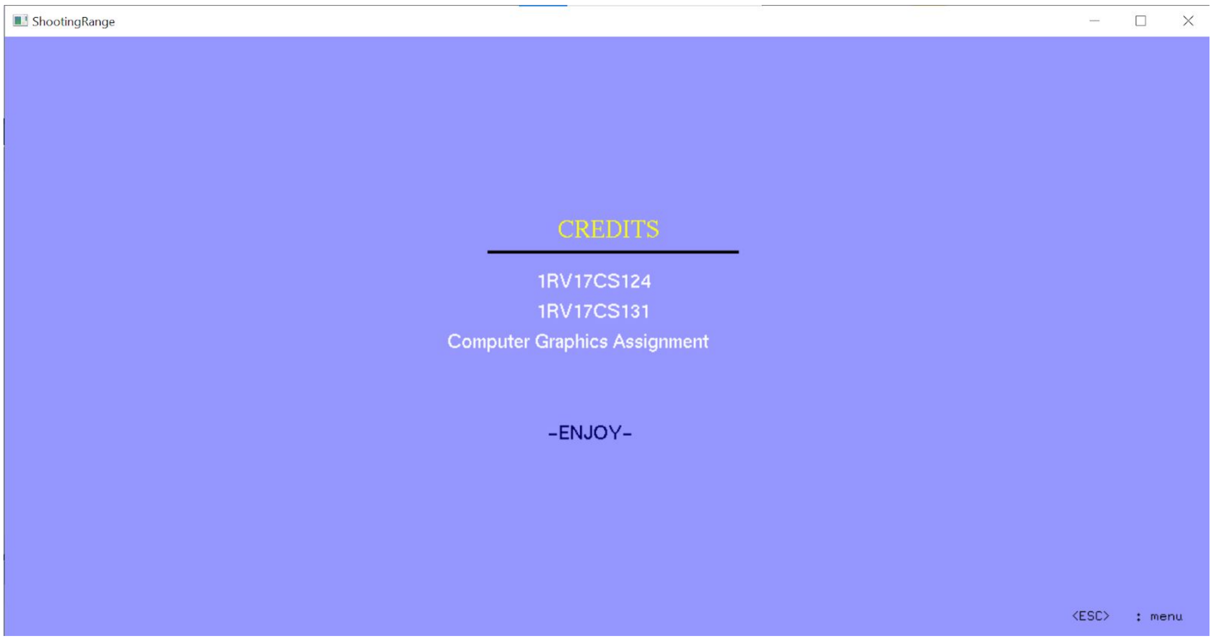
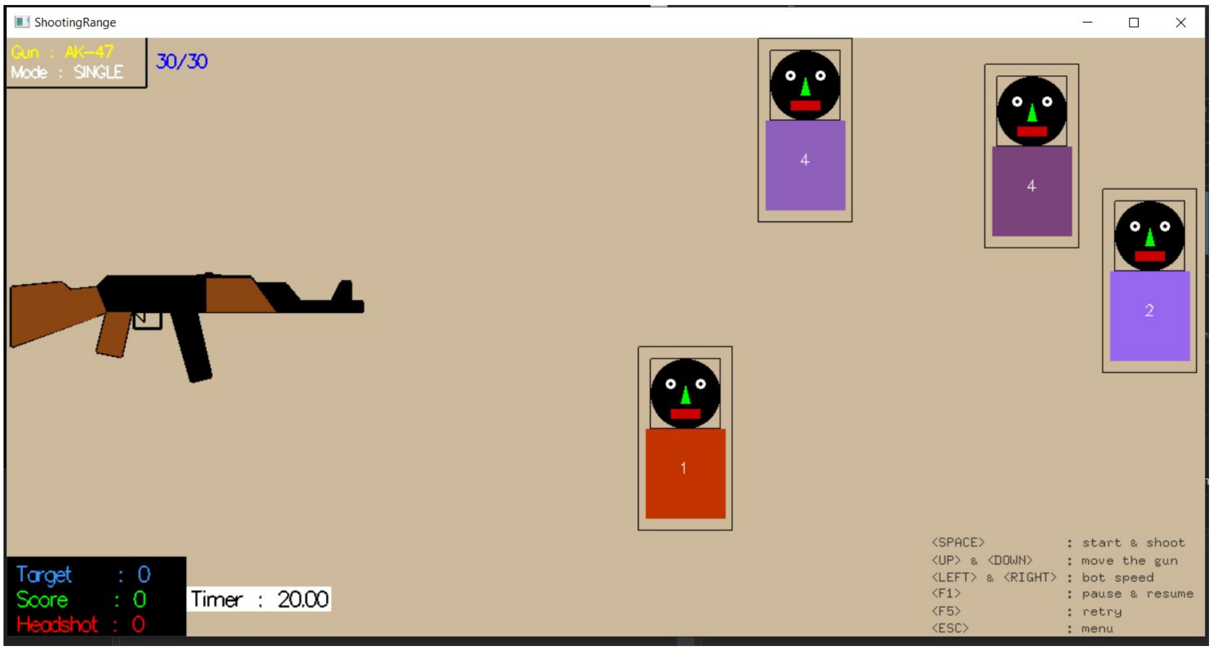


Fig5: CreditsPage



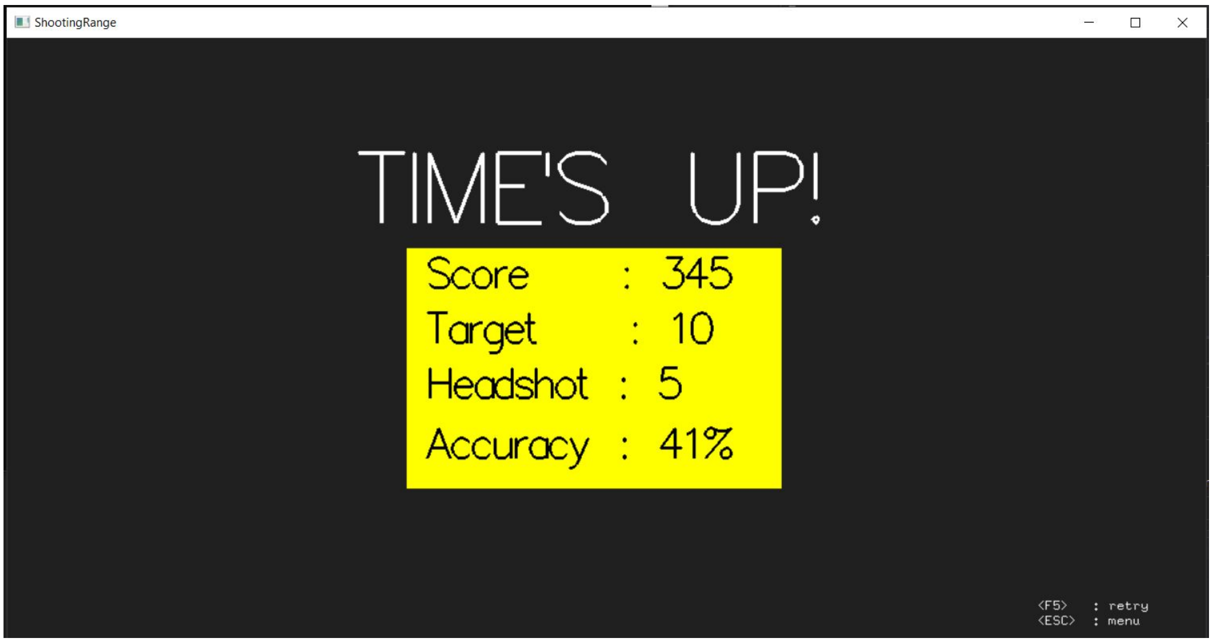
```
Gun  : AK-47
Mode : SINGLE
30/30

Target   : 0
Score    : 0
Headshot : 0

Timer : 20.00

<SPACE>             : start & shoot
<UP> & <DOWN>       : move the gun
<LEFT> & <RIGHT>    : bot speed
<F1>                : pause & resume
<F5>                : retry
<ESC>               : menu
```

Fig6: ShootingRange



```
TIME'S  UP!

Score     :  345
Target    :  10
Headshot  :  5
Accuracy  :  41%

<F5>   : retry
<ESC>  : menu
```

Fig7: ScoreBoard

5. Conclusion

The project was successfully completed and the game was rendered. It has a smooth interface as per plan which mostly involves the keyboard and a single key to give the game a set of simple controls for easy gameplay. The idea behind the simple set of controls was keeping in mind the target audience, being children and especially differently-abled children to improve their hand eye coordination. This game can be used to train children in their cognitive abilities and also for spending their leisure time. The project also has a colourful and vibrant interface to attract children and gamers. Finally, the game serves its purpose.

6. Bibliography

1. Computer Graphics with OpenGL, Donald D. Hearn, M. Pauline Baker, Warren Carithers, 4th Edition, 2010, Pearson Education, ISBN-13: 978-0136053583.
2. The official OpenGL website www.opengl.org
3. A free online encyclopedia, created and edited by volunteers around the world and hosted by the Wikimedia Foundation www.wikipedia.org
4. Hosam, Osama. (2015). Developing Simple Games with OpenGL. 10.13140/RG.2.1.1685.8324.

APPENDIX A- SOURCE CODE

```
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <stdarg.h>
#include <time.h>

#define WINDOW_WIDTH 1200
#define WINDOW_HEIGHT 600

#define FPS 60

#define LOGO 0
#define MENU 1
#define LOAD 2
#define RUN 3
#define CREDITS 4
#define OVER 5
int State = LOGO; // initial Scene

#define MAXBULLET 31 // 30 bullet in a magazine
#define MAXBOT 5 // MAX 5 OBJECTS

int winWidth, winHeight, we, he;
int resume, logoTime = 0;
int numberofBullet, dotCounter, bulletCounter, accuracy;
int gameScore, gameHs, gameTarget; // HS point will be more
double gunY;
double gameTime, notifCount, loadingTime;
char gunMode[10] = "SINGLE";
bool init, spawnBullet, checkTimer, accuracyBool, spawnBot, botMovement, controlSpeed;
int bulletX[MAXBULLET], bulletY[MAXBULLET];

typedef struct { // bot pos.
        double x, y, r, g, b;
        int score, num, visiB, Body, Hs;
}bot_t;
bot_t BOT[MAXBULLET];

void calBotFunc(); //PROTOTYPE

void circle(int x, int y, int r)
{
#define PI 3.1415
        float angle;
```

```
	glBegin(GL_POLYGON);
	for (int i = 0; i < 100; i++)
	{
		angle = 2 * PI*i / 100;
		glVertex2f(x + r * cos(angle), y + r * sin(angle));
	}
	glEnd();
}

void vprint(int x, int y, void *font, const char *string, ...)
{
	va_list ap;
	va_start(ap, string);
	char str[1024];
	vsprintf_s(str, string, ap);
	va_end(ap);

	int len, i;
	glRasterPos2f(x, y);
	len = (int)strlen(str);
	for (i = 0; i < len; i++)
	{
		glutBitmapCharacter(font, str[i]);
	}
}

void vprint2(int x, int y, float size, const char *string, ...) {
	va_list ap;
	va_start(ap, string);
	char str[1024];
	vsprintf_s(str, string, ap);
	va_end(ap);
	glPushMatrix();
	glTranslatef(x, y, 0);
	glScalef(size, size, 1);

	int len, i;
	len = (int)strlen(str);
	for (i = 0; i < len; i++)
	{
		glutStrokeCharacter(GLUT_STROKE_ROMAN, str[i]);
	}
	glPopMatrix();
}

void Init_Globals() {

	gameTime = 20.00;
	numberofBullet = 30;
	resume = 1;
	bulletX[bulletCounter] = notifCount = gameScore = loadingTime = gameHs =
dotCounter = bulletCounter = gunY = accuracy = gameTarget = 0;

	spawnBullet = checkTimer = controlSpeed = accuracyBool = false;
	init = spawnBot = botMovement = true;

	for (int i = 0; i <= MAXBULLET; i++)
		BOT[i] = { 0 };

	calBotFunc(); //call the func

	// reset according to current windows positions
	winWidth = we;
	winHeight = he;
	glViewport(0, 0, we, he);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(-we / 2, we / 2, -he / 2, he / 2, -1, 1);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}

void showLogo() {

	glClearColor(192.0 / 255.0, 192.0 / 255.0, 192.0 / 255.0, 1.0f); //gray
	glClear(GL_COLOR_BUFFER_BIT);

	glColor3f(0, 0, 0);
	circle(0, 5, 175);

	glColor3f(1, 1, 1);
	glLineWidth(4);
	vprint2(-100, -55, 1.25, "GO");

}

void gunColor() {

	glColor3f(0, 0, 0);

	glBegin(GL_QUADS);
	glVertex2f(-410, 25 + gunY);
	glVertex2f(-395, -40 + gunY);
	glVertex2f(-415, -45 + gunY);
	glVertex2f(-435, 25 + gunY);
	glEnd();
	glBegin(GL_QUADS);
	glVertex2f(-500, 59 + gunY);
	glVertex2f(-400, 59 + gunY);
```

```
glVertex2f(-400, 25 + gunY);
glVertex2f(-490, 25 + gunY);
glEnd();
glBegin(GL_QUADS);
glVertex2f(-510, 50 + gunY);
glVertex2f(-500, 60 + gunY);
glVertex2f(-490, 25 + gunY);
glVertex2f(-510, 25 + gunY);
glEnd();
glBegin(GL_QUADS);
glVertex2f(-351, 52 + gunY);
glVertex2f(-320, 52 + gunY);
glVertex2f(-298, 25 + gunY);
glVertex2f(-332, 25 + gunY);
glEnd();
glBegin(GL_QUADS);
glVertex2f(-310, 36 + gunY);
glVertex2f(-245, 36 + gunY);
glVertex2f(-245, 24 + gunY);
glVertex2f(-310, 24 + gunY);
glEnd();
glBegin(GL_QUADS);
glVertex2f(-265, 55 + gunY);
glVertex2f(-255, 55 + gunY);
glVertex2f(-255, 24 + gunY);
glVertex2f(-280, 24 + gunY);
glEnd();

///////////other color////////////
glColor3ub(139, 69, 19);

glBegin(GL_QUADS);
glVertex2f(-475, 25 + gunY);
glVertex2f(-485, -20 + gunY);
glVertex2f(-510, -15 + gunY);
glVertex2f(-500, 25 + gunY);
glEnd();
glBegin(GL_QUADS);
glVertex2f(-595, 50 + gunY);
glVertex2f(-544, 55 + gunY);
glVertex2f(-510, 20 + gunY);
glVertex2f(-595, -10 + gunY);
glEnd();
glBegin(GL_QUADS);
glVertex2f(-540, 47 + gunY);
glVertex2f(-510, 50 + gunY);
glVertex2f(-500, 25 + gunY);
glVertex2f(-535, 12 + gunY);
glEnd();
glBegin(GL_QUADS);
```

```
        glVertex2f(-400, 59 + gunY);
        glVertex2f(-355, 59 + gunY);
        glVertex2f(-330, 25 + gunY);
        glVertex2f(-400, 25 + gunY);
        glEnd();
}

void drawGun() {

        //GUN
        glLineWidth(3);
        glColor3f(0, 0, 0);
        glBegin(GL_LINE_STRIP);
        glVertex2f(-595, 50 + gunY);
        glVertex2f(-545, 55 + gunY);
        glVertex2f(-535, 47 + gunY);
        glVertex2f(-510, 50 + gunY);
        glVertex2f(-500, 60 + gunY);
        glVertex2f(-355, 60 + gunY);
        glVertex2f(-350, 53 + gunY);
        glVertex2f(-320, 53 + gunY);
        glVertex2f(-305, 35 + gunY);
        glVertex2f(-275, 35 + gunY);
        glVertex2f(-265, 55 + gunY);
        glVertex2f(-255, 55 + gunY);
        glVertex2f(-255, 35 + gunY);
        glVertex2f(-243, 35 + gunY);
        glVertex2f(-243, 25 + gunY);
        glVertex2f(-410, 25 + gunY);
        glVertex2f(-435, 25 + gunY);
        glVertex2f(-410, 25 + gunY);
        glVertex2f(-395, -40 + gunY);
        glVertex2f(-415, -45 + gunY);
        glVertex2f(-435, 25 + gunY);
        glVertex2f(-475, 25 + gunY);
        glVertex2f(-485, -20 + gunY);
        glVertex2f(-510, -15 + gunY);
        glVertex2f(-500, 25 + gunY);
        glVertex2f(-595, -10 + gunY);
        glVertex2f(-595, 50 + gunY);
        glEnd();

        //trigger
        glBegin(GL_LINE_STRIP);
        glVertex2f(-472, 25 + gunY);
        glVertex2f(-472, 8 + gunY);
        glVertex2f(-445, 8 + gunY);
        glVertex2f(-445, 25 + gunY);
        glEnd();
        //trigger-2
        glLineWidth(2);
        glBegin(GL_LINE_STRIP);
        glVertex2f(-472, 25 + gunY);
        glVertex2f(-462, 15 + gunY);
        glVertex2f(-462, 25 + gunY);
        glEnd();

        //notch
        glBegin(GL_LINE_STRIP);
        glVertex2f(-401, 60 + gunY);
        glVertex2f(-401, 64 + gunY);
        glVertex2f(-394, 64 + gunY);
        glVertex2f(-394, 60 + gunY);
        glEnd();

        glRectf(-410, 63 + gunY, -385, 60 + gunY);

        gunColor();

}

void drawBullet(double x, double y) {

        glColor3ub(75, 75, 75);
        glRectf(-260 + x, 33 + y, -245 + x, 27 + y);
        glColor3ub(255, 165, 0);
        glBegin(GL_TRIANGLES);
        glVertex2f(-245 + x, 35 + y);
        glVertex2f(-245 + x, 25 + y);
        glVertex2f(-239 + x, 30 + y);
        glEnd();
}

void drawBots(double x, double y, int i) { //random pos, random color, random
score(hs-body)

        glColor4f(0, 0, 0, BOT[i].visiB);
        circle(75 + x, -100 + y, 35); //FACE

        glColor4f(1, 1, 1, BOT[i].visiB);
        circle(90 + x, -90 + y, 5);      //LEFT EYE
        circle(60 + x, -90 + y, 5);      //RIGHT EYE
        glColor4f(0, 0, 0, BOT[i].visiB);
        circle(90 + x, -90 + y, 2.5);    //LEFT EYE-2
        circle(60 + x, -90 + y, 2.5);    //RIGHT EYE-2

        //NOSE
        glColor4f(0, 1, 0, BOT[i].visiB);
        glBegin(GL_TRIANGLES);
        glVertex2f(70 + x, -110 + y);
        glVertex2f(80 + x, -110 + y);
        glVertex2f(75 + x, -90 + y);
        glEnd();
        //NOSE

        glColor4ub(204, 0, 0, BOT[i].visiB);
        glRectf(60 + x, -115 + y, 90 + x, -125 + y);//MOUTH

        glColor4ub(BOT[i].r, BOT[i].g, BOT[i].b, BOT[i].visiB);
        glRectf(35 + x, -135 + y, 115 + x, -225 + y);//BODY - Only bots body will be
changed by random RGB values.

        //checkHitBox // (will be displayed)
        glColor4f(0, 0, 0, BOT[i].visiB);
        glLineWidth(1);
        glBegin(GL_LINE_LOOP);
        glVertex2f(28 + x, -53 + y);
        glVertex2f(28 + x, -237 + y);
        glVertex2f(122 + x, -237 + y);
        glVertex2f(122 + x, -53 + y);
        glEnd();

        //faceHit
        glColor4f(0, 0, 0, BOT[i].visiB);
        glLineWidth(1);
        glBegin(GL_LINE_LOOP);
        glVertex2f(40 + x, -65 + y);
        glVertex2f(40 + x, -135 + y);
        glVertex2f(110 + x, -135 + y);
        glVertex2f(110 + x, -65 + y);
        glEnd();
        //checkHitBox

                ////////////////HS AND TARGET HIT TEXTS//////////////////
        glColor4f(1, 1, 1, BOT[i].visiB);
        vprint2(70 + x, -180 + y, 0.11, "%d", BOT[i].num);

        glLineWidth(3);
        if (BOT[i].Body == 1) {
                BOT[i].Hs = 0;
                if (notifCount == 0)
                        BOT[i].Body = 0;
                glColor3ub(255, 128, 0);
                vprint2(55 + x, -50 + y, 0.2, "%d", BOT[i].score);
        }
        else if (BOT[i].Hs == 1) {
                BOT[i].Body = 0;
                if (notifCount == 0)
                        BOT[i].Hs = 0;
```

```
            glColor3ub(204, 0, 102);
            vprint2(20 + x, -50 + y, 0.25, "%d HS", BOT[i].score);
        }
        ////////////////HS AND TARGET HIT TEXTS//////////////////
}

void showMenu() {

        glClearColor(32.0 / 255.0, 32.0 / 255.0, 32.0 / 255.0, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        glColor3ub(50, 150, 250); // nameSurname
        vprint(-(winWidth / 2) + 25, (winHeight / 2) - 20, GLUT_BITMAP_8_BY_13,
"CSE");
        vprint(-(winWidth / 2) + 11.5, (winHeight / 2) - 40, GLUT_BITMAP_8_BY_13,
"Lab Assignment");

        glColor3ub(255, 100, 0);
        glLineWidth(2);
        glBegin(GL_LINE_LOOP);
        glVertex2f(-(winWidth / 2) + 10, (winHeight / 2) - 5);
        glVertex2f(-(winWidth / 2) + 125, (winHeight / 2) - 5);
        glVertex2f(-(winWidth / 2) + 125, (winHeight / 2) - 50);
        glVertex2f(-(winWidth / 2) + 10, (winHeight / 2) - 50);
        glEnd();

        glColor3ub(0, 0, 75); // menuRect
        glRectf(-200, 200, 200, -150);

        glColor3f(1, 1, 0);
        glLineWidth(3);
        vprint2(-143, 100, 0.3, "Shooting Range");

        glBegin(GL_LINES);
        glVertex2f(-80, 80);
        glVertex2f(85, 80);
        glEnd();

        glColor3f(0, 0, 0);
        glLineWidth(3);
        glBegin(GL_LINE_LOOP);
        glVertex2f(-110, 50);
        glVertex2f(112, 50);
        glVertex2f(112, -85);
        glVertex2f(-110, -85);
        glEnd();

        glColor3f(1, 1, 1); // Buttons
        vprint2(-45, 10, 0.25, "START");
        vprint2(-50, -30, 0.25, "CREDIT");
```

```
	vprint2(-27, -70, 0.25, "EXIT");

	glColor3ub(0, 255, 0);//Hints
	vprint((winWidth / 2) - 170, -(winHeight / 2) + 50, GLUT_BITMAP_8_BY_13,
"<Enter>  : start");
	vprint((winWidth / 2) - 170, -(winHeight / 2) + 35, GLUT_BITMAP_8_BY_13, "<C>
: credit");
	vprint((winWidth / 2) - 170, -(winHeight / 2) + 20, GLUT_BITMAP_8_BY_13,
"<ESC>    : exit");

}

void infoBoxes() {

	glColor3f(1, 1, 1);
	glRectf(-(winWidth / 2) + 180, -(winHeight / 2) + 50, -(winWidth / 2) + 325,
-(winHeight / 2) + 25); //background for timer

	glColor3f(0, 0, 0);
	vprint2(-(winWidth / 2) + 185, -(winHeight / 2) + 30, 0.15, "Timer : %4.2f",
gameTime);

	glColor3f(0, 0, 0);
	glRectf(-(winWidth / 2), -(winHeight / 2) + 80, -(winWidth / 2) + 180, -(winHeight /
2)); //background score, etc.

	glColor3ub(50, 150, 255);
	vprint2(-(winWidth / 2) + 10, -(winHeight / 2) + 55, 0.15, "Target   : %d",
gameTarget);
	glColor3f(0, 1, 0);
	vprint2(-(winWidth / 2) + 10, -(winHeight / 2) + 30, 0.15, "Score   : %d",
gameScore);
	glColor3f(1, 0, 0);
	vprint2(-(winWidth / 2) + 10, -(winHeight / 2) + 5, 0.15, "Headshot : %d", gameHs);

	glColor3f(0, 0, 0);
	glLineWidth(2);
	glBegin(GL_LINE_STRIP);
	glVertex2f(-(winWidth / 2), (winHeight / 2) + -50);
	glVertex2f(-(winWidth / 2) + 140, (winHeight / 2) + -50);
	glVertex2f(-(winWidth / 2) + 140, (winHeight / 2));
	glEnd();

	glColor3f(1, 1, 0);
	vprint2(-(winWidth / 2) + 5, (winHeight / 2) + -20, 0.115, "Gun : AK-47");
	glColor3f(1, 1, 1);
	vprint2(-(winWidth / 2) + 5, (winHeight / 2) + -40, 0.115, "Mode : %s", gunMode);

	if (numberofBullet == 0) {
		glColor3f(1, 0, 0);
		vprint2(-(winWidth / 2) + 155, (winHeight / 2) + -30, 0.13, "%2d/0",
numberofBullet);
	}
	else {
		glColor3f(0, 0, 1);
		vprint2(-(winWidth / 2) + 150, (winHeight / 2) + -30, 0.13, "%2d/%2d",
numberofBullet, MAXBULLET - 1);
	}

	glColor3ub(50, 50, 50);
	vprint((winWidth / 2) - 275, -(winHeight / 2) + 90, GLUT_BITMAP_8_BY_13,
"<SPACE>         : start & shoot");
	vprint((winWidth / 2) - 275, -(winHeight / 2) + 72, GLUT_BITMAP_8_BY_13,
"<UP> & <DOWN>   : move the gun");
	vprint((winWidth / 2) - 275, -(winHeight / 2) + 55, GLUT_BITMAP_8_BY_13,
"<LEFT> & <RIGHT> : bot speed");
	vprint((winWidth / 2) - 275, -(winHeight / 2) + 39, GLUT_BITMAP_8_BY_13,
"<F1>            : pause & resume");
	vprint((winWidth / 2) - 275, -(winHeight / 2) + 22, GLUT_BITMAP_8_BY_13,
"<F5>            : retry");
	vprint((winWidth / 2) - 275, -(winHeight / 2) + 5, GLUT_BITMAP_8_BY_13,
"<ESC>           : menu");

}

void showRun() {

	glClearColor(205.0 / 255.0, 185.0 / 255.0, 156.0 / 255.0, 1.0f); //dust
	glClear(GL_COLOR_BUFFER_BIT);

	if (strcmp(gunMode, "SINGLE") == 0 && spawnBullet)
		drawBullet(bulletX[bulletCounter], bulletY[bulletCounter]); // if press
spacebar

	if (spawnBot == true)
		for (int i = 0; i < MAXBOT; i++)
			drawBots(BOT[i].x, BOT[i].y, i);

	drawGun();
	infoBoxes();
}

void showLoad() {

	glClearColor(32.0 / 255.0, 32.0 / 255.0, 32.0 / 255.0, 1.0f);
	glClear(GL_COLOR_BUFFER_BIT);

	glColor3f(1, 1, 1);
	vprint2(-180, 0, 0.5, "INITIALIZING");
	glColor3f(1, 1, 0);
	for (int i = 0; i < dotCounter; i++)
		vprint2(-57 + i * 30, -50, 0.3, "*");

}

void showCredits() {

	glClearColor(150.0 / 255.0, 150.0 / 255.0, 255.0 / 255.0, 1.0f);
	glClear(GL_COLOR_BUFFER_BIT);

	glColor3f(1, 1, 0);
	vprint(-50, 100, GLUT_BITMAP_TIMES_ROMAN_24, "CREDITS");

	glColor3f(0, 0, 0);
	glBegin(GL_LINES);
	glVertex2f(-120, 85);
	glVertex2f(130, 85);
	glEnd();

	glColor3f(1, 1, 1);
	vprint(-71, 50, GLUT_BITMAP_HELVETICA_18, "1RV17CS124");
	vprint(-40, 20, GLUT_BITMAP_HELVETICA_18, "1RV17CS131");
	vprint(-160, -10, GLUT_BITMAP_HELVETICA_18, "Computer Graphics
Assignment");

	glColor3ub(0, 0, 102);
	vprint(-60, -100, GLUT_BITMAP_HELVETICA_18, "-ENJOY- ");


	glColor3ub(0, 0, 0);
	vprint((winWidth / 2) - 140, -(winHeight / 2) + 20, GLUT_BITMAP_8_BY_13,
"<ESC>   : menu");

}

void showOver() {

	glClearColor(32.0 / 255.0, 32.0 / 255.0, 32.0 / 255.0, 1.0f);
	glClear(GL_COLOR_BUFFER_BIT);

	glColor3f(1, 1, 1);
	glLineWidth(4);
	vprint2(-250, 115, 0.7, "TIME'S UP!");

	glColor3f(1, 1, 0);
	glRectf(-200, 90, 175, -150);

	glLineWidth(3);
	glColor3f(0, 0, 0);
	vprint2(-180, 50, 0.3, "Score   : %d", gameScore);
	vprint2(-180, -5, 0.3, "Target   : %d", gameTarget);
	vprint2(-180, -60, 0.3, "Headshot : %d", gameHs);
	vprint2(-180, -120, 0.3, "Accuracy : %d%%", (accuracy * 100) / (MAXBULLET - 1 -
numberofBullet));

	glColor3f(1, 1, 1);
	vprint((winWidth / 2) - 170, -(winHeight / 2) + 30, GLUT_BITMAP_8_BY_13,
"<F5>   : retry");
	vprint((winWidth / 2) - 170, -(winHeight / 2) + 15, GLUT_BITMAP_8_BY_13,
"<ESC>  : menu");
}

void proj_display() {

	switch (State) {
	case LOGO:    showLogo(); break;
	case MENU:    showMenu(); break;
	case LOAD:    showLoad(); break;
	case RUN:     showRun(); break;
	case CREDITS: showCredits(); break;
	case OVER:    showOver(); break;
	}

	glutSwapBuffers();
}

void onKeyDown(unsigned char key, int x, int y)
{

	if (key == 27 && State == MENU)// ESC-Exit (only in menu)
		exit(0);
	if (key == 27 && State == RUN)
		State = MENU;
	if (key == 13 && State == MENU)
		State = LOAD;
	if (State == MENU)
		if (key == 'C' || key == 'c')
			State = CREDITS;
	if (key == 27 && State == CREDITS)
		State = MENU;
	if (key == 27 && State == OVER)
		State = MENU;

	glutPostRedisplay();
}

void onKeyUp(unsigned char key, int x, int y)
{
	if (resume == 1) {
		if (key == ' ' && State == RUN && numberofBullet > 0) // space
		{
			spawnBullet = true;
			checkTimer = true;

			if (bulletX[bulletCounter] == 0)
				bulletY[bulletCounter] = gunY;

			if (spawnBullet) {
				if (bulletX[bulletCounter] > -50 && bulletX[bulletCounter] <
50) {
					numberofBullet--;
					if (numberofBullet == -1)
						spawnBullet = false;
				}
			}
		}
	}

	glutPostRedisplay();
}

void onSpecialKeyDown(int key, int x, int y)
{
	switch (key) {
	case GLUT_KEY_UP:
		if (State == RUN && gunY < 180 && resume) // will not moved behind the
information boxes.
			gunY += 10;
		break;
	case GLUT_KEY_DOWN:
		if (State == RUN && gunY > -180 && resume)
			gunY -= 10;
		break;
	case GLUT_KEY_LEFT:
		controlSpeed = false;
		break;

	case GLUT_KEY_RIGHT:
		controlSpeed = true;
		break;
	}

}

void onSpecialKeyUp(int key, int x, int y)
{

	if (key == GLUT_KEY_F5 && State == OVER)
		State = LOAD;
	if (key == GLUT_KEY_F5 && State == RUN && checkTimer)
		State = LOAD;

	if (key == GLUT_KEY_F1 && State == RUN && checkTimer && resume == 1) {
		resume = 0;
		checkTimer = false;		botMovement = false;
	}
	else if (key == GLUT_KEY_F1 && State == RUN && !checkTimer && resume ==
0) {
		resume = 1;
		checkTimer = true;		botMovement = true;
	}

	glutPostRedisplay();
}

void calBulletFunc() {

	if (State == RUN && checkTimer) {

		if (bulletX[bulletCounter] >= 840) {
			if (accuracyBool)
				accuracy++;
			spawnBullet = false;
			accuracyBool = false;
			bulletX[bulletCounter] = 0;
		}
		if (strcmp(gunMode, "SINGLE") == 0 && spawnBullet)
			bulletX[bulletCounter] += 15;
	}
}

void calGunFunc() {

	if (State == RUN && spawnBullet && checkTimer) {

		if (strcmp(gunMode, "SINGLE") == 0) {

			for (int i = 0; i < MAXBOT; i++) {
				if (bulletX[bulletCounter] >= 274 + BOT[i].x &&
bulletX[bulletCounter] <= 289 + BOT[i].x)
				{
					if (bulletY[bulletCounter] <= -160 + BOT[i].y &&
bulletY[bulletCounter] >= -250 + BOT[i].y) { // BODY PART
						BOT[i].visiB = 0;
						BOT[i].score = rand() % 19 + 1;//normal points /
between 1-20
						BOT[i].Body = 1;
						gameScore += BOT[i].score;
						notifCount = 25;
```

```
                                                gameTarget++;
                                                accuracyBool = true;
                                        }
                                        else if (bulletY[bulletCounter] <= -90 + BOT[i].y &&
bulletY[bulletCounter] >= -160 + BOT[i].y) { // HS PART
                                                BOT[i].visiB = 0;
                                                BOT[i].score = rand() % 79 + 21; // extra hs
points / 20 - 100
                                                BOT[i].Hs = 1;
                                                gameScore += BOT[i].score;
                                                notifCount = 25;
                                                gameTarget++;
                                                gameHs++;
                                                accuracyBool = true;
                                        }
                                }
                        }
                }
        }
}

void calBotFunc() {

        if (State == RUN && resume == 1) {

                for (int m = 0; m < MAXBOT; m++) {

                        if (controlSpeed)
                                BOT[m].y += 8.0;
                        else
                                BOT[m].y += 5.0;

                        if (BOT[m].y >= 552.0 || init) {
                                BOT[m] = { {rand() % 15 + (115.0 * m)}, {(rand() % 225 +
350.0) * (-1)}, //Positions
                                {rand() % 256 * 1.0}, {rand() % 256 * 1.0}, {rand() % 256 *
1.0}, //colors
                                0, {rand() % 4 + 1}, 255, 0, 0 };
                                notifCount = -5;
                        }
                }
        }

}

void onTimer(int v) {

        glutTimerFunc((1000 / FPS), onTimer, 0);

        if (State == LOGO) {
```

```
        logoTime++;
        if (logoTime >= 75)
            State = MENU;
    }

    if (State == LOAD) {
        dotCounter = (dotCounter + 1) % 4;
        loadingTime += 0.5;
        if (loadingTime >= 25) {
            Init_Globals(); State = RUN;
        }
    }

    if (State == RUN && checkTimer && gameTime != 0) {
        gameTime -= 0.0165;
        if (gameTime < 0) {
            gameTime = 0;
            State = OVER;
            checkTimer = false;
        }
    }

    ////////////////////////////////////////////////////// call the functions
//////////////////////////////////////////////////////
    calGunFunc();
    calBulletFunc();
    calBotFunc();  init = false;
    ////////////////////////////////////////////////////// call the functions
//////////////////////////////////////////////////////

    if (notifCount > -5 && resume)
        notifCount -= 1;

    glutPostRedisplay();
}

void onResize(int w, int h)
{
    we = w; he = h;
    winWidth = w;
    winHeight = h;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-w / 2, w / 2, -h / 2, h / 2, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    proj_display();
}
```

```
void Init() {

        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        Init_Globals();
}

void main(int argc, char *argv[]) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
        glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("ShootingRange");

        glutDisplayFunc(proj_display);
        glutReshapeFunc(onResize);

        glutKeyboardFunc(onKeyDown);
        glutSpecialFunc(onSpecialKeyDown);
        glutKeyboardUpFunc(onKeyUp);
        glutSpecialUpFunc(onSpecialKeyUp);

        srand(time(NULL));
        glutTimerFunc((1000 / FPS), onTimer, 0);

        Init();
        glutMainLoop();
}
```