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

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

"2D Dodge Car Game"

COMPUTER GRAPHICS LAB (16CS73)

OPEN ENDED EXPERIMENT REPORT

VII SEMESTER

2020-2021

Submitted by

Satya Sujan C 1RV17CS142
Sanjaya S Hegde 1RV17CS139

Under the Guidance of
Prof. Mamatha T
Assistant Professor
Department of CSE
RV College of Engineering, Bangalore

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 "2D Dodge Car Game" has been carried out by Satya Sujan C(1RV17CS142) and Sanjaya S Hegde(1RV17CS139)**,** 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.

Prof. Mamatha T,
Assistant Professor,
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, **Satya Sujan C(1RV17CS142)** and **Sanjaya S Hegde(1RV17CS139)** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled **"2D Dodge Car 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 **Satya Sujan C**

Date: **Sanjaya S Hegde**

ACKNOWLEDGEMENT

We take this opportunity to express our profound gratitude and deep regards to our guide Prof. Mamatha T for her exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by her from time to time shall carry us a long way in the journey of life on which we are about to embark.

We also take this opportunity to express a deep sense of gratitude to Dr.Ramakanth Kumar P, HoD, Department of Computer science and Engineering, R V College of Engineering ,for his valuable information and guidance, which helped us in completing this task through various stages.

We are also obliged to thank my peers, for the valuable suggestions provided by them in their respective fields. We are grateful for their cooperation during the period of our assignment.

Lastly, Our parents, siblings and friends for their constant encouragement without which this assignment would not be possible.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Computer Graphics

To draw a picture, say, fish moving inside the water. Suddenly I will get an idea to use paint, but the degree of accuracy, quality of image is not satisfied and I become very sad. There is no need to worry, for every problem there will be a solution, so this problem of creating fish moving inside the water can be solved using COMPUTER GRAPHICS without any difficulties.

Computer Graphics became a powerful tool for the rapid and economical production of pictures. There is virtually no area in which Graphical displays cannot be used to some advantage so it is not surprising to find the use of CG so widespread.

Although early applications in engineering & science had to rely on expensive & cumbersome equipment, advances in computer technology have made interactive computer graphics a practical tool.

Computer Graphics in a diverse area such as science, engineering, medicine, business, industry, government, art, entertainment, education and training.

Now it can be answered about computer graphics as a generalized tool for drawing and creating pictures and simulates the real world situations within a small computer window.

1.2 OpenGL

1.2.1 OpenGL Graphics Architecture

Most of the application will be designed to access OpenGL directly through functions in three libraries. Functions in the main GL (or OpenGL in windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in windows).

The second is the **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 begin with letters glu.

To interface with the window system and to get input from external devices into the programs, you need at least one more system-specific library that provides the "glue" between the window system and OpenGL. For the X window system, this library is functionality that should be expected in any modern windowing system.

Fig 2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.

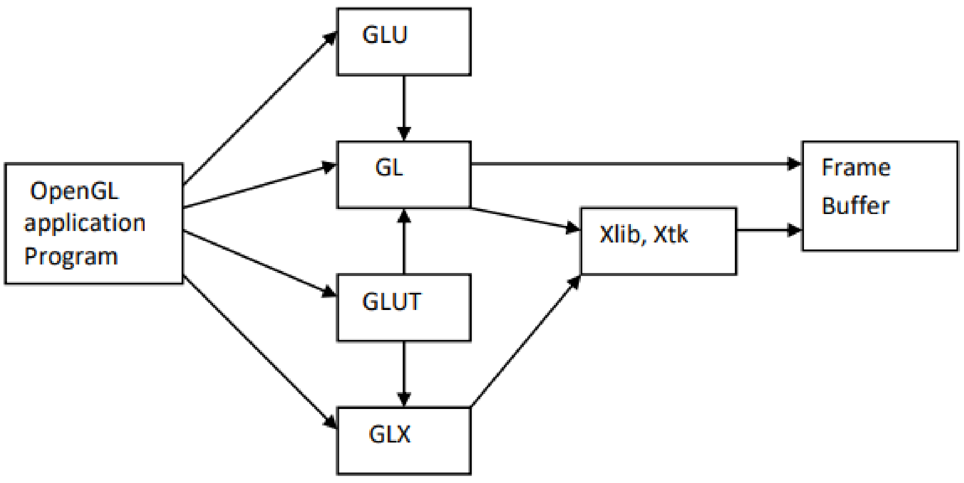


Fig 1. Library organization of OpenGL

Libraries

1. The **OpenGL Utility Library** (**GLU**) is a computer graphics library for OpenGL.

It consists of a number of functions that use the base OpenGL library to provide higher-level drawing routines from the more primitive routines that OpenGL provides. It is usually distributed with the base OpenGL package. GLU is not implemented in the embedded version of the OpenGL package, OpenGL ES.

Among these features are mapping between screen- and world-coordinates, generation of texture mipmaps, drawing of quadric surfaces, NURBS, tessellation of polygon primitives, interpretation of OpenGL error codes, an extended range of transformation routines for setting up viewing volumes and simple positioning of the camera, generally in more human
friendly terms than the routines presented by OpenGL. It also provides additional primitives for use in OpenGL applications, including spheres, cylinders and disks.

All GLU functions start with the glu prefix. An example function is gluOrtho2D which defines a two dimensional orthographic projection matrix.

2. The **OpenGL Utility Toolkit** (**GLUT**) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres and the Utah teapot. GLUT also has some limited support for creating pop-up menus.

GLUT was written by Mark J. Kilgard, author of *OpenGL Programming for the X Window System* and *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*, while he was working for Silicon Graphics Inc.

The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross-platform) and to make learning OpenGL easier. Getting started with OpenGL programming while using GLUT often takes only a few lines of code and does not require knowledge of operating system–specific windowing APIs.

3. Core OpenGL (GL): consists of hundreds of functions, which begin with a prefix "gl" (e.g., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives, such as point, line, and polygon.

4. **OpenGL Extension Wrangler Library (GLEW)**: "GLEW is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform.

Each of the software package consists of:

1. **A *header* file**: "gl.h" for core OpenGL, "glu.h" for GLU, and "glut.h" (or "freeglut.h") for GLUT, typically kept under "include\GL" directory.
2. **A *static library***: for example, in Win32, "libopengl32.a" for core OpenGL, "libglu32.a" for GLU, "libglut32.a" (or "libfreeglut.a" or "glut32.lib") for GLUT, typically kept under "lib" directory.
3. **An optional *shared library*:** for example, "glut32.dll" (for "freeglut.dll") for GLUT under Win32, typically kept under "bin" or "c:\windows\system32".

It is important to locate the *directory path* and the *actual filename* of these header files and libraries in your operating platform in order to properly setup the OpenGL programming environment.

1.2.2 Primitives and Attributes

1. Line Primitives

GL_LINES: Vertices 0 and 1 are considered a line. Vertices 2 and 3 are considered a line. And so on. If the user specifies a non-even number of vertices, then the extra vertex is ignored.

GL_LINE_STRIP: The adjacent vertices are considered lines. Thus, if you pass n vertices, you will get n-1 lines. If the user only specifies 1 vertex, the drawing command is ignored.

GL_LINE_LOOP: As line strips, except that the first and last vertices are also used as a line. Thus, you get n lines for n input vertices. If the user only specifies 1 vertex, the drawing command is ignored. The line between the first and last vertices happens after all of the previous lines in the sequence.

2. Triangle Primitives

GL_TRIANGLES: Vertices 0, 1, and 2 form a triangle. Vertices 3, 4, and 5 form a triangle. And so on.

GL_TRIANGLE_STRIP: Every group of 3 adjacent vertices forms a triangle. The face direction of the strip is determined by the winding of the first triangle. Each successive triangle will have its effective face order reversed, so the system compensates for that by testing it in the opposite way. A vertex stream of *n* length will generate *n*-2 triangles.

GL_TRIANGLE_FAN: The first vertex is always held fixed. From there on, every group of 2 adjacent vertices form a triangle with the first. So with a vertex stream, you get a list of triangles like so: (0, 1, 2) (0, 2, 3), (0, 3, 4), etc. A vertex stream of *n* length will generate *n*-2 triangles.

3. Quad Primitives

GL_QUADS: Vertices 0-3 form a quad, vertices 4-7 form another, and so on. The vertex stream must be a number of vertices divisible by 4 to work.

GL_QUAD_STRIP: Similar to triangle strips, a quad strip uses adjacent edges to form the next quad. In the case of quads, the third and fourth vertices of one quad are used as the edge of the next quad. So vertices 0-3 are a quad, 2-5 are a quad, and so on. A vertex stream of *n* length will generate (*n* - 2) / 2 quads. As with triangle strips, the winding order of quads is changed for every other quad.

4. Attributes

void glColour3f(GLdouble red, GLdouble green, GLdouble blue) : Specify new red, green, and blue values for the current color.

glClearColor(): Specifies a new alpha value for the current color. Included only in the four argument glColor4 commands.

1.2.3 Color, Viewing and Control Function

- **glMatrixMode():** Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The initial value is GL_MODELVIEW. Additionally, if the ARB_imaging extension is supported, GL_COLOR is also accepted.
- **glColor3f():** Sets the current color.
- **glutInit():** allows applications to get command line arguments and initializes system
- **glutInitDisplayMode():** requests properties for the window (the rendering context)
  - o RGB color.
  - o Single buffering
  - o Properties logically ORed together
- **glutWindowSize():** in pixels glutWindowPosition from top-left corner of display
- **glutCreateWindow():** create window with a particular title

1.3 Proposed System

1.3.1 Objective of the project

The Objective of the project titled “2D Dodge car game” is to develop a simple, easy to-play Two dimensional car video game using OpenGL. The player has to dodge cars that he/she comes across in different lanes at regular intervals and survive for as long as possible.
The Game is over when the player’s car collides with one of the others.

1.3.2 Methodology

The game screen contains multiple tracks with opposing cars moving on them periodically. The objective of the game is to survive for as long as possible by dodging the incoming cars. The movement of opposing cars is in fact facilitated by downward motion of the road dividers and cars are in fact stationary. Similarly the player’s car in reality moves along the horizontal axis as all vertical motion is of the road through its dividers. The left key button is used for movement of the player’s car along the negative X-axis, The right key button is used for movement of the player’s car along the positive

X-axis , The up key button is used to increase the vertical speed of the player's car until an upper limit as governed by the level i.e the speed with dividers move and opposing cars come, the down key button is used to decrease the vertical speed of the player's car i.e the speed with which dividers move and opposing cars come upto a lower limit. The score is calculated for every divider passed successfully by dodging incoming cars i.e. without colliding with them. For every 50 points, a new level starts and base the speed is incremented therefore increasing the difficulty with each passing level.

Arrow Key Up – Increase Vertical Speed
Arrow Key Down – Decrease Vertical Speed
Arrow Key Left – Move Left
Arrow Key Right – Move Right
ESCAPE - Quit Game

1.3.3 Scope

It is the basic implementation of the classic game "2D Dodge Car". It is created by using minimum classes and functions. OpenGL has been used effectively for all the animation without the use of any additional libraries. This allows for the creation and implementation of many other games using only OpenGL. The simplicity of OpenGL makes the game very light and convenient to run even on low specification hardware. The Game alone has lots of room for improvement. The future versions of the game can enable the player to choose the scenery i.e day, night, sunset, desert, ocean highway etc. The graphics of the game also have high scope for improvement including constructing a 3d version with multiple views.

Chapter 2

Requirements Specification

2.1 Hardware Requirements

The Hardware requirements are very minimal and the program can be run on most of the machines.

- **Processor** - Intel 486/Pentium or Above
- **Processor Speed** - 500MHz or above
- **Hard Disk** - 20GB
- **RAM** - 64MB or above
- **Storage Space** - Approx 2MB
- **Keyboard**

2.2 Software Requirements

- **Operating System** - Windows 7/8/10
- **Development tool -** Microsoft Visual Studio 2019
- **Graphics Package -** OpenGL

Chapter 3

System Design and Implementation

3.1 Flow Diagram

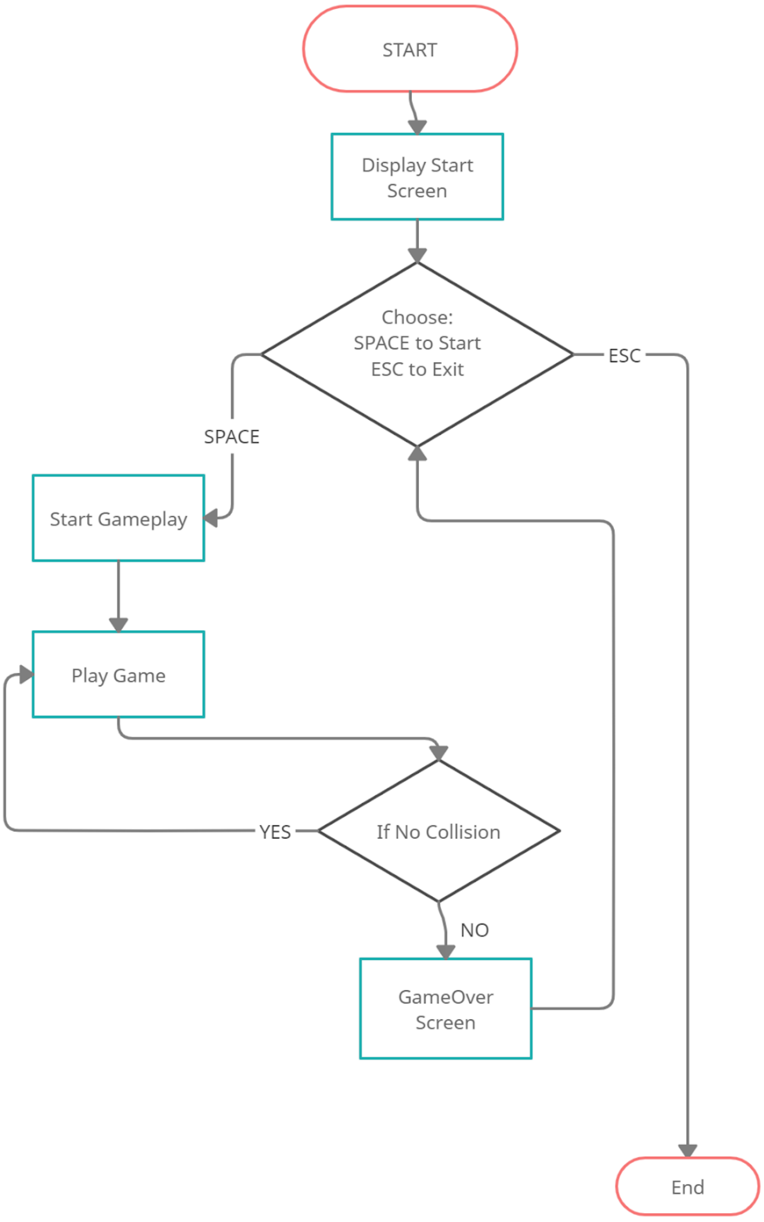


Fig 2 - Flow Diagram of the Game

3.2 Structure Chart

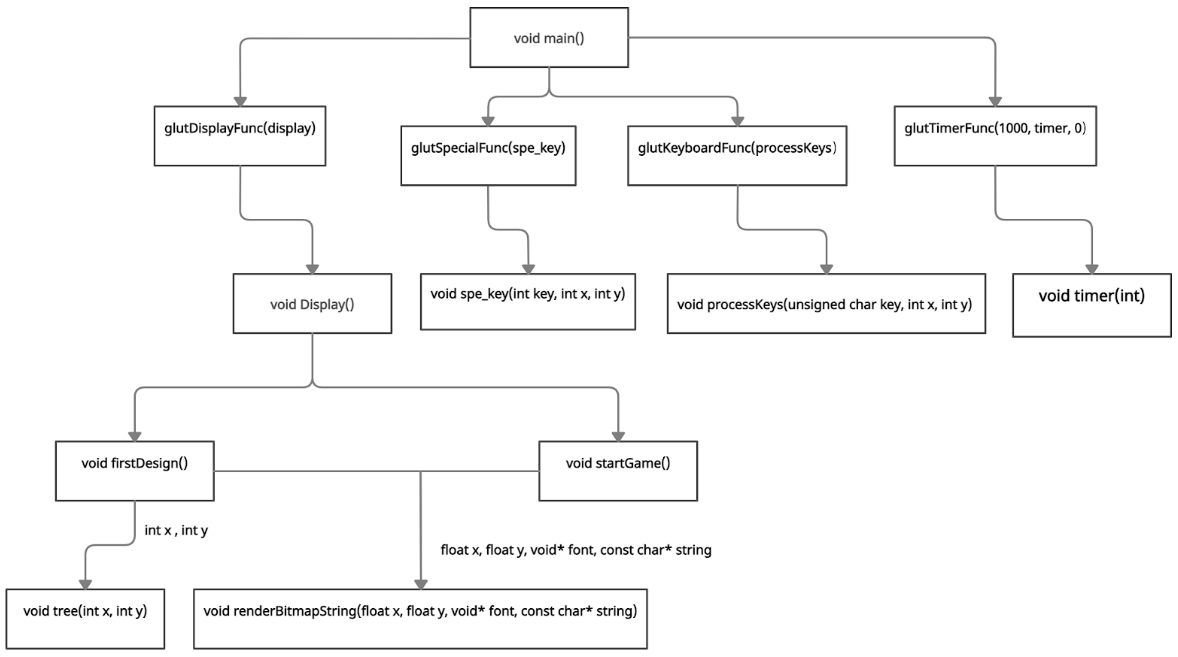

void main()
glutDisplayFunc(display)
glutSpecialFunc(spe_key)
glutKeyboardFunc(processKeys)
glutTimerFunc(1000, timer, 0)
void Display()
void spe_key(int key, int x, int y)
void processKeys(unsigned char key, int x, int y)
void timer(int)
void firstDesign()
void startGame()
int x , int y
float x, float y, void* font, const char* string
void tree(int x, int y)
void renderBitmapString(float x, float y, void* font, const char* string)

Fig 3 - Structure Chart of the Game

3.3 Modular Description

- **void tree(int x, int y) -** To insert a coniferous tree onto the scene
- **void renderBitmapString(float x, float y, void* font, const char* string) -** Inserts a String text
- **void display()** - Display Function
- **void firstDesign() -** Maps the layout of the Start/Game Over Screen
- **void startGame() -** Starts the Gameplay and Calculates Score
- **void timer(int) -** Calculates gameplay time
- **void processKeys(unsigned char key, int x, int y) -** Plays Background Music
- **void spe_key(int key, int x, int y) -** Controls car movements using Keyboard Keys

OpenGL Functions

- **void glBegin(glEnum Mode) -** Initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES and GL_POLYGON.
- **void glEnd( ) -** Terminates a list of vertices.
- **void glColor3f[i f d] (TYPE r, TYPE g, TYPE b) -** Sets the present RGB colors. Valid types are int (i), float (f) and double (d). The maximum and minimum values of the floating-point types are 1.0 and 0.0, respectively.
- **void glClearColor(GLclampf r,GLclampf g,GLclampf b,GLclampf a); -** Sets the present RGBA clear color used when clearing the color buffer. Variables of GLclampf are floating-point numbers between 0.0 and 1.0.
- **int glutCreateWindow(char *title);**- Creates a window on the display. The string title can be used to label the window. The return value provides a reference to the window that can be used where there are multiple windows.
- **void glutInitWindowSize(int width, int height); -** Specifies the initial height and width of the window in pixels

- **void glutInitWindowPosition(int x, int y);** Specifies the initial position of the top-left corner of the window in pixels.

- **void glutInitDisplayMode(unsigned int mode); -** Request a display with the properties in mode. The value of mode is determined by the logical OR of operation including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE);

- **void glFlush( ); -** Forces any buffered OpenGL commands to execute.

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

- **void glutMainLoop( ); -** Cause the program to enter an event processing loop. It should be the last statement in main.

- **void glutDisplayFunc(void (*func) (void));** Registers the display function func that is executed when the window needs to be redrawn.

- **gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);** Defines a two-dimensional viewing rectangle in the plane Z=0;

- **void glutBitmapCharacter(void *font, int char); -** Renders the character with ASCII code char at the current raster position using the raster font given by font. Fonts include GLUT_BITMAP_TIMES_ROMAN_10 and GLUT_BITMAP_TIMES_ROMAN_8_Y_13. The raster position is incremented by the width of the character.

- **void glClear(GL_COLOR_BUFFER_BIT); -** To make the screen solid and white.

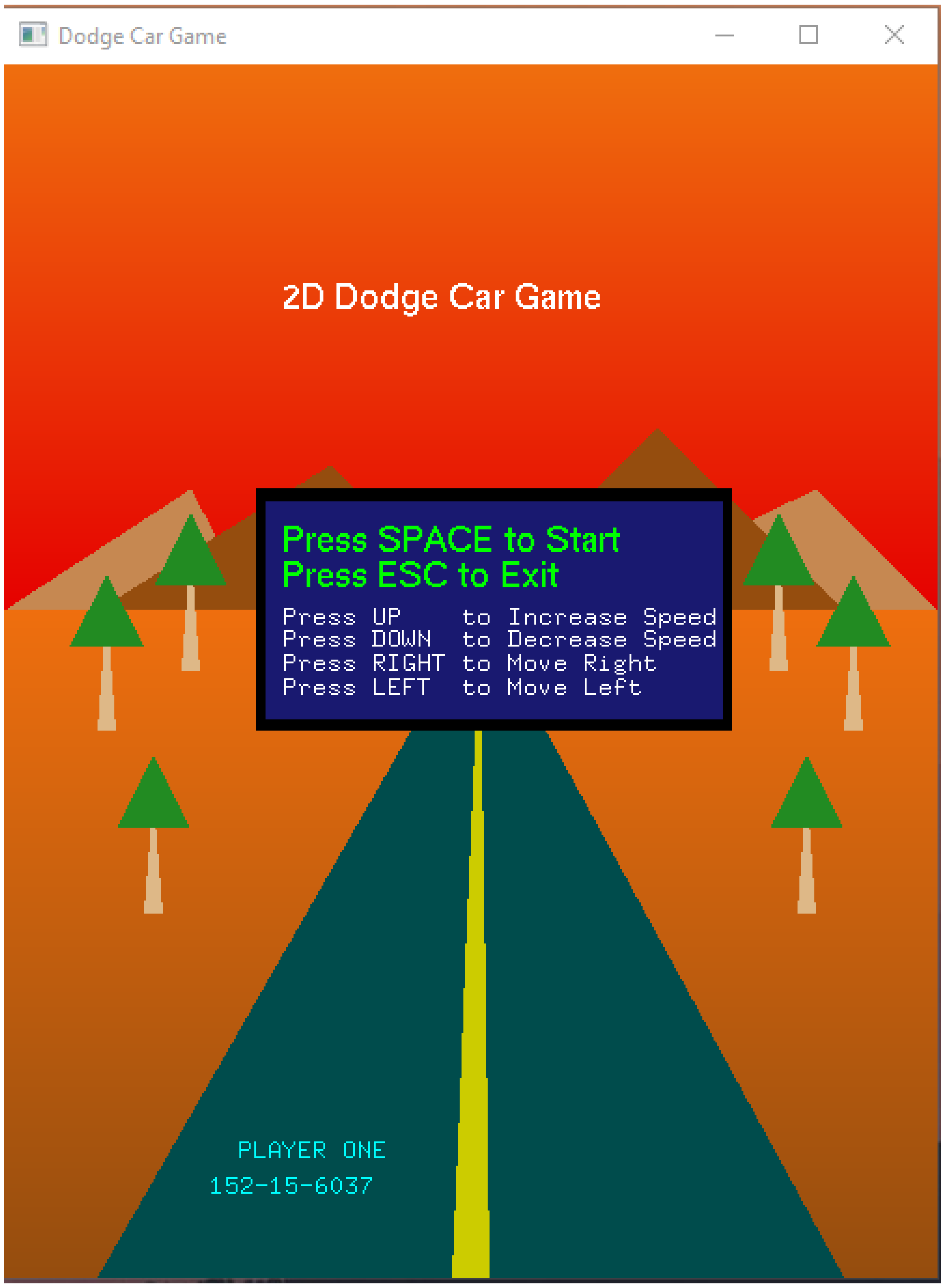
- **void glutSpecialFunc(void (*func)(int key, int x, int y)); -** It is used for the implementation of keyboard interface. glutSpecialFunc sets the special keyboard callback for the current window. Passing control to **void spe_key(int key, int x, int y)**

- **void glutTimerFunc(unsigned int msecs, void (*func)(int value), value); -** glutTimerFunc registers a timer callback to be triggered in a specified number of milliseconds.

- **void glutPostRedisplay(); -** Re-render the scene again as and when the input is added dynamically.

Chapter 4

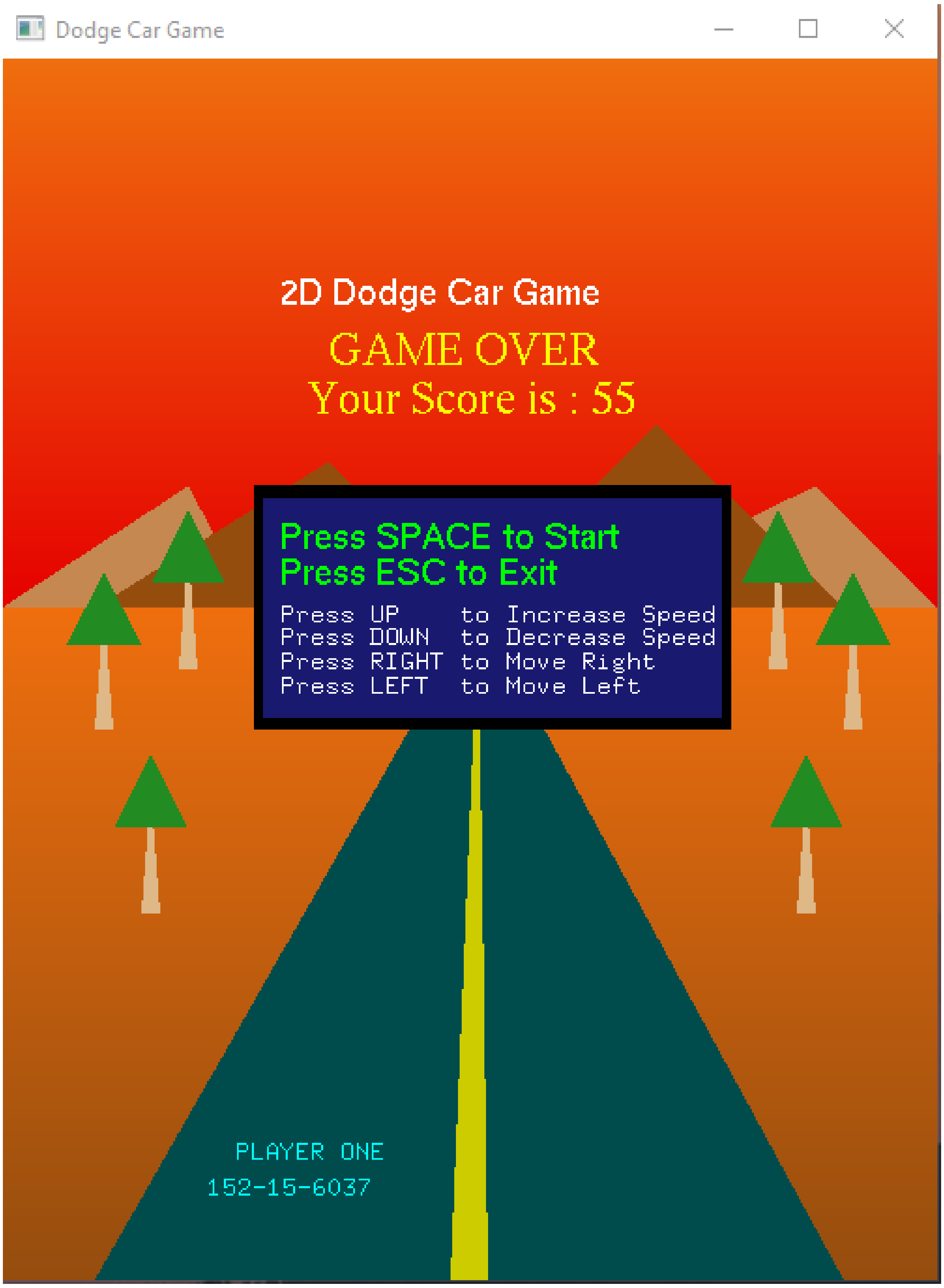
Results and Snapshots



```
Press SPACE to Start
Press ESC to Exit

Press UP    to Increase Speed
Press DOWN  to Decrease Speed
Press RIGHT to Move Right
Press LEFT  to Move Left
```

Fig 4 - Home Screen



2D Dodge Car Game

GAME OVER
Your Score is : 55

Press SPACE to Start
Press ESC to Exit

```
Press UP    to Increase Speed
Press DOWN  to Decrease Speed
Press RIGHT to Move Right
Press LEFT  to Move Left
```

PLAYER ONE
152-15-6037

Fig 5 - Game Over Screen

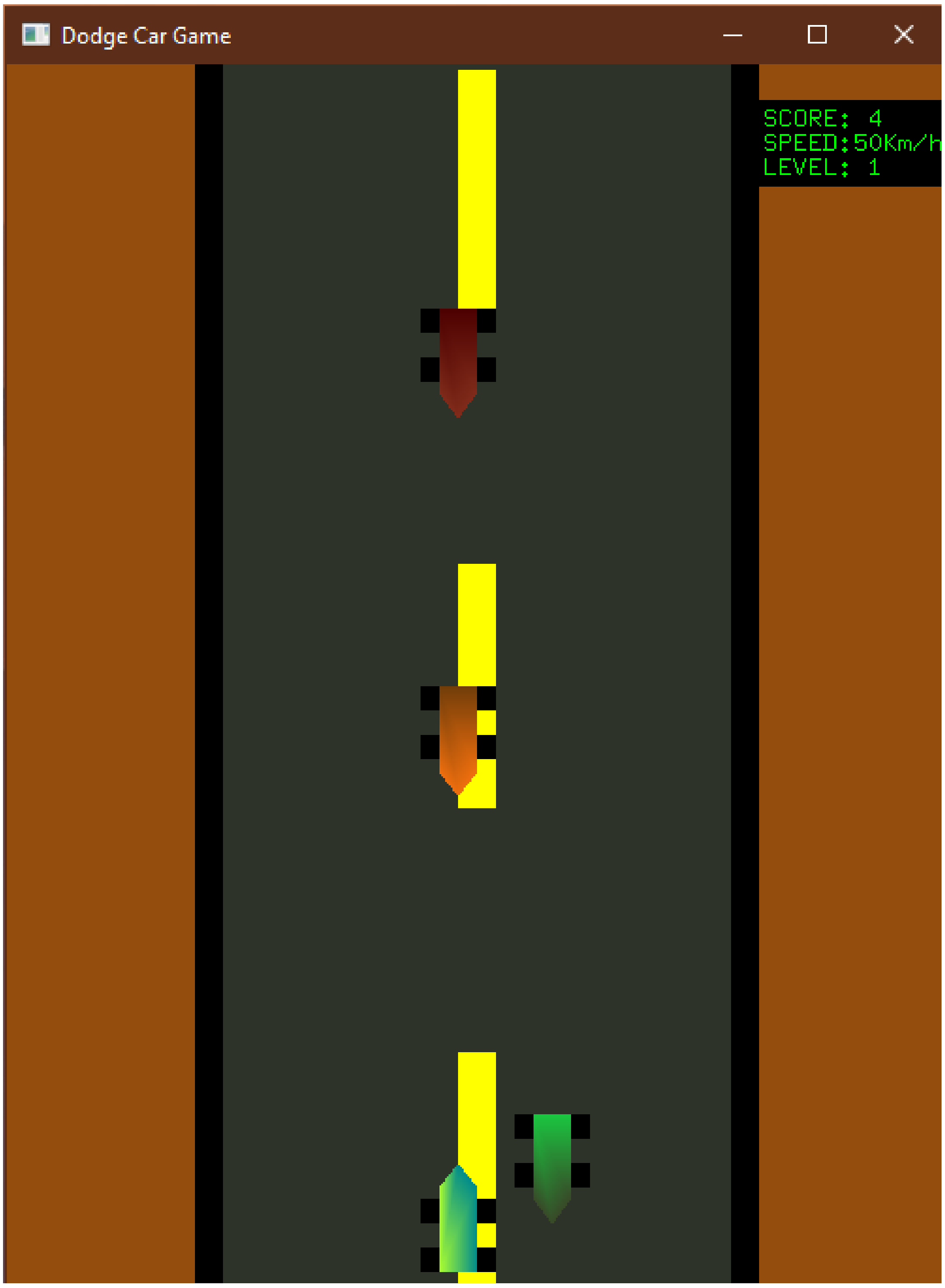


Fig 6 - Gameplay Screen

Chapter 5
Conclusion

As the project was progressive in nature, we learnt about gaming using computer graphics. It also helped us to understand various inbuilt functions of OpenGL. The keyboard interaction has helped us to implement this project in an easier way .As the project work was progressive in nature, and OpenGL methodology was employed, we obtained a good experience in OpenGL software development.

5.1 Future Work

The Game alone has lots of room for improvement. The future versions of the game can enable the player to choose the scenery i.e day, night, sunset, desert, ocean highway etc. The graphics of the game also have high scope for improvement including constructing a 3d version with multiple views. Some of the avenues for improvement are:

1. Choice of Scenery/Theme during Gameplay
2. 3D Rendering
3. Multiple Views
4. Inclusion of a Leaderboard
5. Inclusion of Choice of Initial Level of Difficulty
6. Inclusion of textures, complex shading and lighting models to enhance the aesthetic and clarity
7. More Streamlined Gameplay

References and Bibliography

1. Edward Angel-Interactive Computer Graphics:A Top-Down Approach using OpenGLFifth Edition, Published by Pearson Education, 2009
2. Code Resources - OpenGL Wiki (khronos.org) "https://www.khronos.org/opengl/wiki/Code_Resources"
3. The OpenGL Programming Guide, 5thEdition. The Official guide to learning OpenGL Version 2.1 by OpenGL Architecture Review Board
4. https://www.opengl.org/resources/libraries/

Appendix A: Source Code

```
#include <Windows.h>
#include <GL\glew.h>
#include <GL\freeglut.h>
#include<iostream>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <string>

//Game Speed
int FPS = 50;
//Game Track
int start = 0;
int gv = 0;
int level = 0;

//Track Score
int score = 0;

//For move track
float roadDivTop = 0;
float roadDivMdl = 0;
float 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 tree(int x, int y)
{
   int newx = x;
   int newy = y;
   //Bottom
   glColor3f(0.871, 0.722, 0.529);
   glBegin(GL_TRIANGLES);
   glVertex2f(newx + 11, newy + 55);
   glVertex2f(newx + 12, newy + 55 - 10);
   glVertex2f(newx + 10, newy + 55 - 10);
   glEnd();
   //Top
   glColor3f(0.133, 0.545, 0.133);
   glBegin(GL_TRIANGLES);
   glVertex2f(newx + 11, newy + 55 + 3);
   glVertex2f(newx + 12 + 3, newy + 55 - 3);
   glVertex2f(newx + 10 - 3, newy + 55 - 3);
   glEnd();
}


void startGame()
{
   //Road
   //glColor3f(0.412, 0.412, 0.412);   //light grey normal
   //glColor3f(1.0f, 1.0f, 1.0f);   //White Vector
   glColor3f(0.18, 0.2, 0.164);
   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);
   glColor3f(0.0f, 0.0f, 0.0f);
   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);
glColor3f(0.0f, 0.0f, 0.0f);
glBegin(GL_POLYGON);
glVertex2f(77, 0);
glVertex2f(77, 100);
glVertex2f(80, 100);
glVertex2f(80, 0);
glEnd();

//TOP
glColor3f(1.000, 1.000, 0.000);   // Yellow Original
//glColor3f(0.098, 0.098, 0.439);   // Violet Blue
//glColor3f(0.0f, 0.0f, 0.0f);        //Black Vector
glBegin(GL_POLYGON);
glVertex2f(48, roadDivTop + 80);
glVertex2f(48, roadDivTop + 100);
glVertex2f(52, roadDivTop + 100);
glVertex2f(52, roadDivTop + 80);
glEnd();
roadDivTop = roadDivTop - 0.5;
if (roadDivTop < -100)
{
   roadDivTop = 20;
   score++;
}
//Middle
glColor3f(1.000, 1.000, 0.000);       //Yellow Original
//glColor3f(0.098, 0.098, 0.439);       //Dark Blue Vector
//glColor3f(0.0f,0.0f,0.0f);           //Black Vector
glBegin(GL_POLYGON);
glVertex2f(48, roadDivMdl + 40);
glVertex2f(48, roadDivMdl + 60);
glVertex2f(52, roadDivMdl + 60);
glVertex2f(52, roadDivMdl + 40);
glEnd();
roadDivMdl = roadDivMdl - 0.5;
if (roadDivMdl < -60)
{
   roadDivMdl = 60;
   score++;
}
//Bottom
glColor3f(1.000, 1.000, 0.000);   //Yellow Original
//glColor3f(0.098, 0.098, 0.439);   //Dark Violet Blue Vector
```

```
//glColor3f(0.0f, 0.0f, 0.0f);      //Black Vector
glBegin(GL_POLYGON);
glVertex2f(48, roadDivBtm + 0);
glVertex2f(48, roadDivBtm + 20);
glVertex2f(52, roadDivBtm + 20);
glVertex2f(52, roadDivBtm + 0);
glEnd();
roadDivBtm = roadDivBtm - 0.5;
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 % 50 == 0)
{
   int last = score / 30;
   if (last != level)
   {
      level = score / 30;
      FPS = FPS + 10;
   }
}
char level_buffer[50];
sprintf_s(level_buffer, "LEVEL: %d", level + 1);
glColor3f(0.000, 1.000, 0.000);
renderBitmapString(80.5, 95 - 4, (void*)font3, level_buffer);

//Increase 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(0.678, 1.000, 0.184);
glBegin(GL_POLYGON);
glVertex2f(lrIndex + 26, 1);
glVertex2f(lrIndex + 26, 8);
glColor3f(0.000, 0.545, 0.545);
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); //Red Car Normal
glColor3f(0.501, 0.168, 0.102);
glBegin(GL_POLYGON);
glColor3f(0.3, 0.0, 0.0);
glVertex2f(lrIndex1 + 26, car1 + 100);
glColor3f(0.501, 0.168, 0.102);
glVertex2f(lrIndex1 + 26, car1 + 100 - 7);
```

```
glVertex2f(lrIndex1 + 28, car1 + 100 - 9);
glVertex2f(lrIndex1 + 30, car1 + 100 - 7);
glColor3f(0.3, 0.0, 0.0);
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))
{
   PlaySound(TEXT("MyAppSound"), NULL, SND_APPLICATION);
   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();

//glColor3f(0.300, 0.000, 0.500); //Violet Normal
//glClearColor(0.000, 0.392, 0.000, 1);
//glClearColor(0.219, 0.286, 0.156, 1); //Murky Green Grass Normal
glColor3f(0.219, 0.286, 0.156); //Violet New
glBegin(GL_POLYGON);
glColor3f(0.101, 0.78, 0.250);
glVertex2f(lrIndex2 + 26, car2 + 100);
glColor3f(0.219, 0.286, 0.156);
glVertex2f(lrIndex2 + 26, car2 + 100 - 7);
glVertex2f(lrIndex2 + 28, car2 + 100 - 9);
glVertex2f(lrIndex2 + 30, car2 + 100 - 7);
glColor3f(0.101,0.78,0.250);
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))
{
   PlaySound(TEXT("MyAppSound"), NULL, SND_APPLICATION);
   start = 0;
   gv = 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); //Orange Normal


glColor3f(0.937, 0.433, 0.054);
glBegin(GL_POLYGON);
glColor3f(0.447, 0.239, 0.035);
glVertex2f(lrIndex3 + 26, car3 + 100);
glColor3f(0.937, 0.433, 0.054);
glVertex2f(lrIndex3 + 26, car3 + 100 - 7);
glVertex2f(lrIndex3 + 28, car3 + 100 - 9);
glVertex2f(lrIndex3 + 30, car3 + 100 - 7);
glColor3f(0.447, 0.239, 0.035);
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))
    {
        PlaySound(TEXT("MyAppSound"), NULL, SND_APPLICATION);
        start = 0;
        gv = 1;
    }
}

void firstDesign()
{

    //Road BackGround
    //glColor3f(0.000, 0.350, 0.000); //Day and Night Dark green
    glColor3f(0.937, 0.433, 0.054);
    glBegin(GL_POLYGON);
    glVertex2f(0, 55);
    glVertex2f(100, 55);
    glColor3f(0.584, 0.3019, 0.0549); //Darker Front Hill color Sunset Dark Brown
    //glColor3f(0.6, 0.8, 0.3);
    glVertex2f(100, 50 - 50);
    glVertex2f(0, 50 - 50);
    glEnd();

    //Road Design In Front Page
    glColor3f(0, 0.3, 0.3);
    glBegin(GL_TRIANGLES);
    glVertex2f(32 - 2 + 21, 55);
    glVertex2f(32 + 58, 50 - 50);
    glVertex2f(32 - 22, 50 - 50);
    glEnd();
    //Road Middle
    glColor3f(0.8, 0.8, 0);
    glBegin(GL_TRIANGLES);
    glVertex2f(32 - 2 + 21, 55);
    glVertex2f(50 + 2, 50 - 50);
    glVertex2f(50 - 2, 50 - 50);
    glEnd();

    //Road Sky
    //glColor3f(0.000, 0.60, 1.000); //Day Sky Blue
    //glColor3f(0.0, 0.0, 0.0); //Night Black
    glColor3f(0.937, 0.433, 0.054); //Orange Sunset
    glBegin(GL_POLYGON);
    glVertex2f(100, 100);
    glVertex2f(0, 100);
    //glColor3f(0.933, 0.933, 0.933); //Light Greyish white Day
    //glColor3f(0.039, 0.050, 0.125);  //Night Dark Blue
    glColor3f(0.9, 0.0, 0.0);
    glVertex2f(0, 55);
```

```
glVertex2f(100, 55);
glEnd();

//Hill 1
//glColor3f(0.235, 0.702, 0.443); //Lighter Background hill color Day Green
//glColor3f(0.000, 0.400, 0.000); //Darker Front Hill color Day Green
glColor3f(0.776, 0.533, 0.3176); //Lighter Hill by the back Sunset light Brown
glBegin(GL_TRIANGLES);
glVertex2f(20, 55 + 10);
glVertex2f(20 + 7, 55);
glVertex2f(0, 55);
glEnd();

//Hill 2
//glColor3f(0.000, 0.400, 0.000); //Darker Front Hill color Day Green
//glColor3f(0.282, 0.376, 0.196); //Darker Front Hill color Night Faded Green
glColor3f(0.584, 0.3019, 0.0549);
glBegin(GL_TRIANGLES);
glVertex2f(20 + 15, 55 + 12);
glVertex2f(20 + 20 + 10, 55);
glVertex2f(0 + 10, 55);
glEnd();

//Hill 4
//glColor3f(0.235, 0.702, 0.443); //Lighter Background hill color Day Green
glColor3f(0.776, 0.533, 0.3176); //Lighter Hill by the back Sunset light Brown
glBegin(GL_TRIANGLES);
glVertex2f(87, 55 + 10);
glVertex2f(100, 55);
glVertex2f(60, 55);
glEnd();


//Hill 3
//glColor3f(0.000, 0.400, 0.000); //Darker Front Hill color Day Green
glColor3f(0.584, 0.3019, 0.0549); //Darker Front Hill color Sunset Dark Brown
glBegin(GL_TRIANGLES);
glVertex2f(70, 70);
glVertex2f(90, 55);
glVertex2f(50, 55);
glEnd();

//Tree on front page
tree(5, -15);
tree(0, 0);
tree(9, 5);
tree(80, 0);
tree(75, -15);
tree(72, 5);
```

```
//Menu Place Holder
  glColor3f(0.098, 0.098, 0.439);
  glBegin(GL_POLYGON);
  glVertex2f(32 - 4, 50 + 5 + 10);
  glVertex2f(32 + 46, 50 + 5 + 10);
  glVertex2f(32 + 46, 50 - 15 + 10);
  glVertex2f(32 - 4, 50 - 15 + 10);
  glEnd();

  glColor3f(00, 0, 0.000);
  glBegin(GL_POLYGON);
  glVertex2f(32 - 4, 50 + 5 + 10);
  glVertex2f(32 + 46, 50 + 5 + 10);
  glVertex2f(32 + 46, 50 + 4 + 10);
  glVertex2f(32 - 4, 50 + 4 + 10);
  glEnd();
  glBegin(GL_POLYGON);
  glVertex2f(32 + 45, 50 + 5 + 10);
  glVertex2f(32 + 46, 50 + 5 + 10);
  glVertex2f(32 + 46, 50 - 15 + 10);
  glVertex2f(32 + 45, 50 - 15 + 10);
  glEnd();
  glBegin(GL_POLYGON);
  glVertex2f(32 - 4, 50 - 14 + 10);
  glVertex2f(32 + 46, 50 - 14 + 10);
  glVertex2f(32 + 46, 50 - 15 + 10);
  glVertex2f(32 - 4, 50 - 15 + 10);
  glEnd();
  glBegin(GL_POLYGON);
  glVertex2f(32 - 4, 50 + 5 + 10);
  glVertex2f(32 - 5, 50 + 5 + 10);
  glVertex2f(32 - 5, 50 - 15 + 10);
  glVertex2f(32 - 4, 50 - 15 + 10);
  glEnd();

  //Text Information in First Page
  if (gv == 1)
  {
     //glColor3f(1.000, 0.000, 0.000); //GAME OVER IN RED
     glColor3f(1.0, 1.0, 0.0);
     renderBitmapString(35, 60 + 15, (void*)font1, "GAME OVER");
     glColor3f(1.000, 1.000, 0.000);
     char buffer2[50];
     sprintf_s(buffer2, "Your Score is : %d", score);
     renderBitmapString(33, 60 - 4 + 15, (void*)font1, buffer2);
  }

  glColor3f(1.000, 1.000, 1.000);
  renderBitmapString(30, 80, (void*)font2, "2D Dodge Car Game ");
```

```
    glColor3f(0.000, 1.000, 0.000);
    renderBitmapString(30, 50 + 10, (void*)font2, "Press SPACE to Start");
    renderBitmapString(30, 50 - 3 + 10, (void*)font2, "Press ESC to Exit");

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

    glColor3f(0.000, 1.000, 1.000);
    renderBitmapString(30 - 5, 50 - 40, (void*)font3, "PLAYER ONE");

    renderBitmapString(30 - 8, 50 - 43, (void*)font3, "152-15-6037");
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);

    if (start == 1)
    {
        //glClearColor(0.000, 0.392, 0.000, 1); //Light Green Grass Normal
        //glClearColor(0.098, 0.098, 0.439, 1); //Violet Blue Ocean Vector
        //glClearColor(0.219, 0.286, 0.156, 1); //Murky Green Grass Normal
        glClearColor(0.584, 0.3019, 0.0549,1); //Sunset Dark Brown Dirt

        startGame();
    }

    else
    {
        firstDesign();
    }

    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 - 2*(FPS / 10);
            if (lrIndex < 0)
            {
                lrIndex = -1;
            }
        }
        break;

    case GLUT_KEY_RIGHT:
        if (lrIndex <= 44)
        {
            lrIndex = lrIndex + 2*(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)
        {
            LPCTSTR filename = "car.wav";
            //PlaySound(filename, NULL, SND_ASYNC | SND_FILENAME | SND_LOOP);
            start = 1;
            gv = 0;
            FPS = 50;
            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("Dodge Car Game");



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

    glOrtho(0, 100, 0, 100, -1, 1);
    glClearColor(0.184, 0.310, 0.310, 1);

    glutTimerFunc(1000, timer, 0);
    glutMainLoop();

    return 0;
}
```