Project Name: 2D car racing game

Introduction:

A car racing game that we made is a simple project. Games have become an integral part of everyday life for many people. For this Car Racing Game, we would like to accomplish a video game imitating the existing game. The theme of our game is to compete with the other opponents that are controlled by computer in a racing tournament. The object of this game is to survive as long as possible and get to the High scores in the shortest possible time while avoiding the obstacles on the tracks. The application is target towards teenagers who enjoy playing racing game and want a new twist on the racing game genre. We will keep in mind when designing the user interface, drawing the graphics and creating the physics.

This game consists of three major modules. First part is the Game Logic Generator which calculates the logic of this game, such as to detect bumps to obstacle speed control based on keyboard input, opponents control and road generation, and this module is based on software. The second part is the Screen Rendering Module we adopt the Sprite Graphics technique to decompose the display screen into 7 layers which will be explained in derails in later section. The last part is the Audio Module which generates the proper sound under the control of game logic.

The main aim of this project report is to highlight the features as follows: -

- To show how the project is developed.
- To show the details of graphics and design.
- To show how the user can work with the software.

Requirement Analysis:

• **Hardware Requirements**

The most common set of requirements defined by any operating system or software application are the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatibility and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

Processor: Core i5 8th Gen

Ram: 4GB

Hard Disk: 1T

CPU Speed: 1.6GHz – 1.8GHz

• **Software Requirements**

Software Requirements deal with defining software resource requirements and pre-requisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or pre-requisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Operating System: Windows 10

IDE: Code Block (Version 17.10)

Glut Library Functions:

1. glDisable( ) & glEnable( )

The function was called between a call to glBegin and the corresponding call to glEnd. The glEnable and glDisable functions enable and disable various OpenGL graphics capabilities. Use glIsEnabled or glGet to determine the current setting of any capability.

2. glColor3f( )

The GL stores both a current single-valued color index and a current four valued RGBA color. glColor sets a new four-valued RGBA color. glColor has two major variants: glcolor3 and glcolor4. glColor3 variants specify new red, green, and blue values explicitly and set the current alpha value to 1.0 (full intensity) implicitly. glColor4 variants specify all four-color components explicitly.

3. glClear(GL_COLOR_BUFFER_BIT)

The glClear function sets the bitplane area of the window to values previously selected by glClearColor, glClearIndex, glClearDepth, glClearStencil, and glClearAccum. You can clear multiple color buffers simultaneously by selecting more than one buffer at a time using glDrawBuffer. The pixel-ownership test, the scissor test, dithering, and the buffer writemasks affect the operation of glClear. The scissor box bounds the cleared region. The glClear function ignores the alpha function, blend function, logical operation, stenciling, texture mapping, and zbuffering.4 The glClear function takes a single argument (mask) that is the bitwise OR of several values indicating which buffer is to be cleared.

4. glutBitmapCharacter( )

The function glutBitmapCharacter is used to print a bitmap character at a 3D position inside the model. A series of characters can be printer using glutBitmapCharacter which makes a string like functionality.

5. glutSolidSphere

Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

6. glVertex2f( )

The glVertex function commands are used within glBegin / glEnd pairs to specify point, line, and polygon vertices. The current color, normal, and texture coordinates are associated with the vertex when glVertex is called. When only x and y are specified, z defaults to 0.0 and w defaults to 1.0.

7. glLineStipple( )

Specifies a multiplier for each bit in the line stipple pattern. If factor is 3, for example, each bit in the pattern is used three times before the next bit in the pattern is used. factor is clamped to the range [1, 256] and defaults to 1. pattern. Specifies a 16-bit integer whose bit pattern determines which fragments of a line will be drawn when the line is rasterized.

8. glBegin ()&glEnd ()

glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives. glBegin accepts a single argument that specifies which often ways the vertices are interpreted.

Algorithm:

To implement our project, we have followed Two-Dimensional Transformations Algorithm. In this 2D Transformation.

Translation:

A translation process moves every point a constant distance in a specified direction. It can be described as a rigid motion. A translation can also be interpreted as the addition of a constant vector to every point, or as shifting the origin of the coordinate system. Suppose, If point (X, Y) is to be translated by amount Dx and Dy to a new location (X', Y') then new coordinates can be obtained by adding Dx to X and Dy to Y as: X' = Dx + X Y' = Dy + Y or P' = T + P where P' = (X', Y'), T = (Dx, Dy), P = (X, Y) Here, P (X, Y) is the original point. T (Dx, Dy) is the translation factor, i.e., the amount by which the point will be translated. P'(X', Y') is the coordinates of point P after translation.

Implementations:

```
#include<windows.h>

#ifdef __x86_64__ ///operating system on which the program is running with the
help of C programming

#else ///Use else to specify a block of code to be executed, if the same condition is
false

#include <GL/glut.h>

#endif

#include <stdlib.h>

#include <stdio.h>

#include <iostream>

#include <string>


//Game Speed

int FPS = 60;

//Game Track

int start=0;

int gv=0;

int level = 0;


//Track Score

int score = 0;
```

```
//Form move track

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; ///A 24-point
proportional spaced Times Roman font. The exact bitmaps to be used is defined by
the standard X glyph bitmaps for the X font named:

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;
    //Tree Left
            //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();
           //Up
        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);
   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 Middel Border
  //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++;
}
   //Midle
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 (buffer, "SCORE: %d", score);
glColor3f(0.000, 1.000, 0.000);
renderBitmapString(80.5,95,(void *)font3,buffer);
//Speed Print
char buffer1 [50];
sprintf (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 /50;
    if(last!=level){
        level = score /50;
        FPS=FPS+2;
    }
}

char level_buffer [50];
sprintf (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(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);
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();
glColor3f(0.294, 0.000, 0.510);
glBegin(GL_POLYGON);
glVertex2f(lrIndex2+26,car2+100);
glVertex2f(lrIndex2+26,car2+100-7);
glVertex2f(lrIndex2+28,car2+100-9);
glVertex2f(lrIndex2+30,car2+100-7);
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;

        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);

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 fristDesign(){
```

```
//Road Backgound
glColor3f(1.000, 1.392, 0.000);
glBegin(GL_POLYGON);
glVertex2f(0,55);
glVertex2f(100,55);
glColor3f(0.604, 0.804, 0.196);
glVertex2f(100,50-50);
glVertex2f(0,50-50);
glEnd();

//Road Design In Front Page
glColor3f(00, 0, 0);
glBegin(GL_TRIANGLES);
glVertex2f(32-2+21,55);
glVertex2f(32+58,50-50);
glVertex2f(32-22,50-50);
glEnd();
//Road Midle
glColor3f(1, 1, 1);
glBegin(GL_TRIANGLES);
glVertex2f(32-2+21,55);
```

```
glVertex2f(50+2,50-50);

glVertex2f(50-2,50-50);

glEnd();


 //Road Sky

glColor3f(0.000, 0.749, 1.000);

glBegin(GL_POLYGON);

glVertex2f(100,100);

glVertex2f(0,100);

glColor3f(0.686, 0.933, 0.933);

glVertex2f(0,55);

glVertex2f(100,55);

glEnd();


//Hill 1

glColor3f(0.235, 0.702, 0.443);

glBegin(GL_TRIANGLES);

glVertex2f(20,55+10);

glVertex2f(20+7,55);

glVertex2f(0,55);

glEnd();
```

```
   //Hill 2
glColor3f(0.000, 0.502, 0.000);
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);
glBegin(GL_TRIANGLES);
glVertex2f(87,55+10);
glVertex2f(100,55);
glVertex2f(60,55);
glEnd();

 //Hill 3
glColor3f(0.000, 0.502, 0.000);
glBegin(GL_TRIANGLES);
glVertex2f(70,70);
glVertex2f(90,55);
```

```
  glVertex2f(50,55);
  glEnd();



  //Tree Left
     //Bottom
  glColor3f(0.871, 0.722, 0.529);
  glBegin(GL_TRIANGLES);
  glVertex2f(11,55);
  glVertex2f(12,55-10);
  glVertex2f(10,55-10);
  glEnd();
     //Up
glColor3f(0.133, 0.545, 0.133);
  glBegin(GL_TRIANGLES);
  glVertex2f(11,55+3);
  glVertex2f(12+3,55-3);
  glVertex2f(10-3,55-3);
  glEnd();

  tree(5,-15);
tree(9,5);

tree(85,9);

tree(75,-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 Frist Page

if(gv==1){
        glColor3f(1.000, 0.000, 0.000);

        renderBitmapString(35,60+10,(void *)font1,"GAME OVER");

        glColor3f(1.000, 0.000, 0.000);

        char buffer2 [50];

        sprintf (buffer2, "Your Score is : %d", score);

        renderBitmapString(33,60-4+10,(void *)font1,buffer2);


    }


    glColor3f(1.000, 1.000, 0.000);

    renderBitmapString(30,80,(void *)font1,"2D Car Racing 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 turn Right");

    renderBitmapString(30,50-12+10,(void *)font3,"Press LEFT to turn Left");
```

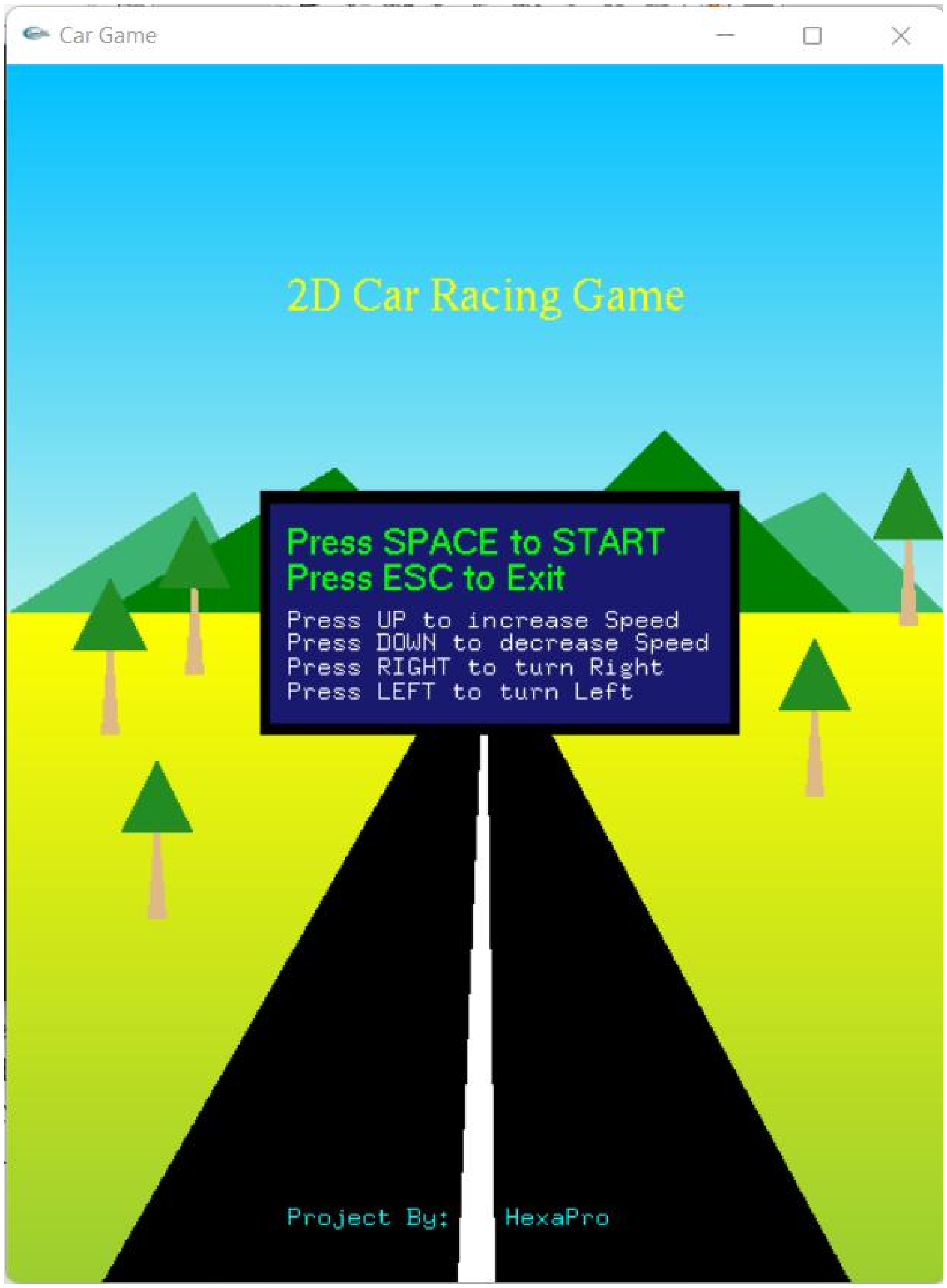
```
        glColor3f(0.000, 1.000, 1.000);

        renderBitmapString(35-5,30-25,(void *)font3,"Project By:");

        renderBitmapString(58-5,30-25,(void *)font3,"HexaPro");

}


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{

        fristDesign();

        //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("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;
}
```

Result and Snapshots:

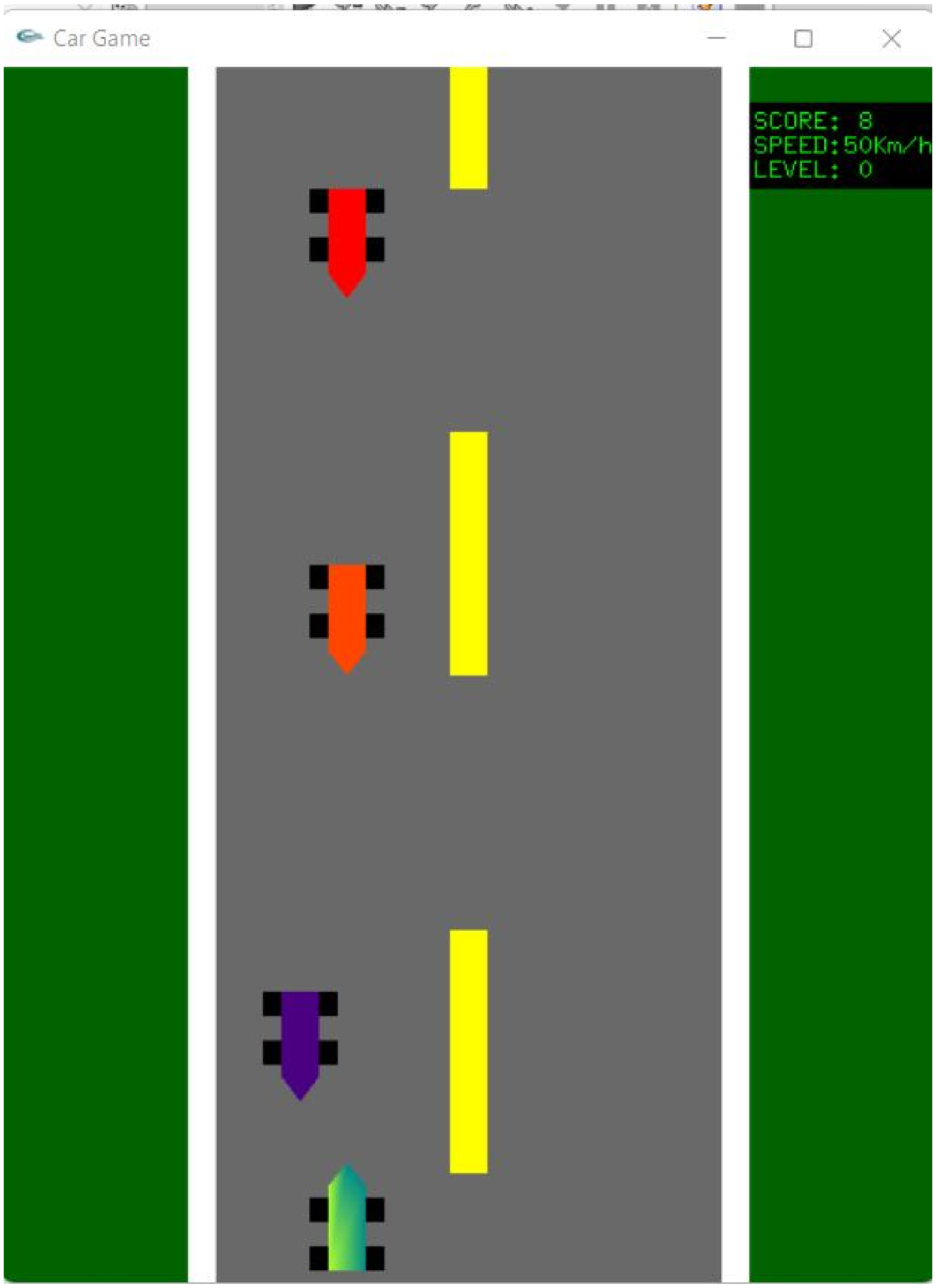


2D Car Racing Game

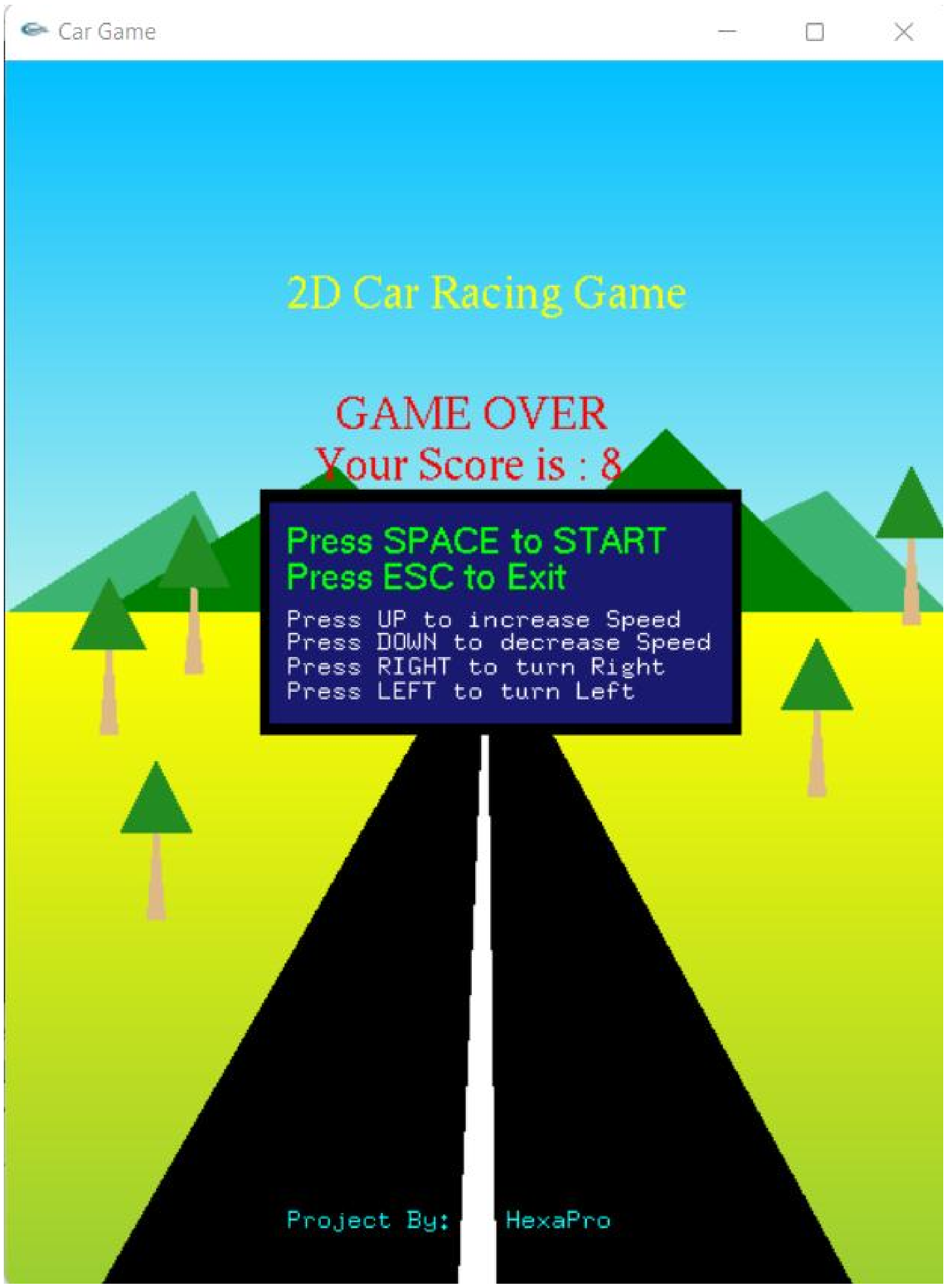
Press SPACE to START
Press ESC to Exit

Press UP to increase Speed
Press DOWN to decrease Speed
Press RIGHT to turn Right
Press LEFT to turn Left

Project By: HexaPro



```
SCORE: 8
SPEED:50Km/h
LEVEL: 0
```



2D Car Racing Game

GAME OVER
Your Score is : 8

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: HexaPro

Conclusion:

Always give it a try approach, before making comments I believe that gaming is a conceptual design, for a players it’s a virtual world but for a developer it’s how he make efficient use of key presses, key release and loop.