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

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

"SPIDER SHOOTER"

COMPUTER GRAPHICS LAB (16CS73)

OPEN ENDED EXPERIMENT REPORT

VII SEMESTER

2020-2021

Submitted by

Dhara R Rachh (1RV18CS404)
Vedika Agarwal (1RV17TE063)

Under the Guidance of

MAMATHA T
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 "**SPIDER SHOOTER**" has been carried out by **Dhara R. Rachh (1RV18CS404), Vedika Agarwal (1RV17TE063),** 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
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, **Dhara R. Rachh (1RV18CS424), Vedika Agarwal (1RV18CS427)** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled **"SPIDER SHOOTER"** 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 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
Date: 06-01-2021

Dhara R Rachh
Vedika Agarwal

Contents

1. INTRODUCTION

This project is about implementing **"SPIDER SHOOTER"** using a set of OpenGL functions. It contains 2 buckets one red and other green that catch the respective color spider's and a blue gun in order to shoot white colored spider's. If opposite color spider's fall into opposite buckets then the base will be being eaten and hence the score grows into negative direction and on completion of the base, the life goes down.

1.1 Computer Graphics

Computer graphics is one of the most exciting and rapidly growing computer fields. It is also an extremely effective medium for communication between man and computer; a human being can understand the information content of a displayed diagram or perspective view much faster than he can understand a table of numbers or text containing the same information. Thus computer graphics is being used more extensively. There is a lot of development in hardware and software required to generate images, and nowadays the Cost of hardware and software is dropping rapidly. Due to this, interactive computer graphics is becoming available to more and more people.

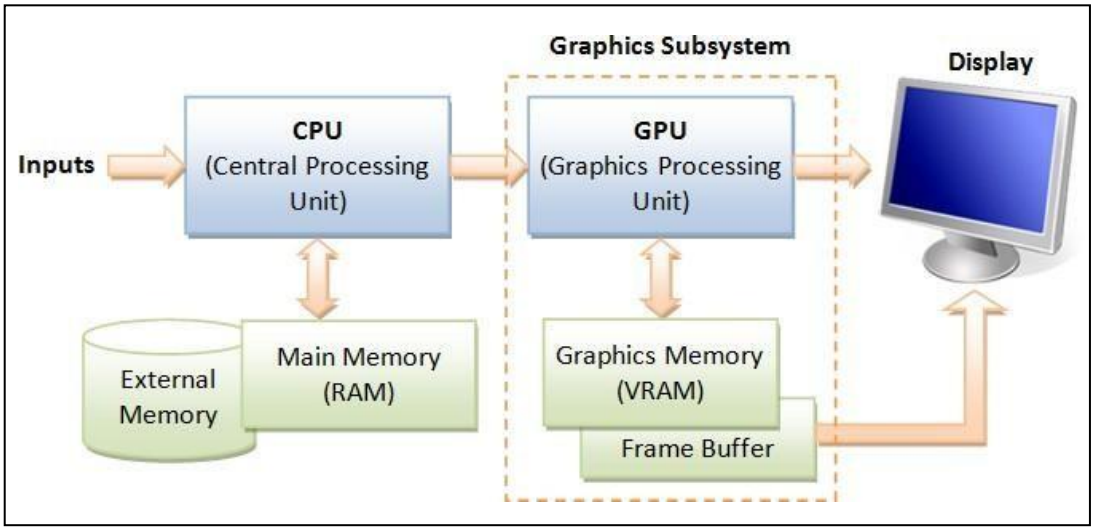

Figure 1.1 Computer Graphics System

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube (CRT) screens soon after the introduction of computers themselves. It has grown to include the creation, storage and manipulation of models and manipulation of models and images of objects. These models come from a diverse and expanding set of fields, and include physical, mathematical, engineering, architectural, and even conceptual structures, natural phenomena, and soon.

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 a keyboard, mouse, or touch sensitive panel on the screen. The handling of such devices is included in the study of computer graphics, because of the close relationship between the input devices and the display.

1.2 OpenGL

OpenGL (open graphics library) is a standard specification defining a cross language across platform API for writing applications that produce 2D and 3D computer graphics. OpenGL was developed by silicon graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization and flight simulation. It is also used in video games.

OpenGL serves two main purpose:

1. To hide the complexities of interfacing with different 3D accelerators, by presenting programmer with a single, uniform API
2. To hide the differing capabilities of hardware platforms, by requiring that all Implementations support the full OpenGL, featureset.

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects.

Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are:

- **Main GL:** Library has names that begin with the letter **gl**and are stored in a library usually referred to asGL.
- **OpenGL Utility Library (GLU):** This library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations. Functions in the GLU library begins with the letters**glu**.
- **OpenGL Utility Toolkit(GLUT):** This provides the minimum functionality that should be accepted in any modern windowing system. For the X window system, this library is called GLX, for windows, it is WGL or Wiggle. And for Macintosh, it is AGL. Rather than using different library for each system, we use a readily available library called **GLUT.**

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL. Instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, ormolecules.

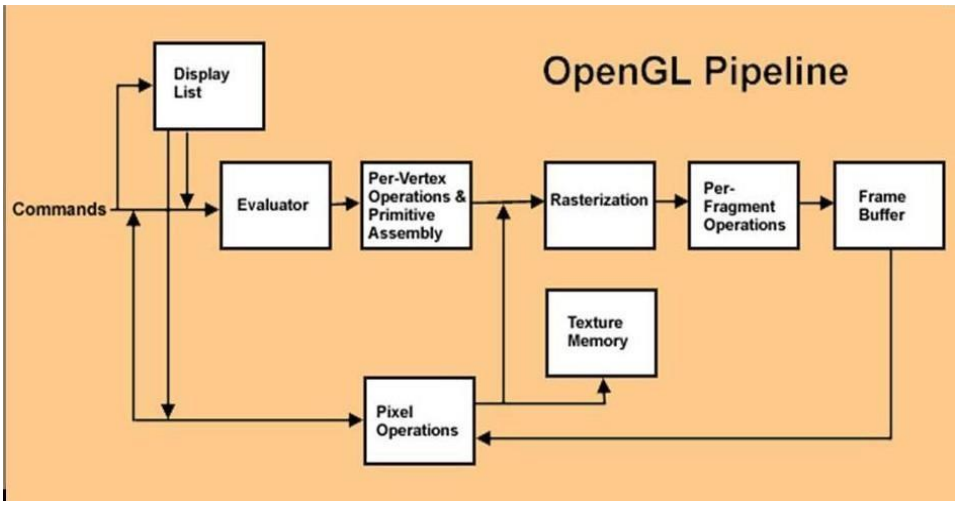

Figure 1.2 OpenGL Graphics Architecture

2.OpenGL SYNTAX

- **glVertex*():-**The glVertex function commands are used within**glBegin/glEnd**pairstospecify point, line, and polygon vertices. The current color, normal, texture coordinates, and fog coordinate are associated with the vertex when glVertexis called. When only x and y are specified, z defaults to 0 and w defaults to1.

- **glColor*( )**:-It sets a few four-valued RGBA color. glColor has two major forms and glColor4. glColor3 variants specify new red, green and blue values explicitly and set the current alpha value to 1.0 implicitly. glColor4 variants specify all four color components explicitly.

GL_PROJECTION or GL_TEXTURE.

- **voidglutInit(int *argc, char**argv):-**Initializes GLUT, the arguments from main are passed in and can be used by the application.

- **voidglutInitDisplayMode(unsigned int mode):-**Requests a display with properties in mode. The value of mode is determined by the logical OR of options including the color model and buffering.

- **voidglutInitWindowSize(int width, int height):-** Specifies the initial position of the left corner of the window in pixels.

- **intglutCreateWindow(char *title):-**A window on the display. The string title is used to label the window. The return value provides references to the window that can be used when there are multiple windows.

- **voidglutMouseFunc(void *f(int button, int state, int x, int y):-**Register the callback function f. The callback function returns the button,the state of button after the event and the position of the mouse relative to the top-left corner of the window.

- **void glutKeyboardFunc(void(*func) (void)):-**This function is called every time you enter key to resume the game or when you press r,g,b key to go back to the initial screen or

when you press Esc key to exit from the application.

- **voidglutDisplayFunc(void (*func) (void)):-**Register the display function that is executed when the window needs to be redrawn.

- **voidglutSpecialFunc(void(*func)(void)):-**This function is called when you press special keys in the keyboard like arrow keys, function keys etc. In our program, the function is invoked when the up arrow or down arrow key is pressed for selecting the options in the main menu and when the left or right arrow key is pressed for moving the object(car) accordingly.

- **voidMouseFunc(void (*func) void)):-** This function is invoked when mouse keys are pressed. This function is used as an alternative to the previous function i.e, it is used to move the object(car) to right or left in our program by clicking left and right button.

- **voidglutMainLoop( ):-** Cause the program to enter an event processing loop. It is last statement in main function

2. SYSTEM REQUIREMENTS SPECIFICATION

3.1 HARDWARE REQUIREMENTS

- RAM: 2GB andhigher
- Hard Disk: 40GB andhigher
- Keyboard: QWERTYkeyboard
- Mouse: 2 or 3 ButtonMouse
- Monitor: 1024 x 768 displayResolution

3.2 SOFTWAREREQUIREMENTS

- Programming Language: C/C++ using OpenGL
- Operating System: Linux Operating System
- Compiler – GCC compiler
- Graphics library: GL/glut.h
- OpenGL2.0

3. USER INTERACTION

Using Alphabetical Keys:

R: MOVING red bucket for collecting red colored spider's

G: MOVING green bucket for collecting red colored spider's

B: MOVING AND USING the gun

P: PAUSE the game

Q: Quit the game

Using Special Keys:

LEFT ARROW: Moving the bucket in left direction

RIGHT ARROW: Moving the bucket in right direction

UP AND DOWN ARROWS: Rotating the gun

4. SNAPSHOTS

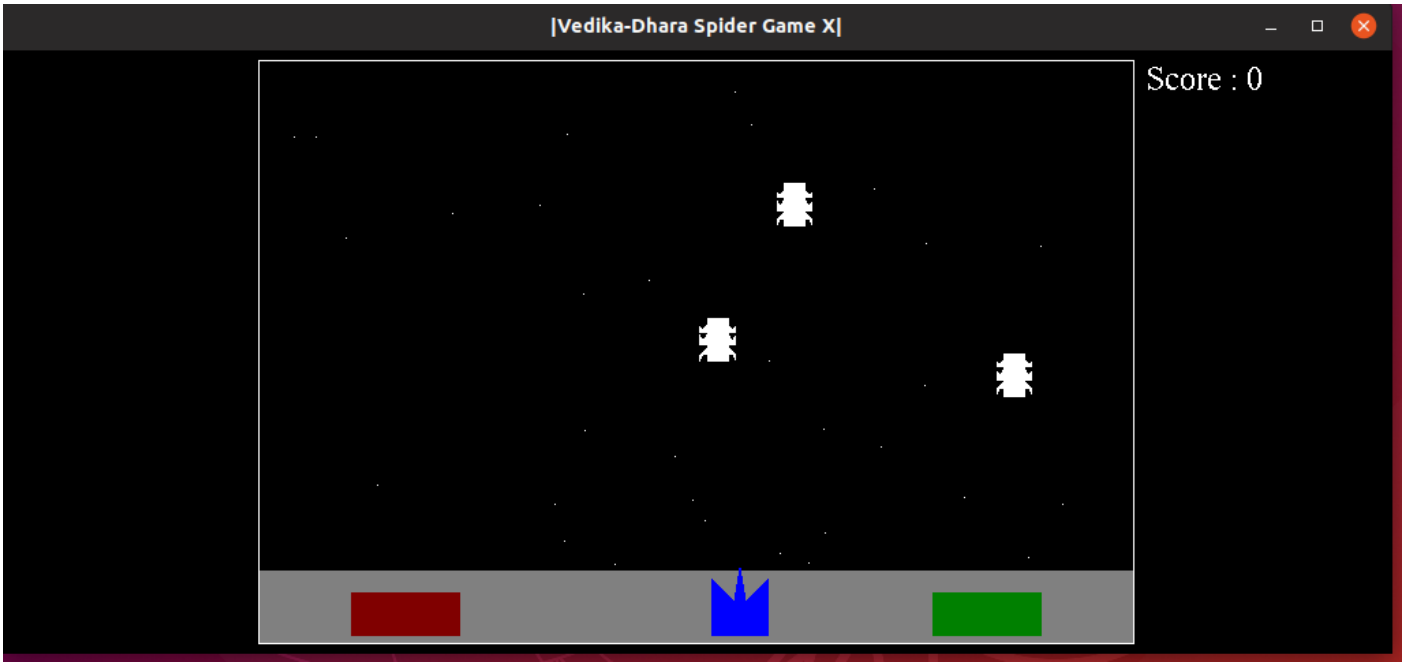

Fig 4.1: The game just started with white spiders approaching

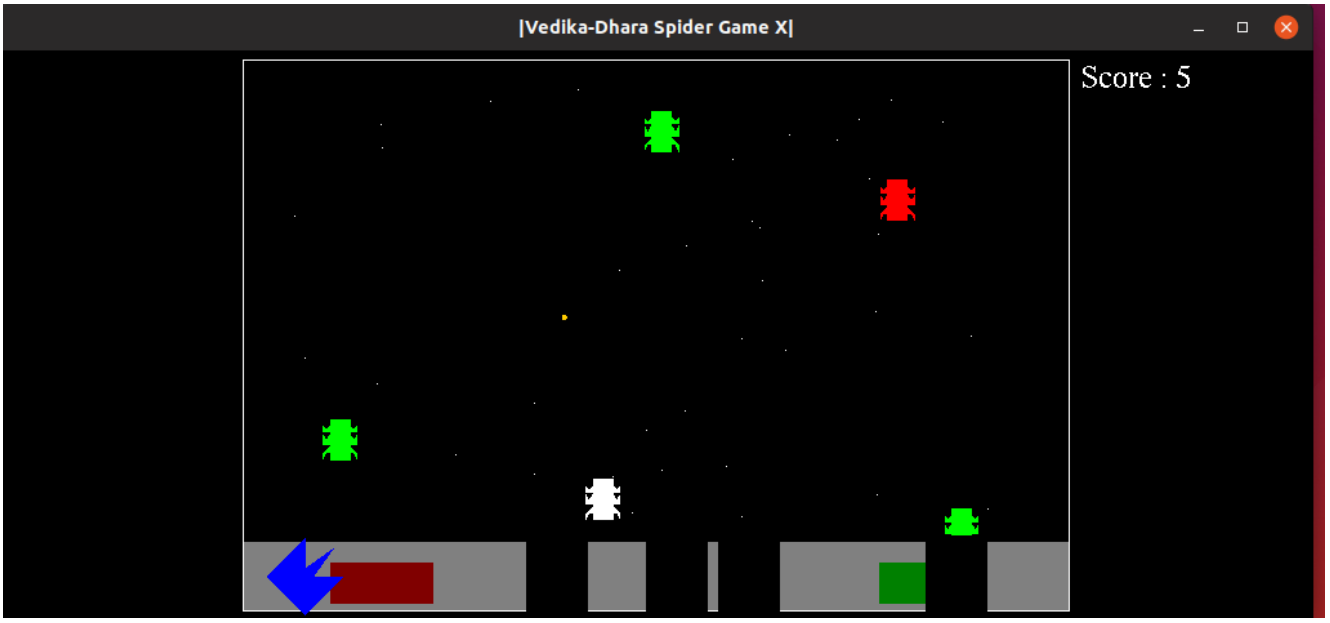

Fig 4.2: Rotating gun to kill the while colored spider and collect green and red colored spiders

Fig 4.3: The game gets over as the spider falls on our gun

5. CONCLUSION

We have attempted to design and implement "**SPIDER SHOOTER**". OpenGL supports enormous flexibility in the design and the use of OpenGL graphics programs. The presence of many built- in classes methods take care of much functionality and reduce the job of coding as well as makes the implementation simpler. We have implemented the project making it user-friendly and error free as possible. We regret any errors that may have inadvertently crept in.

6. REFERENCES

http://lazyfoo.net/tutorials/OpenGL/02_matrices_and_coloring_polygons/index.phphttp://csciwww.etsu.edu/barrettm/4157/OpenGL%20Functions.htm

https://www.glprogramming.com/red/chapter01.html

https://www2.cs.duke.edu/courses/cps124/fall01/resources/ogl_ref.pdf

.

9. APPENDICES

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <cmath>
#include <GL/glut.h>
using namespace std;


#define PI 3.141592653589
#define DEG2RAD(deg) (deg * PI / 180)


// Function Declarations
void drawScene();
void update(int value);
void drawBucket(float ht,float wid);
void drawBox(float ht,float wid);
void drawBall(float rad);
void drawStar(float size);
void drawTriangle();
void drawSpider();
void initRendering();
void handleResize(int w, int h);
void handleKeypress1(unsigned char key, int x, int y);
void handleKeypress2(int key, int x, int y);
void handleMouseclick(int button, int state, int x, int y);
void spiderCreator(int value);
void backMusic(int value);
void starCreator();
void ballCreator(float tank_x, float tank_y, float theta);
float drag(float position);
void GetOGLPos(int x, int y);
void dragwithmouse(int x, int y);
void drawBitmapText(char* string,float x, float y, float z);
float distance(float x1, float y1, float x2, float y2);


// Global Variables
int i,j,k,l,m,n;
int a,b,c,d,e,f;
int score = 0;
bool stop = false;
```

```
bool over = false;



char selected = 'b';
char scoreStr[100];
char fscore[100];
float box_ht = 4.0f;
float box_wid = 6.0f;
float score_x = box_wid/2 + 0.1f;
float score_y = box_ht/2 - 0.2f;
float base_x = 0.0f;
float base_y = -1.75f;
float base_ht = 0.5f;
float base_wid = box_wid;
float ball_x;
float ball_y;
float ball_speed = 0.04f;
float ball_rad = 0.02f;
float tank_x = 0.0f;
float tank_y = -1.75f;
float tank_wid = 0.5f;
float theta = 0.0f;
float rbuck_x = -2.0f;
float rbuck_y = -1.8f;
float gbuck_x = 2.0f;
float gbuck_y = -1.8f;
float buck_ht = 0.3f;
float buck_wid = 0.75f;
float spi_x = 0.0f;
float spi_y = 2.0f;
int spi_col = 2;
float spi_wid = 0.15f;
float spi_ht = 0.3f;
float star_size = 0.009f;
int cannon_flag = 1;
int cannon_time = 0;
int rotcan = 0;
float mousex;
float mousey;
float mouseposx;
float mouseposy;


typedef struct Vect {
        float x;
        float y;
}Vector2D;
```

```
//Star's charactersitics.
typedef struct s {
	Vector2D pos;
	float speed;
}Star;



//CannonBall's characteristics.
typedef struct ball {
	Vector2D position;
	Vector2D speed;
	float angle;
	int blasted;
}CannonBall;


//The spiders characteristics.
typedef struct Spid {
	char color;
	int state;
	Vector2D position;
	float speed;
}SpidChar;


//Array to hold stars.
Star stars[1000];
int n_stars = 100;


//Array to hold all the spiders.
SpidChar spiders[100000];
int n_spiders = 0;


//Array to hold cannons.
CannonBall cannonBalls[1000000];
int n_balls = 0;


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


	// Initialize GLUT
	glutInit(&argc, argv);
```

```
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

        scanf("%f%f",&box_ht,&box_wid);


        int w = glutGet(GLUT_SCREEN_WIDTH);
        int h = glutGet(GLUT_SCREEN_HEIGHT);
        int windowWidth = w*3/4;
        int windowHeight = h*2/3;


        glutInitWindowSize(windowWidth, windowHeight);
        glutInitWindowPosition((w - windowWidth) / 2, (h - windowHeight) / 2);


        glutCreateWindow("|X Cyber Spider Mayhem X|");  // Setup the window
        initRendering();


        // Register callbacks
        glutDisplayFunc(drawScene);
        glutIdleFunc(drawScene);
        glutKeyboardFunc(handleKeypress1);
        glutSpecialFunc(handleKeypress2);
        glutMouseFunc(handleMouseclick);
        glutReshapeFunc(handleResize);


        glutMouseFunc(handleMouseclick);
        glutMotionFunc(dragwithmouse);


        starCreator();


        glutTimerFunc(10, update, 0);
        glutTimerFunc(2000, spiderCreator, 1);
        glutTimerFunc(10, backMusic, 2);


        glutMainLoop();
        return 0;
}


// Function to draw objects on the screen
void drawScene() {
```

```
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glPushMatrix();


        // Draw Box
        glTranslatef(0.0f, 0.0f, -5.0f);
        glColor3f(1.0f, 1.0f, 1.0f);
        drawBox(box_ht,box_wid);


        //Draw base
        glPushMatrix();

glTranslatef(base_x, base_y, 0.0f);
        glColor3f(0.5f, 0.5f, 0.5f);


drawBucket(base_ht, base_wid);
        glPopMatrix();


        //Draw red-bucket
        glPushMatrix();
        glTranslatef(rbuck_x, rbuck_y, 0.0f);
        glColor3f(0.5f, 0.0f, 0.0f);
        drawBucket(buck_ht,buck_wid);
        glPopMatrix();


        //Draw green-bucket
        glPushMatrix();
        glTranslatef(gbuck_x, gbuck_y, 0.0f);
        glColor3f(0.0f, 0.5f, 0.0f);
        drawBucket(buck_ht,buck_wid);
        glPopMatrix();


        //Draw stars
        for (a=0; a<n_stars; a++)
        {
                glPushMatrix();
                glTranslatef(stars[a].pos.x, stars[a].pos.y, -0.1f);
                glColor3f(1.0f, 1.0f, 1.0f);
                drawStar(star_size);
                glPopMatrix();
```

```
	}


	// Draw cannon-ball.
	for (a=0; a<n_balls; a++)
	{
		if (cannonBalls[a].blasted == 0)
		{
			glPushMatrix();
			glTranslatef(cannonBalls[a].position.x, cannonBalls[a].position.y, 0.0f);
			glColor3f(1.0f, 0.8f, 0.0f);
			drawBall(ball_rad);
			glPopMatrix();
		}
	}


	// Draw Tank
	glPushMatrix();

glTranslatef(tank_x, tank_y, 0.0f);
	glRotatef(theta, 0.0f, 0.0f, 1.0f);
	glScalef(0.5f, 0.5f, 0.5f);
	glColor3f(0.0f, 0.0f, 1.0f);


drawTriangle();
	glPopMatrix();


	//Draw Spider
	for (j=0; j<n_spiders; j++)
	{
		if (spiders[j].state == 1)
		{
			glPushMatrix();
			glTranslatef(spiders[j].position.x, spiders[j].position.y, 0.0f);
			if (spiders[j].color == 0)
				glColor3f(1.0f, 0.0f, 0.0f);
			if (spiders[j].color == 1)
				glColor3f(0.0f, 1.0f, 0.0f);
			if (spiders[j].color == 2)
				glColor3f(1.0f, 1.0f, 1.0f);
			drawSpider();
			glPopMatrix();
		}
	}
```

```
	//Destroying platform where spider exists.
	for (k=0; k<n_spiders; k++) {
		if (spiders[k].state == 1 && spiders[k].color == 2 && spiders[k].position.y <=
base_y)
		{
			glPushMatrix();
			glTranslatef(spiders[k].position.x, spiders[k].position.y, 0.0f);
			glColor3f(0.0f, 0.0f, 0.0f);
			drawBucket(1.2*base_ht, 3*spi_wid);
			glPopMatrix();
		}
	}


	//Print Score.
	if (over == false)
	{
		glColor3f(1.0f, 1.0f, 1.0f);
		sprintf(scoreStr,"Score : %d",score);
		drawBitmapText(scoreStr, score_x, score_y, 0.0f);
	}


	//Draw final score.
	if (over == true)
	{
		glColor3f(1.0f, 1.0f, 1.0f);
		sprintf(fscore,"Final Score:%d",score);
		drawBitmapText(fscore,0.0f, -1.0f, 0.0f);
		sprintf(scoreStr,"GameOver! Press q to quit");

drawBitmapText(scoreStr,0.0f, 0.0f, 0.0f);
	}


	glPopMatrix();
	glutSwapBuffers();
}


// Function to handle all calculations in the scene
// updated evry 10 milliseconds
void update(int value) {


	//overall flag to pause the game.
	if (stop == false)
```

```
	{
		// Handle ball collisions with box
		for (c=0; c<n_balls; c++)
		{
			if((cannonBalls[c].blasted == 0) && (cannonBalls[c].position.x + ball_rad >= box_wid/2 || cannonBalls[c].position.x - ball_rad <= -box_wid/2))
			{
				cannonBalls[c].speed.x *= -1;
			}
		}
		//Handle containing boxes into the walls.
		if (rbuck_x + (buck_wid/2) >= box_wid/2)
			rbuck_x = (box_wid - buck_wid)/2;
		if (rbuck_x - (buck_wid/2) <= -box_wid/2)
			rbuck_x = -(box_wid - buck_wid)/2;
		if (gbuck_x + (buck_wid/2) >= box_wid/2)
			gbuck_x = (box_wid - buck_wid)/2;
		if (gbuck_x - (buck_wid/2) <= -box_wid/2)
			gbuck_x = -(box_wid - buck_wid)/2;


		//Handle containing tank into the walls.
		if (tank_x + tank_wid/2 >= box_wid/2)
			tank_x = (box_wid/2) - tank_wid/2;
		if (tank_x - tank_wid/2 <= -box_wid/2)
			tank_x = -(box_wid/2 - tank_wid/2);


		//If tank reaches end of destroyed base.
		for (l=0; l<n_spiders; l++)
		{
			if ((spiders[l].position.y <= base_y) && (tank_x - tank_wid/2 <= spiders[l].position.x + spi_wid) && (tank_x - tank_wid/2 >= spiders[l].position.x - spi_wid) && (spiders[l].color == 2) && spiders[l].state == 1)
			{


tank_x = spiders[l].position.x + spi_wid + tank_wid/2;
			}
			if ((spiders[l].position.y <= base_y) && (tank_x + tank_wid/2 >= spiders[l].position.x - spi_wid) && (tank_x + tank_wid/2 <= spiders[l].position.x + spi_wid) && (spiders[l].color == 2) && spiders[l].state == 1)
			{
				tank_x = spiders[l].position.x - spi_wid - tank_wid/2;
			}
			if ((spiders[l].position.y <= base_y) && (spiders[l].position.y >= base_y + base_ht/2) && (tank_x <= spiders[l].position.x + spi_wid/2) && (tank_x >= spiders[l].position.x - spi_wid/2) && spiders[l].state == 1)
```

```
                {
                        stop = true;
                        over = true;
                }
        }


        //Make the spider fall
        for (i=0; i<n_spiders; i++)
                spiders[i].position.y -= spiders[i].speed;


        //Make the white spider sit on the platform.
        for (i=0; i<n_spiders; i++) {
                if (spiders[i].color == 2 && spiders[i].position.y <= base_y)
                {
                        spiders[i].speed = 0;
                }
        }


        //Stars fall down.
        for (i=0; i<n_stars; i++)
        {
                stars[i].pos.y -= stars[i].speed;


                if (stars[i].pos.y < base_y+base_ht/2)
                {
                        stars[i].pos.y = 1.9f;
                        stars[i].pos.x = (-2.85 + static_cast <float> (rand()) /( static_cast
<float> (RAND_MAX/(2.85+2.85))));
                        stars[i].speed = 0.004f + static_cast <float> (rand()) /( static_cast
<float> (RAND_MAX/(0.004f)));
                }
        }


        //Collect spider in correct color.
        for (i=0; i<n_spiders; i++) {
                if (spiders[i].state == 1 && spiders[i].color == 0 &&


spiders[i].position.x <= rbuck_x + buck_wid/2 && spiders[i].position.x >= rbuck_x - buck_wid/2 &&
spiders[i].position.y <= rbuck_y) {
                        spiders[i].speed = 0;
                        spiders[i].state = 0;
```

```
                        score += 5;
                }
                if (spiders[i].state == 1 && spiders[i].color == 1 && spiders[i].position.x
<= gbuck_x + buck_wid/2 && spiders[i].position.x >= gbuck_x - buck_wid/2 && spiders[i].position.y
<= gbuck_y) {
                        spiders[i].speed = 0;
                        spiders[i].state = 0;
                        score += 5;
                }
        }


        //killing colored spiders that have crossed game space.
        for (m=0; m<n_spiders; m++)
        {
                if (spiders[m].state == 1 && spiders[m].position.y < base_y-0.3f )
                {
                        spiders[m].state = 0 ;
                        score -= 2;
                }
        }


        //to add one second gap between lasers.
        cannon_time++;
        if (cannon_time >= 100)
        {
                cannon_flag = 1;
                cannon_time = 0;
        }


        //moving the cannon-ball.
        for (b=0; b<n_balls; b++)
        {

if (cannonBalls[b].blasted == 0)
                {
                        cannonBalls[b].position.x += (cannonBalls[b].speed.x*cos(DEG2RAD(PI/2
+ cannonBalls[b].angle)));
                        cannonBalls[b].position.y += (cannonBalls[b].speed.y*sin(DEG2RAD(PI/2
+ cannonBalls[b].angle)));
                }
        }


        //cannon-ball explodes on going out of bounds.
        for (d=0; d<n_balls; d++)
```

```
		{

if (cannonBalls[d].blasted == 0 &&(cannonBalls[d].position.y + ball_rad > box_ht/2 ||
cannonBalls[d].position.y - ball_rad < -box_ht/2))
			{
				cannonBalls[d].blasted = 1;
			}
		}


		//game over if spider falls on us.
		for (n=0; n<n_spiders; n++)
		{
			if (spiders[n].state == 1 && (spiders[n].position.x <= tank_x + tank_wid/2 &&
spiders[n].position.x >= tank_x - tank_wid/2) && spiders[n].position.y <= tank_y)
			{
				stop = true;
				over = true;
				system("aplay gameover.wav &");
			}
		}


		//killing the spiders from cannon-balls.
		for (e=0; e<n_balls; e++)
		{
			if (cannonBalls[e].blasted == 0)
			{
				for (f=0; f<n_spiders; f++)
				{
					if (spiders[f].state == 1 && spiders[f].color == 2 &&
spiders[f].position.y > base_y)
					{
						if (((cannonBalls[e].position.x + ball_rad >=
spiders[f].position.x - spi_wid/2 && cannonBalls[e].position.x + ball_rad <= spiders[f].position.x
+ spi_wid/2) || (cannonBalls[e].position.x - ball_rad <= spiders[f].position.x + spi_wid/2 &&
cannonBalls[e].position.x - ball_rad >= spiders[f].position.x - spi_wid/2)) &&
(cannonBalls[e].position.y + ball_rad >=  spiders[f].position.y - spi_ht/2 &&
cannonBalls[e].position.y + ball_rad <= spiders[f].position.y + spi_ht/2))

{
							spiders[f].state = 0;
							cannonBalls[e].blasted = 1;
							score += 5;
							system ("aplay kill.wav &");
						}
					}
				}
```

```
                        }
                }
        }
        glutTimerFunc(10, update, 0);
}


void ballCreator(float spawn_x, float spawn_y, float angle) {


        CannonBall ball;
        ball.position.x = tank_x - (0.3f*sin(DEG2RAD(theta)));
        ball.position.y = tank_y + (0.3f*cos(DEG2RAD(theta)));
        ball.speed.x = ball_speed;
        ball.speed.y = ball_speed;
        ball.angle = theta;
        ball.blasted = 0;


        cannonBalls[n_balls++] = ball;
}


void spiderCreator(int value) {


        if (stop == false)
        {
                SpidChar spidey;
                spidey.state = 1;
                spidey.color = rand()%3;
                spidey.speed = 0.004f + static_cast <float> (rand()) /( static_cast <float>
(RAND_MAX/(0.004f)));
                spidey.position.y = 1.9f;
                spidey.position.x = (-2.85 + static_cast <float> (rand()) /( static_cast <float>
(RAND_MAX/(2.85+2.85))));


                spiders[n_spiders++] = spidey;
        }
        glutTimerFunc(2000, spiderCreator, 1);
}


void backMusic(int value) {


        if (over == false)
```

```
		system("aplay back.wav &");


	glutTimerFunc(6000, backMusic, 2);
}


void starCreator() {


	int i;
	for (i=0; i<n_stars; i++)
	{
		Star temp;
		temp.pos.x = (-2.85 + static_cast <float> (rand()) /( static_cast <float>
(RAND_MAX/(2.85+2.85))));
		temp.pos.y = (-1.9 + static_cast <float> (rand()) /( static_cast <float>
(RAND_MAX/(1.9+1.9))));
		temp.speed = 0.004f + static_cast <float> (rand()) /( static_cast <float>
(RAND_MAX/(0.004f)));


		stars[i] = temp;
	}
}


void drawBox(float ht,float wid) {


	glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
	glBegin(GL_QUADS);
	glVertex2f(-wid / 2, -ht / 2);
	glVertex2f(wid / 2, -ht / 2);
	glVertex2f(wid / 2, ht / 2);
	glVertex2f(-wid / 2, ht / 2);
	glEnd();
	glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
}




void drawBucket(float ht,float wid) {


	glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
```

```
glBegin(GL_QUADS);
        glVertex2f(-wid / 2, -ht / 2);
        glVertex2f(wid / 2, -ht / 2);
        glVertex2f(wid / 2, ht / 2);
        glVertex2f(-wid / 2, ht / 2);
        glEnd();
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
}


void drawSpider() {




        glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
        glBegin(GL_QUADS);
        glVertex2f(-spi_wid/2, -spi_ht/2);
        glVertex2f(-spi_wid/2, spi_ht/2);
        glVertex2f(spi_wid/2, spi_ht/2);
        glVertex2f(spi_wid/2, -spi_ht/2);
        glVertex2f(spi_wid/2, 0.04f);
        glVertex2f(spi_wid/2, -0.04f);
        glVertex2f((spi_wid/2)+0.05f, 0.04f);
        glVertex2f((spi_wid/2)+0.05f, -0.04f);
        glVertex2f(spi_wid/2, 0.1f);
        glVertex2f(spi_wid/2, 0.05f);
        glVertex2f((spi_wid/2)+0.05f, 0.1f);
        glVertex2f((spi_wid/2)+0.05f, 0.05f);
        glVertex2f(spi_wid/2, -0.1f);
        glVertex2f(spi_wid/2, -0.05f);
        glVertex2f((spi_wid/2)+0.05f, -0.15f);
        glVertex2f((spi_wid/2)+0.05f, -0.1f);
        glVertex2f(-spi_wid/2, 0.04f);
        glVertex2f(-spi_wid/2, -0.04f);
        glVertex2f((-spi_wid/2)-0.05f, 0.04f);
        glVertex2f((-spi_wid/2)-0.05f, -0.04f);
        glVertex2f(-spi_wid/2, 0.1f);
        glVertex2f(-spi_wid/2, 0.05f);
        glVertex2f((-spi_wid/2)-0.05f, 0.1f);
        glVertex2f((-spi_wid/2)-0.05f, 0.05f);
        glVertex2f(-spi_wid/2, -0.1f);
        glVertex2f(-spi_wid/2, -0.05f);
        glVertex2f((-spi_wid/2)-0.05f, -0.15f);
        glVertex2f((-spi_wid/2)-0.05f, -0.1f);
        glEnd();
}
```

```
void drawBall(float rad) {


	glBegin(GL_TRIANGLE_FAN);
	for(int i=0 ; i<360 ; i++) {
		glVertex2f(rad * cos(DEG2RAD(i)), rad * sin(DEG2RAD(i)));
	}
	glEnd();
}


void drawTriangle() {


	glBegin(GL_TRIANGLES);
	glVertex2f(0.4f, 0.4f);
	glVertex2f(-0.4f, -0.4f);
	glVertex2f(0.4f, -0.4f);


	glVertex2f(-0.4f, 0.4f);
	glVertex2f(-0.4f, -0.4f);
	glVertex2f(0.4f, -0.4f);
	glVertex2f(0.1f,0.0f);
	glVertex2f(-0.1f,0.0f);
	glVertex2f(0.0f,0.6f);
	glEnd();
}


void drawStar(float size) {


	glBegin(GL_TRIANGLES);
	glVertex2f(size/3, size/3);
	glVertex2f(-size/3, size/3);
	glVertex2f(0, -1.414*size/3);
	glVertex2f(size/3, -size/3);
	glVertex2f(-size/3, -size/3);
	glVertex2f(0, 1.414*size/3);

glEnd();
}


// Initializing some openGL 3D rendering options
void initRendering() {
```

```
	glEnable(GL_DEPTH_TEST);	// Enable objects to be drawn ahead/behind one another
	glEnable(GL_COLOR_MATERIAL);	// Enable coloring
	glClearColor(0.0f, 0.0f, 0.0f, 1.0f);	// Setting a background color
}



// Function called when the window is resized
void handleResize(int w, int h) {


	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(45.0f, (float)w / (float)h, 0.1f, 200.0f);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}


void handleKeypress1(unsigned char key, int x, int y) {


	if (key == 27) {
		exit(0);	// escape key is pressed

}
	if (key == 113 && stop == true && over == true)
	{
		exit(0);
	}
	if (key == 114)		//red-box (r)
		selected = 'r';
	if (key == 103)			//green-box (g)
		selected = 'g';
	if (key == 98)			//blue-cannon (b)
		selected = 'b';
	if (key == 112) {
		stop = (stop == true ? false : true); //pause (p)
	}
	if (key == 32) {


		if (cannon_flag == 1)
		{
			cannon_flag = 0;
```

```
                        system("aplay laser.wav &");
                        ballCreator(tank_x, tank_y, theta); // hit space to shoot.
                }
        }
}


void handleKeypress2(int key, int x, int y) {


        if (key == GLUT_KEY_LEFT)
        {

if (selected == 'b')
                        tank_x -= 0.1;
                else if (selected == 'r')
                        rbuck_x -= 0.1;
                else
                        gbuck_x -= 0.1;
        }
        if (key == GLUT_KEY_RIGHT)
        {
                if (selected == 'b')
                        tank_x += 0.1;
                else if (selected == 'r')
                        rbuck_x += 0.1;
                else
                        gbuck_x += 0.1;
        }
        if (key == GLUT_KEY_UP)
        {
                theta += 15;
        }

else if (key == GLUT_KEY_DOWN)
        {
                theta -= 15;
        }
}







void handleMouseclick(int button, int state, int x, int y) {
```

```
	if(stop == true)
		return;

	GetOGLPos(x,y);
	mousex = mouseposx;
	mousey = mouseposy;


	if (state == GLUT_DOWN)
	{
		if (button == GLUT_LEFT_BUTTON)
		{
			if(distance(mousex,mousey,tank_x,tank_y)<tank_wid)
			{
				rotcan = 0;
				selected = 'b';

}
			else if(distance(mousex,mousey,gbuck_x,gbuck_y)<buck_wid/2)
			{
				rotcan = 0;
				selected = 'g';
			}
			else if(distance(mousex,mousey,rbuck_x,gbuck_y)<buck_wid/2)
			{
				rotcan = 0;
				selected = 'r';
			}
		}
		if(button == GLUT_RIGHT_BUTTON)
		{
			if(distance(mousex,mousey,tank_x,tank_y)<0.5)
			{
				rotcan=1;
			}
			else
			{
				rotcan=0;

}
		}
	}
	glutPostRedisplay();
}


void GetOGLPos(int x, int y)
{
```

```
    GLint viewport[4];
    GLdouble modelview[16];
    GLdouble projection[16];
    GLfloat winX, winY, winZ;
    GLdouble posX, posY, posZ;

    glGetDoublev( GL_MODELVIEW_MATRIX, modelview );
    glGetDoublev( GL_PROJECTION_MATRIX, projection );
    glGetIntegerv( GL_VIEWPORT, viewport );

    winX = (float)x;
    winY = (float)viewport[3] - (float)y;
    glReadPixels( x, int(winY), 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &winZ );

    gluUnProject( winX, winY, winZ, modelview, projection, viewport, &posX, &posY, &posZ);
        mouseposx=posX*100/2;
        mouseposy=(posY*100)/2;
}



float drag(float position)
{
       float  move;
       if(distance(mousex,mousey,position,tank_y)<0.7)
       {
              move=2*(mousex-position);
              if(move>0)
              {
                     position+=(move*0.3);
              }
              else if(move<0)
              {
                     position+=(move*0.3);
              }
       }
       return position;
}


void dragwithmouse(int x,int y)
{
       if(stop == true)

return;

       GetOGLPos(x,y);
       mousex=mouseposx;
```

```
	mousey=mouseposy;

	if(selected == 'r')
	{
		rbuck_x=drag(rbuck_x);
	}
	else if(selected == 'g')
	{
		gbuck_x=drag(gbuck_x);
	}
	else if(selected == 'b')
	{
		tank_x=drag(tank_x);
	}
	if(rotcan && distance(mousex,mousey,tank_x,tank_y)<1.0f)
	{
		if(tank_x-mousex>0)
		{
			if(theta + 1 < 90)
				theta += 1;
		}
		else
		{
			if(theta - 1 > -90)

theta -= 1;
		}
	}


}


float distance(float x1, float y1, float x2, float y2)
{
	return sqrt(((x1-x2)*(x1-x2)) + ((y1-y2)*(y1-y2)));
}


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