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

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

"Road Rage Game"

COMPUTER GRAPHICS LAB (16CS73)

OPEN ENDED EXPERIMENT REPORT

VII SEMESTER

2020-2021

Submitted by

Shetty Rohan 1RV17CS199
Yashwanth YS 1RV17CS194

Under the Guidance of

Dr. Hemavathy R.
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 "Road Rage Game" has been carried out by **Shetty Rohan (1RV17CS199) Yashwanth Y S (1RV17CS194),** 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.

Dr. Hemavathy R.
Faculty In-charge,
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, **Shetty Rohan (1RV17CS199) &Yashwanth Y S (1RV17CS194)** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled **"Road Rage 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 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: 31-12-2020

Shetty Rohan (1RV17CS199)
Yashwanth Y S (1RV17CS194)

ABSTRACT

A 2D graphics-based ball game “Bricks Breaker Ball Game” is a great start for a student who starts learning computer graphics & visualization. The development of the game has large scope to learn computer graphics from scratch. We used OpenGL utility toolkit to implement the algorithm, written it in C++ language.

There is still scope left in the development of project like, after “Game Over” a list should show top ten high scores. Another great improvement to the game would be if the game could be accessed and played by multiple players at the same time. Even the interface could look better and be more user friendly.

In the future we hope we would implement these modifications for a better experience while playing this game. Finally, we could say by developing the game we have learnt the basics of computer graphics and in the future, by improving it further, we shall learn more. It would be our pleasure if we could develop this game using a 3D graphics package.

ACKNOWLEDGEMENTS

While presenting our project on “Bricks Breaker Ball Game”, we feel that it is our duty to acknowledge the help rendered to us by various persons. We would like to convey our thanks to the Principal, the HoD (CSE) Dr. Ramakanth Kumar P (RVCE, Bangalore), for being kind enough to provide us with an opportunity to do a project in this institution. We would greatly mention the enthusiastic influence provided by Dr. Hemavathy R and Prof. Mamatha T, our project guide, for her ideas and co-operation showed on us during the venture and making this project a great success.

We are very much pleasured to express our sincere gratitude to the friendly cooperation showed by all the staff members of Computer Science Department, RVCE. We would also like to thank our friends for helping us with doubts and clarifications and supporting us in spirit to see this project through.

CONTENTS

1. INTRODUCTION

'Road Rage' is a single player 2D car racing game wherein the user operates the car using the keyboard while trying to dodge obstacles and trying to pick up coins in order to maximize the score. The game runs in an endless loop until the user crashes his car with an obstacle. The difficulty level of the game increases as the user progresses in the game.

The program has a simple interactive interface for the user to play the game and provides brief instructions of the game. The game window displays the score, level and speed of the car along with the actual gameplay. The game has two screens, one that contains game instructions and information regarding high score and the other is the actual game screen.

The game is completely developed using C++ programming language with the help of some basic OpenGL functions. The game is able to achieve its objectives and the user can play the game effectively with all features as mentioned previously. The program is a good beginner's project as it covers various components of the computer graphics programming and at the same time allows creativity to be expressed.

1.1 Computer Graphics:

Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computers themselves. It has grown to include the creation, storage, 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 so on. Computer graphics today is largely interactive. 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 patter-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 OpenGL Interface:

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:

1. **Main GL:** Library has names that begin with the letter gl and are stored in a library usually referred to as GL.
2. **OpenGL Utility Library (GLU):** This library uses only GL functions but contains code for creating common objects and simplifying viewing.
3. **OpenGL Utility Toolkit (GLUT):** This provides the minimum functionality that should be accepted in any modern windowing system.

1.3 OpenGL Overview:

- OpenGL(Open Graphics Library) is the interface between a graphic program and graphics hardware. ***It is streamlined.*** In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.

- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- ***It is system-independent.*** It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- ***It is a state machine.*** At any moment during the execution of a program there is a current model transformation
- ***It is a rendering pipeline.*** The rendering pipeline consists of the following steps:
    - Defines objects mathematically.
    - Arranges objects in space relative to a viewpoint.
    - Calculates the color of the objects.
    - Rasterizes the objects.

2. SYSTEM SPECIFICATION

2.1 HARDWARE REQUIREMENTS:

- Dual Core Processor
- 2GB RAM
- 40GB Hard disk
- Mouse and other pointing devices
- Keyboard

2.2 SOFTWARE REQUIREMENTS:

- Programming language – C++ using OpenGL
- Operating system – Windows/Linux operating system
- Compiler – C++ Compiler
- Graphics library – GL/glut.h
- OpenGL 2.0

2.3 FUNCTIONAL REQUIREMENTS:

OpenGL APIs:

If we want to have a control on the flow of program and if we want to interact with the window system then we use OpenGL API'S. Vertices are represented in the same manner internally, whether they are specified as two-dimensional or three-dimensional entities, everything that we do are here will be equally valid in three dimensions. Although OpenGL is easy to learn, compared with other APIs, it is nevertheless powerful. It supports the simple three dimensional programs and also supports the advanced rendering techniques.

GL/glut.h:

We use a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

The application program uses only GLUT functions and can be recompiled with the GLUT library for other window system. OpenGL makes a heavy use of macros to increase code readability and avoid the use of magic numbers. In most implementation, one of the include lines.

3. ABOUT THE PROJECT

3.1 Overview:

Our game is a simple ball with bat game. The bat will be moved according to the movement of the mouse. And the ball will move randomly in the created window. When the ball hits the right, left, or top wall – we will refer to the window border as a wall - it will return back. When it hits the bottom wall it will not only return back but it will increase the score of the computer, but if the player can hold it by the bat, his score will be increased.

3.2 User Interface:

The interface is mainly concentrated on use of mouse and keyboard. Clicking right button of mouse displays a menu which has various options which helps in changing color of bat, ball and background..

By pressing P and R we can pause and restart the game. By pressing N we can move from the current window to next window.

3.3 Purpose:

The aim of this project is to develop a graphics package which supports basic operations which include building a using Open GL. The package must also has a user-friendly interface. The objective of developing this model was to design and apply the skills we learnt in class.

3.4 Scope:

It provides most of the features that a graphics model should have. It is developed in C language. It has been implemented on LINUX platform. The graphics package designed here provides an interface for the users for playing 2D GAME using bat and ball.

4. IMPLEMENTATION

4.1 FUNCTIONS IN OPEN GL

- **void glClear(glEnum mode);**

Clears the buffers namely color buffer and depth buffer. mode refers to GL_COLOR_BUFFER_BIT or DEPTH_BUFFER_BIT.

- **void glutSwapBuffers();**

Swaps the front and back buffers.

- **void glLoadIdentity( );**

Sets the current transformation matrix to identity matrix.

- **void glEnable(GLenum feature);**

Enables an OpenGL feature. Feature can be GL_DEPTH_TEST (enables depth test for hidden surface removal), GL_LIGHTING (enables for lighting calculations), GL_LIGHTi (used to turn on the light for number i), etc.

- **void glutBitmapCharacter(void *font, int character);**

Without using any display lists, `glutBitmapCharacter` renders the `character` in the named bitmap `font`. The fonts used are GLUT_BITMAP_TIMES_ROMAN_24, GLUT_BITMAP_ HELVETICA_18, GLUT_BITMAP_8_BY_13.

- **void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);**

Defines an orthographic viewing volume with all the parameters measured from the centre of the projection plane.

- **void glutInit(int *argc, char **argv);**

Initializes GLUT; the arguments from main are passed in and can be used by the application.

- **void glutInitDisplayMode(unsigned int mode);**

Requests a display with the properties in the mode; the value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

- **void glutCreateWindow(char *title);**

Creates a window on display; the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

- **void glutPostRedisplay(void);**

Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

- **void glutMainLoop();**

Causes the program to enter an event-processing loop.

- **void glutDisplayFunc(void (*func)(void))**

Registers the display function func that is executed when the window needs to be redrawn.

- **void glutMouseFunc(void *f(int button, int state, int x, int y)**

Registers the mouse callback function f. The callback function returns the button (GLUT_LEFT_BUTTON,etc., the state of the button after the event (GLUT_DOWN), and the position of the mouse relative to the top-left corner of the window.

- **void glutKeyboardFunc(void *f(char key, int width, int height))**

Registers the keyboard callback function f. The callback function returns the ASCII code of the key pressed and the position of the mouse.

- **void glClearColor(GLclampf r, GLclampf g, GLclamp b, Glclamp a)**

Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

- **void glColor3[b I f d ub us ui](TYPE r, TYPE g, TYPE b)**

Sets the present RGB colors. Valid types are bytes(b), int(i), float(f), double(d), unsigned byte(ub), unsigned short(us), and unsigned int(ui). The maximum and minimum value for floating point types are 1.0 and 0.0 respectively, whereas the maximum and minimum values of the discrete types are those of the type, for eg, 255 and 0 for unsigned bytes.

- **void glutTimerFunc(unsigned int msecs, void (*func)(int value), value);**

glutTimerFunc registers the timer callback func to be triggered in at least msecs milliseconds. The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc. Multiple timer callbacks at same or differing times may be registered simultaneously.

- **void glutSpecialFunc(void (*func)(int key, int x, int y));**

glutSpecialFunc sets the special keyboard callback for the current window. The special keyboard callback is triggered when keyboard function or directional keys are pressed. The key callback parameter is a GLUT_KEY_* constant for the special key pressed. The x and y callback parameters indicate the mouse in window relative coordinates when the key was pressed. When a new window is created, no special callback is initially registered and special keystrokes in the window are ignored. Passing NULL to glutSpecialFunc disables the generation of special callbacks.

- **void glBlendFunc(GLenum sfactor, GLenum dfactor);**

glBlendFunc defines the operation of blending when it is enabled. sfactor specifies which method is used to scale the source color components. dfactor specifies which method is used to scale the destination color components. The possible methods are described in the following table. Each method defines four scale factors, one each for red, green, blue, and alpha.

- **void glutInitWindowSize(int width, int height);**

Specifies the initial height and width of the window in pixels.

4.2 USER DEFINED FUNCTIONS:

- **void renderBitmapString() function:**

To print the text such as game rules, score, game name. glutBitMapCharacter() function is used internally to print the text on output display window.

- **int main ( ) function:**

The main function is used for creating the window for display of the game "Road Rage". Here, we initialize the parameters of the display window. The callback functions, i.e., keyboard callback, display callback, etc are written in main. The callback functions registered in main ( ) are,

```
glutKeyboardFunc (mykeys);
glutDisplayFunc (display);
glutSpecialFunc (spec_keys);
glutTimerFunc (1000, timer);
```

- **void processKeys(unsigned char key, int x, int y)**

This function is used to process keys like 'space' which starts the game and to initialize parameters of the game.

- **void spe_key(int key, int x, int y)**

This function is used for controlling the car direction and speed using arrow keys.

- **void HomeMenu()**

This function is used to display the home page which specifies the rules of the game and how to start the game and if it is displaying the home page after the game ends then it displays the score also .

- **void startGame()**

This function is used for displaying the user car, opposite obstacle cars, then the coins to collect and also the highway road. Apart from that logic to compute score, level and user car color change after every level is also included.

5. TESTING

Testing process started with the testing of individual program units such as functions or objects. These were then integrated into sub-systems and systems, and interactions of these units were tested.

Testing involves verification and validation.

Validation: “Are we building right product?”

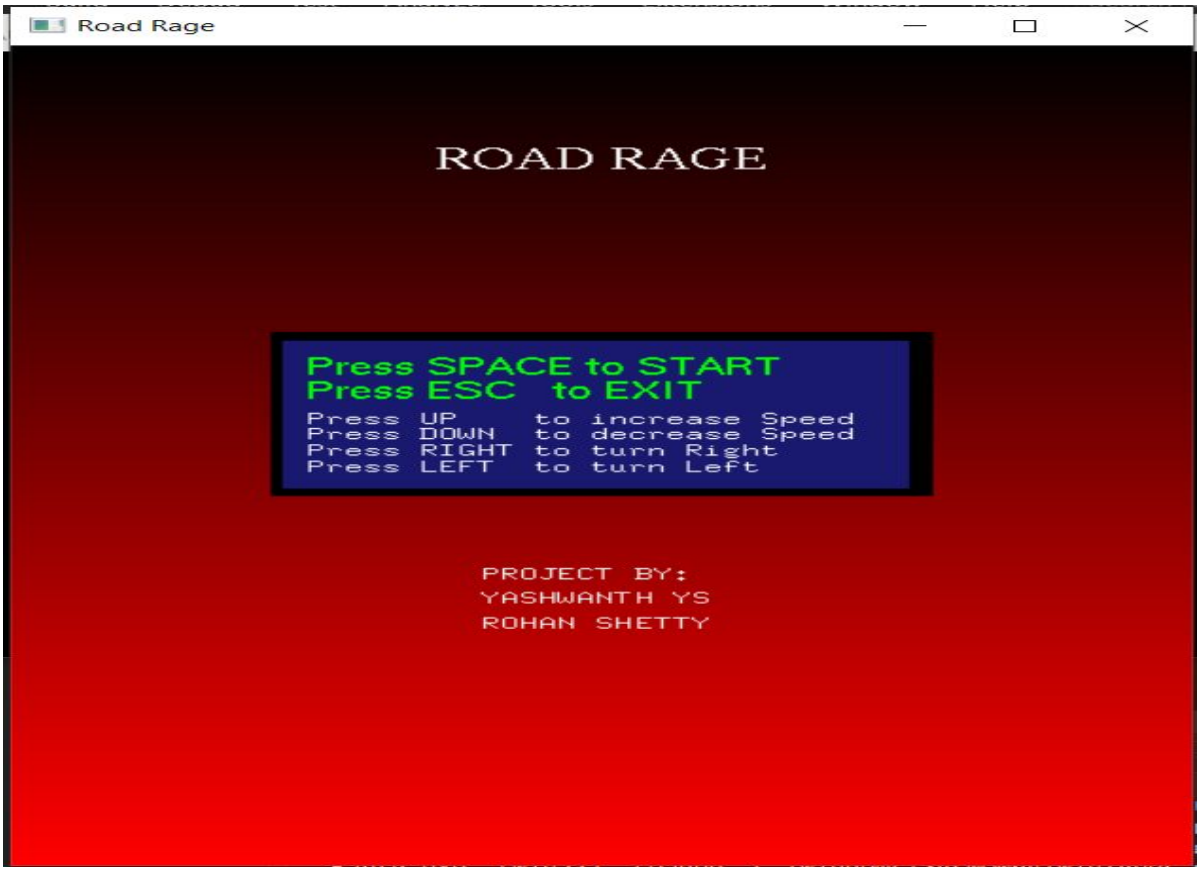
Verification: “Are we building the product right?”

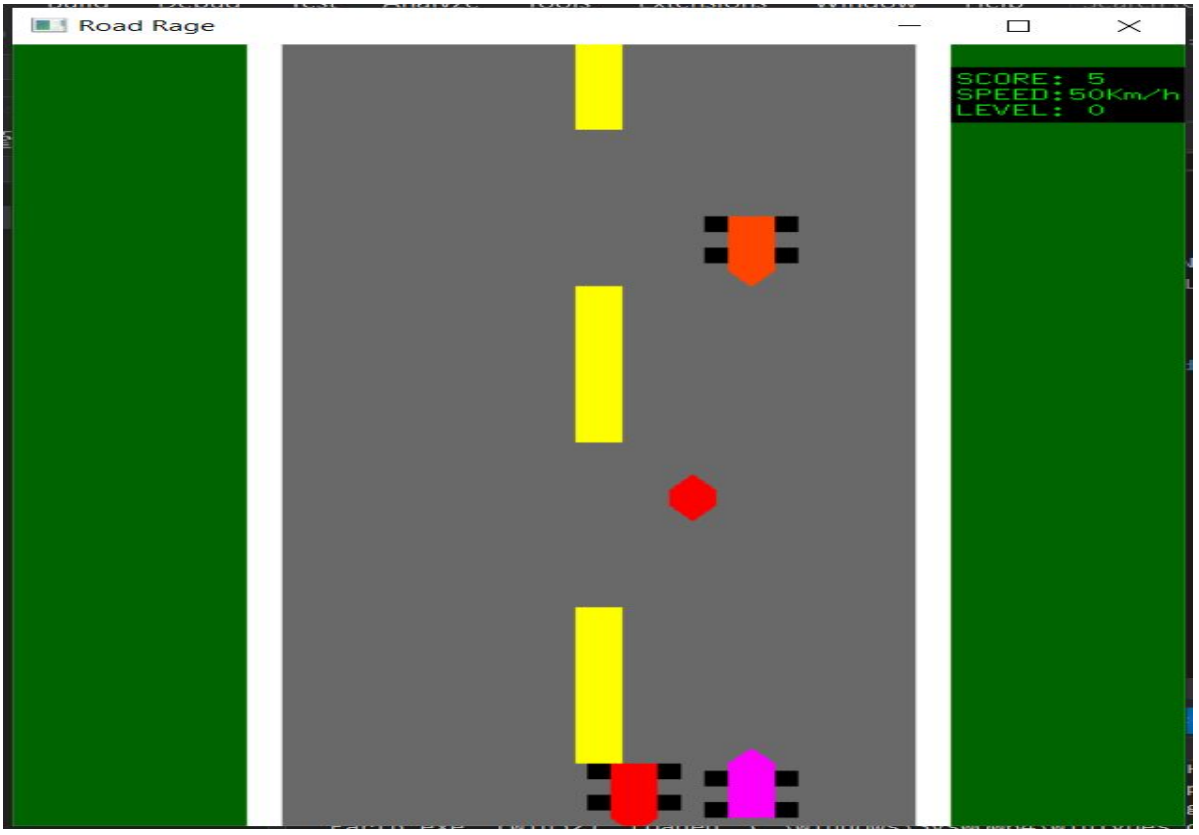
The ultimate goal of the verification and validation process is to establish confidence that the software system is ‘fit for purpose’. The level of required confidence depends on the system’s purpose, the expectations of the system users and the current marketing environment for the system.

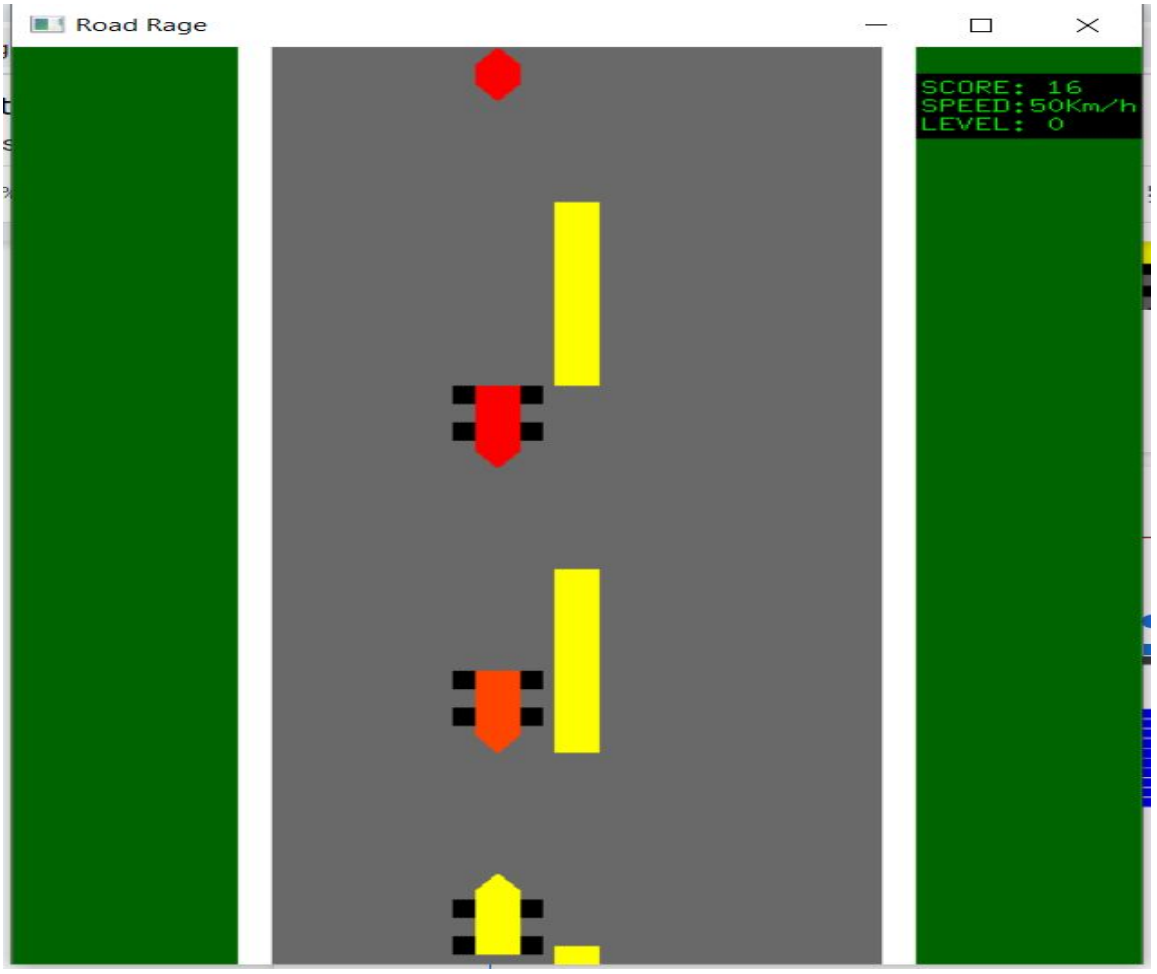
With the verification and validation process, there are two complementary approaches to the system checking and analysis:

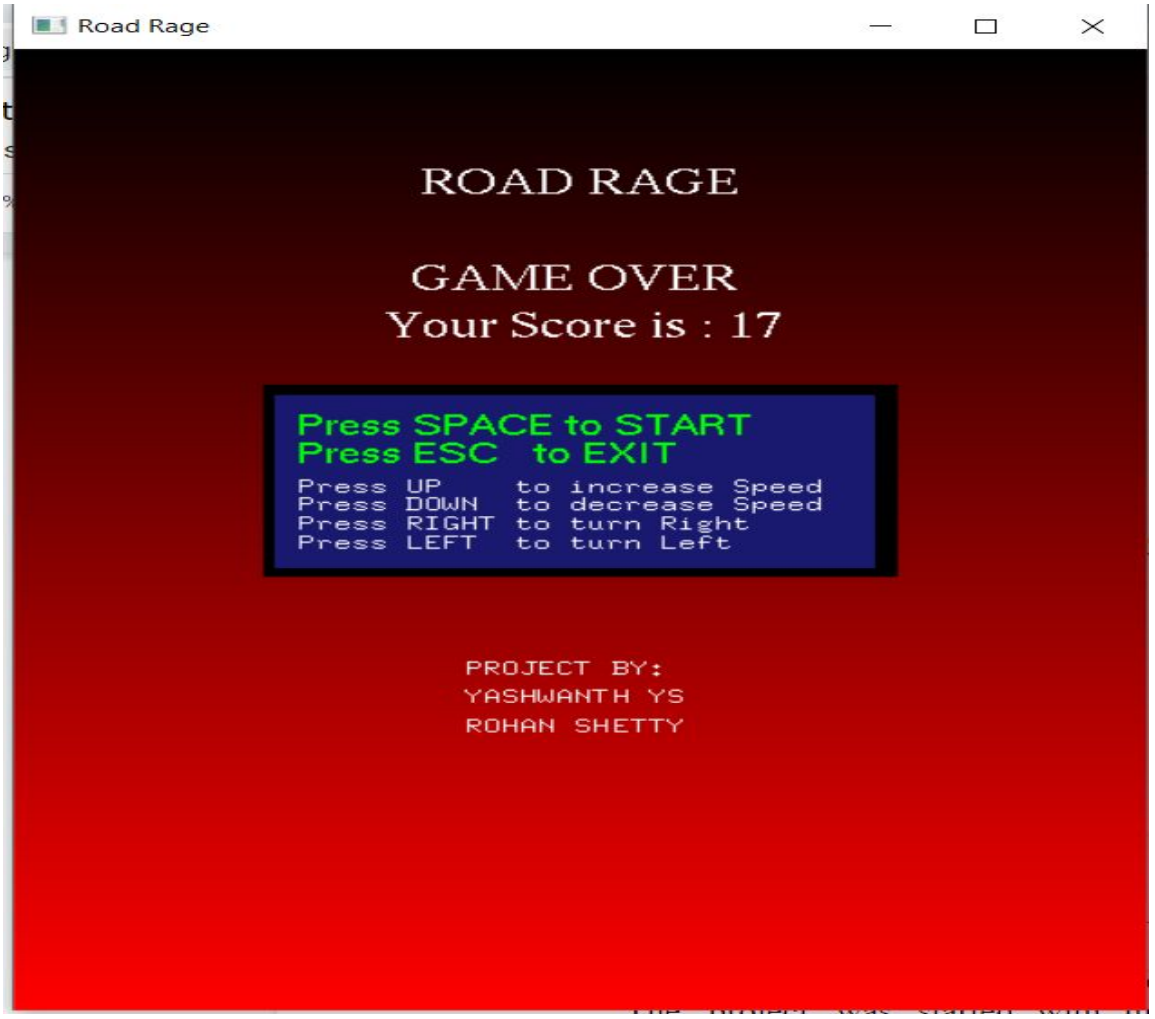
Software inspections or peer reviews analyses and check system representations such as the requirements document, design diagrams, and the program source code. Software testing involves running an implementation of the software with test data.

6. SNAPSHOTS


Road Rage
ROAD RAGE
Press SPACE to START
Press ESC to EXIT
Press UP to increase Speed
Press DOWN to decrease Speed
Press RIGHT to turn Right
Press LEFT to turn Left
PROJECT BY:
YASHWANTH YS
ROHAN SHETTY


Road Rage
SCORE: 5
SPEED:50Km/h
LEVEL: 0


Road Rage
SCORE: 16
SPEED:50Km/h
LEVEL: 0


Road Rage
ROAD RAGE
GAME OVER
Your Score is : 17
Press SPACE to START
Press ESC to EXIT
Press UP to increase Speed
Press DOWN to decrease Speed
Press RIGHT to turn Right
Press LEFT to turn Left
PROJECT BY:
YASHWANTH YS
ROHAN SHETTY

7. CONCLUSION AND ENHANCEMENTS

The project was started with modest aim with no prior experience in any programming projects as this, but ended up in learning many things, fine tuning the programming skills and getting into the real world of software development with an exposure to corporate environment. During the development of any software of significant utility, we are forced with the tradeoff between speed of execution and amount of memory consumed. This is simple interactive application. It is extremely user friendly and has the features, which makes simple graphics project. It has an open source and no security features has been included. The user is free to alter the code for feature enhancement. Checking and verification of all possible types of the functions are taken care. Care was taken to avoid bugs. Bugs may be reported to creator as the need.

Further this project can be enhanced by adding few by changing the scoring system. The color the obstacles can also be changed randomly. Further we can design a 3D game based on this concept. Also a bonus coin can be given randomly and if the user car, collects it, then the user car can apply nitros and the traffic is also cleared for few secs.

8. BIBLIOGRAPHY

Reference Books and E-books

1. Donald D. Hearn, M. Pauline Baker, Warren Carithers , Computer Graphics with OpenGL, 4th Edition, Publisher Pearson Education, November 2010, ISBN-13: 978- 0136053583

2. Edward Angel, Interactive Computer Graphics: A Top-Down Approach Using OpenGL, 5th Edition, Publisher Pearson Education, 2010, ISBN: 978131725306

3. Zhigang Xiang and Roy Plastock ,Computer Graphics 2nd Edition, ASIN: 0070601658, Publisher Tata McGraw-Hill, 2007, ISBN-13: 978-0070601659

4. www.opengl.org/resources/code/samples/redbook.

APPENDIX A – SOURCE CODE

```
#include<windows.h>
#ifdef _APPLE_
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <time.h>


double carred = 0.678, cargreen = 1.000, carblue = 0.184;

//Game Speed
int FPS = 50;
int prev = 0;

//Game Track
int start = 0;
int gv = 0;
int level = 0;
int coin = 0;

//Track Score
int score = 0;

//For road movement
int roadDivTopMost = 0;
int roadDivTop = 0;
int roadDivMdl = 0;
int roadDivBtm = 0;

//For Car Left / Right
int lrIndex = 0;

//Car Coming
int car1 = 0;
int lrIndex1 = 0;
int car2 = +35;
int lrIndex2 = 0;
int car3 = +70;
int lrIndex3 = 0;

//For Display TEXT
const int font1 = (int)GLUT_BITMAP_TIMES_ROMAN_24;
const int font2 = (int)GLUT_BITMAP_HELVETICA_18;
const int font3 = (int)GLUT_BITMAP_8_BY_13;

char s[30];
void renderBitmapString(float x, float y, void* font, const char* string) {
	const char* c;
	glRasterPos2f(x, y);
	for (c = string; *c != '\0'; c++) {
		glutBitmapCharacter(font, *c);
	}
}


void startGame() {
	//Road
	glColor3f(0.412, 0.412, 0.412);
	glBegin(GL_POLYGON);
	glVertex2f(20, 0);
	glVertex2f(20, 100);
```

```
glVertex2f(80, 100);
glVertex2f(80, 0);
glEnd();

//Road Left Border
glColor3f(1.000, 1.000, 1.000);
glBegin(GL_POLYGON);
glVertex2f(20, 0);
glVertex2f(20, 100);
glVertex2f(23, 100);
glVertex2f(23, 0);
glEnd();

//Road Right Border
glColor3f(1.000, 1.000, 1.000);
glBegin(GL_POLYGON);
glVertex2f(77, 0);
glVertex2f(77, 100);
glVertex2f(80, 100);
glVertex2f(80, 0);
glEnd();

//Road Middle White Blocks
//TOP
glColor3f(1.000, 1.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(48, roadDivTop + 80);
glVertex2f(48, roadDivTop + 100);
glVertex2f(52, roadDivTop + 100);
glVertex2f(52, roadDivTop + 80);
glEnd();

roadDivTop--;
if (roadDivTop < -100) {
	roadDivTop = 20;
	score++;
}

//Middle
glColor3f(1.000, 1.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(48, roadDivMdl + 40);
glVertex2f(48, roadDivMdl + 60);
glVertex2f(52, roadDivMdl + 60);
glVertex2f(52, roadDivMdl + 40);
glEnd();



roadDivMdl--;
if (roadDivMdl < -60) {
	roadDivMdl = 60;
	score++;
}

//Bottom
glColor3f(1.000, 1.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(48, roadDivBtm + 0);
glVertex2f(48, roadDivBtm + 20);
glVertex2f(52, roadDivBtm + 20);
glVertex2f(52, roadDivBtm + 0);
glEnd();

roadDivBtm--;
if (roadDivBtm < -20) {
	roadDivBtm = 100;
	score++;
}

//Score Board
glColor3f(0.000, 0.000, 0.000);
glBegin(GL_POLYGON);
```

```
glVertex2f(80, 97);
glVertex2f(100, 97);
glVertex2f(100, 98 - 8);
glVertex2f(80, 98 - 8);
glEnd();

//Print Score
char buffer[50];
sprintf_s(buffer, "SCORE: %d", score);
glColor3f(0.000, 1.000, 0.000);
renderBitmapString(80.5, 95, (void*)font3, buffer);

//Speed Print
char buffer1[50];
sprintf_s(buffer1, "SPEED:%dKm/h", FPS);
glColor3f(0.000, 1.000, 0.000);
renderBitmapString(80.5, 95 - 2, (void*)font3, buffer1);

//Level Print
if (score - prev >= 10) {
	prev = score;
	int last = score / 10;
	if (last != level) {
		level = score / 10;
		FPS = FPS + 2;

		srand(time(NULL));
		carred = rand() % 2;
		cargreen = rand() % 2;
		carblue = rand() % 2;

	}
}
char level_buffer[50];
sprintf_s(level_buffer, "LEVEL: %d", level);
glColor3f(0.000, 1.000, 0.000);
renderBitmapString(80.5, 95 - 4, (void*)font3, level_buffer);


//Increse Speed With level

//MAIN car
//Front Tire
glColor3f(0.000, 0.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(lrIndex + 26 - 2, 5);
glVertex2f(lrIndex + 26 - 2, 7);
glVertex2f(lrIndex + 30 + 2, 7);
glVertex2f(lrIndex + 30 + 2, 5);
glEnd();
//Back Tire
glColor3f(0.000, 0.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(lrIndex + 26 - 2, 1);
glVertex2f(lrIndex + 26 - 2, 3);
glVertex2f(lrIndex + 30 + 2, 3);
glVertex2f(lrIndex + 30 + 2, 1);
glEnd();
//Car Body

glColor3f(carred, cargreen, carblue);
glBegin(GL_POLYGON);
glVertex2f(lrIndex + 26, 1);
glVertex2f(lrIndex + 26, 8);

glColor3f(carred, cargreen, carblue);
glVertex2f(lrIndex + 28, 10);
glVertex2f(lrIndex + 30, 8);
glVertex2f(lrIndex + 30, 1);
glEnd();


//Opposite car 1
```

```
glColor3f(0.000, 0.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(lrIndex1 + 26 - 2, car1 + 100 - 4);
glVertex2f(lrIndex1 + 26 - 2, car1 + 100 - 6);
glVertex2f(lrIndex1 + 30 + 2, car1 + 100 - 6);
glVertex2f(lrIndex1 + 30 + 2, car1 + 100 - 4);
glEnd();
glColor3f(0.000, 0.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(lrIndex1 + 26 - 2, car1 + 100);
glVertex2f(lrIndex1 + 26 - 2, car1 + 100 - 2);
glVertex2f(lrIndex1 + 30 + 2, car1 + 100 - 2);
glVertex2f(lrIndex1 + 30 + 2, car1 + 100);
glEnd();
glColor3f(1.000, 0.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(lrIndex1 + 26, car1 + 100);
glVertex2f(lrIndex1 + 26, car1 + 100 - 7);
glVertex2f(lrIndex1 + 28, car1 + 100 - 9);
glVertex2f(lrIndex1 + 30, car1 + 100 - 7);
glVertex2f(lrIndex1 + 30, car1 + 100);
glEnd();
car1--;
if (car1 < -100) {
        car1 = 0;
        lrIndex1 = lrIndex;
}
//KIll check car1
if ((abs(lrIndex - lrIndex1) < 8) && (car1 + 100 < 10)) {
        start = 0;
        gv = 1;

}

//Opposite car 2
/*glColor3f(0.000, 0.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(lrIndex2 + 26 - 2, car2 + 100 - 4);
glVertex2f(lrIndex2 + 26 - 2, car2 + 100 - 6);
glVertex2f(lrIndex2 + 30 + 2, car2 + 100 - 6);
glVertex2f(lrIndex2 + 30 + 2, car2 + 100 - 4);
glEnd();
glColor3f(0.000, 0.000, 0.000);
glBegin(GL_POLYGON);
glVertex2f(lrIndex2 + 26 - 2, car2 + 100);
glVertex2f(lrIndex2 + 26 - 2, car2 + 100 - 2);
glVertex2f(lrIndex2 + 30 + 2, car2 + 100 - 2);
glVertex2f(lrIndex2 + 30 + 2, car2 + 100);
glEnd();*/
if (coin == 0)
        glColor3f(1, 0, 0);
else {
        glColor4f(0, 0, 0, 0);
        coin = 0;
}
glBegin(GL_POLYGON);
glVertex2f(lrIndex2 + 28, car2 + 100 + 2);
glVertex2f(lrIndex2 + 26, car2 + 100);
glVertex2f(lrIndex2 + 26, car2 + 100 - 2);
glVertex2f(lrIndex2 + 28, car2 + 100 - 4);
glVertex2f(lrIndex2 + 30, car2 + 100 - 2);
glVertex2f(lrIndex2 + 30, car2 + 100);
glEnd();
car2--;
if (car2 < -100) {
        car2 = 0;
        lrIndex2 = lrIndex;
}
//KIll check car2
if ((abs(lrIndex - lrIndex2) < 8) && (car2 + 100 < 10)) {
        /* start = 0; */
                if (car2 + 100 > 5)
                        score += 1;
```

```
			coin = 1;
		}


		//Opposite car 3
		glColor3f(0.000, 0.000, 0.000);
		glBegin(GL_POLYGON);
		glVertex2f(lrIndex3 + 26 - 2, car3 + 100 - 4);
		glVertex2f(lrIndex3 + 26 - 2, car3 + 100 - 6);
		glVertex2f(lrIndex3 + 30 + 2, car3 + 100 - 6);
		glVertex2f(lrIndex3 + 30 + 2, car3 + 100 - 4);
		glEnd();
		glColor3f(0.000, 0.000, 0.000);
		glBegin(GL_POLYGON);
		glVertex2f(lrIndex3 + 26 - 2, car3 + 100);
		glVertex2f(lrIndex3 + 26 - 2, car3 + 100 - 2);
		glVertex2f(lrIndex3 + 30 + 2, car3 + 100 - 2);
		glVertex2f(lrIndex3 + 30 + 2, car3 + 100);
		glEnd();
		glColor3f(1.000, 0.271, 0.000);
		glBegin(GL_POLYGON);
		glVertex2f(lrIndex3 + 26, car3 + 100);
		glVertex2f(lrIndex3 + 26, car3 + 100 - 7);
		glVertex2f(lrIndex3 + 28, car3 + 100 - 9);
		glVertex2f(lrIndex3 + 30, car3 + 100 - 7);
		glVertex2f(lrIndex3 + 30, car3 + 100);
		glEnd();
		car3--;
		if (car3 < -100) {
			car3 = 0;
			lrIndex3 = lrIndex;
		}
		//KIll check car3
		if ((abs(lrIndex - lrIndex3) < 8) && (car3 + 100 < 10)) {
			start = 0;
			gv = 1;

		}

}

void homeMenu() {

		//Road Sky
		glColor3f(0, 0, 0);
		glBegin(GL_POLYGON);
		glVertex2f(100, 100);
		glVertex2f(0, 100);
		glColor3f(1, 0, 0);
		glVertex2f(0, 0);
		glVertex2f(100, 0);
		glEnd();


		//Menu Place Holder
		glColor3f(0.098, 0.098, 0.439);
		glBegin(GL_POLYGON);
		glVertex2f(32 - 10, 50 + 5 + 10);
		glVertex2f(32 + 45, 50 + 5 + 10);
		glVertex2f(32 + 45, 50 - 15 + 10);
		glVertex2f(32 - 10, 50 - 15 + 10);
		glEnd();

		glColor3f(0, 0, 0);
		glBegin(GL_POLYGON);
		glVertex2f(32 - 10, 50 + 5 + 10);
		glVertex2f(32 + 45, 50 + 5 + 10);
		glVertex2f(32 + 45, 50 + 4 + 10);
		glVertex2f(32 - 10, 50 + 4 + 10);
		glEnd();
		glBegin(GL_POLYGON);
		glVertex2f(32 + 44, 50 + 5 + 10);
		glVertex2f(32 + 46, 50 + 5 + 10);
```

```
		glVertex2f(32 + 46, 50 - 15 + 10);
		glVertex2f(32 + 44, 50 - 15 + 10);
		glEnd();
		glBegin(GL_POLYGON);
		glVertex2f(32 - 10, 50 - 14 + 10);
		glVertex2f(32 + 46, 50 - 14 + 10);
		glVertex2f(32 + 46, 50 - 15 + 10);
		glVertex2f(32 - 10, 50 - 15 + 10);
		glEnd();
		glBegin(GL_POLYGON);
		glVertex2f(32 - 10, 50 + 5 + 10);
		glVertex2f(32 - 9, 50 + 5 + 10);
		glVertex2f(32 - 9, 50 - 15 + 10);
		glVertex2f(32 - 10, 50 - 15 + 10);
		glEnd();



		//Text Information in First Page
		if (gv == 1) {
			glColor3f(1.000, 1.000, 10.000);
			renderBitmapString(35, 75, (void*)font1, "GAME OVER");
			glColor3f(1.000, 1.000, 1.000);

			char buffer2[50];
			sprintf_s(buffer2, "Your Score is : %d", score);
			renderBitmapString(33, 70, (void*)font1, buffer2);




		}

		glColor3f(1.000, 1.000, 1.000);
		renderBitmapString(36, 85, (void*)font1, "ROAD RAGE");

		glColor3f(0.000, 1.000, 0.000);
		renderBitmapString(30 - 5, 50 + 10, (void*)font2, "Press SPACE to START");
		renderBitmapString(30 - 5, 50 - 3 + 10, (void*)font2, "Press ESC   to EXIT");

		glColor3f(1.000, 1.000, 1.000);
		renderBitmapString(30 - 5, 50 - 6 + 10, (void*)font3, "Press UP    to increase Speed");
		renderBitmapString(30 - 5, 50 - 8 + 10, (void*)font3, "Press DOWN  to decrease Speed");
		renderBitmapString(30 - 5, 50 - 10 + 10, (void*)font3, "Press RIGHT to turn Right");
		renderBitmapString(30 - 5, 50 - 12 + 10, (void*)font3, "Press LEFT  to turn Left");


		glColor3f(1, 1, 1);
		renderBitmapString(40, 35, (void*)font3, "PROJECT BY:");
		renderBitmapString(40, 32, (void*)font3, "YASHWANTH YS");
		renderBitmapString(40, 29, (void*)font3, "ROHAN SHETTY");

}


void display() {
		glClear(GL_COLOR_BUFFER_BIT);

		if (start == 1) {
			// glClearColor(0.627, 0.322, 0.176,1);

			glClearColor(0.000, 0.392, 0.000, 1);
			startGame();
		}

		else {
			homeMenu();
			//glClearColor(0.184, 0.310, 0.310,1);
		}



		glFlush();
```

```
	glutSwapBuffers();
}



void spe_key(int key, int x, int y) {
	switch (key) {
	case GLUT_KEY_DOWN:
		if (FPS > (50 + (level * 2)))
			FPS = FPS - 2;
		break;
	case GLUT_KEY_UP:
		FPS = FPS + 2;
		break;

	case GLUT_KEY_LEFT:
		if (lrIndex >= 0) {
			lrIndex = lrIndex - (FPS / 10);
			if (lrIndex < 0) {
				lrIndex = -1;
			}
		}
		break;


	case GLUT_KEY_RIGHT:
		if (lrIndex <= 44) {
			lrIndex = lrIndex + (FPS / 10);
			if (lrIndex > 44) {
				lrIndex = 45;
			}
		}
		break;

	default:
		break;
	}

}

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

	switch (key)
	{
	case ' ':
		if (start == 0) {
			start = 1;
			gv = 0;
			FPS = 50;
			roadDivTopMost = 0;
			roadDivTop = 0;
			roadDivMdl = 0;
			roadDivBtm = 0;
			lrIndex = 0;
			car1 = 0;
			lrIndex1 = 0;
			car2 = +35;
			lrIndex2 = 0;
			car3 = +70;
			lrIndex3 = 0;
			score = 0;
			level = 0;
		}
		break;

	case 27:
		exit(0);
		break;
	default:
		break;
	}
}
```

```
void timer(int) {
	glutPostRedisplay();
	glutTimerFunc(1000 / FPS, timer, 0);
}



int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
	glutInitWindowSize(500, 650);
	glutInitWindowPosition(200, 20);
	glutCreateWindow("Road Rage");

	glutDisplayFunc(display);
	glutSpecialFunc(spe_key);
	glutKeyboardFunc(processKeys);

	glOrtho(0, 100, 0, 100, -1, 1);
	glClearColor(0.184, 0.310, 0.310, 1);
	glEnable(GL_BLEND);
	glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
	glutTimerFunc(1000, timer, 0);
	glutMainLoop();

	return 0;
}
```