VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590018

A Computer Graphics & Visualization Mini Project Report
on

"Endless Runner with Body Tracking"

Submitted in Partial fulfillment of the Requirements for VI Semester of the Degree of

Bachelor of Engineering
In
Computer Science & Engineering
By

ABHIGYAN
(1CR19CS004)

AAYUSH DUBEY
(1CR19CS003)

Under the Guidance of

Mr. Kartheek G.C.R
Assistant Professor, Dept. of CSE
and
Mr. Shivaraj V B
Assistant Professor, Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI,

BANGALORE-560037

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI,

BANGALORE-560037

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Computer Graphics & visualization project work entitled **"Endless Runner with Body Tracking"** has been carried out by **Abhigyan (1CR19CS004)** and **Aayush Dubey (1CR19CS003)** bonafide students of CMR Institute of Technology in partial fulfillment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year **2021-2022**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. This CG project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

----------	----------
Signature of Guide	**Signature of HOD**
Mr. Kartheek G.C.R **Assistant Professor** **Dept. of CSE, CMRIT**	**Dr. Shreekanth M Prabhu** **Professor & Head** **Dept. of CSE, CMRIT**

External Viva

Name of the examiners	Signature with date
1.	
2.	

ABSTRACT

The proposed Computer Graphics project is Endless Runner with Body Tracking. This is designed and developed for an interactive gaming experience through real physical movement. This provides a unique and engaging method of exercise and entertainment and proves to be a better method of exercise than traditional method of exercising.

The Technologies used are OpenGL, OpenCV, MediaPipe, Pyautogui and the languages used are C++ and Python.

The primary purpose of this Gaming Application is to provide an alternative, fun and interactive way of playing with actual body movement to boost the motivation of the player. The player can use the body movements to move in-game and have a more personalized experience and can complete with other players and show off their skills.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of project would be incomplete without mentioning the people who made it possible, whose constant guidance and encouragement crowned our effort with success.

First and foremost, we would like to express our sincere words of gratitude and respect to our university, Visvesvaraya Technological University, Belgaum and our college CMR Institute of Technology, Bangalore for providing us an opportunity to carry out our project work report.

We express our sincere thanks to **Dr. Sanjay Jain, Principal** and **Dr.Shreekanth M Prabhu, Professor and Head of Department**, and our guides **Mr. Kartheek G C R, Assistant Professor, Dept of CSE** and **Mr. Shivaraj V B, Assistant Professor, Dept of CSE** who have been driving force behind the completion of our Project report and helping us in every aspect of project development.

Last but not the least we extend our thanks to all the people in the Department of Computer Science, for always being helpful. We are very grateful to our parents and well-wishers for their continuous moral support and encouragement.

TABLE OF CONTENT

Chapter	Title	Page Number
1	Introduction	1
2	System Requirement	2
3	Design	3
4	Implementation	4
5	Discussions and Screenshots	17
6	Conclusion and Future Scope	20
7	Bibliography	21

LIST OF FIGURES

Fig no.	Title	Page Number
3.1	State Transition	3
5.1	Start Game	17
5.2	Gameplay	17
5.3	Gameplay	18
5.4	Gameplay	18
5.5	BodyTracking	19
5.6	BodyTracking	19
5.7	BodyTracking	19

Chapter 1

INTRODUCTION

This is an application targeted towards a field that has much to improve in terms of quality of experience and the various ways and methods of propagating the information targeted towards players. This app enables a way of exercising through a fun game and involving many obstacles and fun elements essentially making playing video game a very fun and innovative process.

The use of games and activities is very minimal in the current architecture of fitness industry. This project aims at alleviating the gap and forming strong foundational bonds between games and exercising. Games have always been thought of as a mode of relaxing and having fun, although deemed in a sense not good or a waste of time by the caretakers of fitness influencers. Studies prove that gaming can actually increase brain co-ordination and function significantly and now combined with physical real-world movement and courses, this could revolutionize how we look at gaming.

This is step towards a better and more connected social way of staying active and healthy, that is sure to be fun, interactive and a better way of playing.

Chapter 2

SYSTEM REQUIREMENT

i. Operating systems: Windows XP,7,8,10,11, Linux, MacOS X
ii. Processors: Intel x64, AMD x64, apple silicon, ARM based processors
iii. Size: >100MB Hard disk space
iv. RAM: minimum 2 GB
v. Resolution: minimum 1280*720, other resolutions supported at 16:9 aspect ratio
vi. GPU: any integrated or dedicated GPU with minimum 256MB VRAM
vii. Camera with at least 2MP resolution

Chapter 3

DESIGN

This game features a very innovating method of playing games through physical movement.

It depends upon an image capturing device and feeds the live feed of the player to the system.

It is then processed by the OpenCV library and body poses and tracking is done through Mediapipe library.

This body tracking is used to determine the position of player in real time and his movements are tracked and detected for movement in game.

This is synced through the keyboard STDIN and provided to the game.

The game moves the player accordingly by the data provided by the image processing server.

The player can move left and right dodging obstacles along the way.

The player can also jump above the blocks in difficult instances.

The goal is to reach as far as possible without colliding with obstacles.

There are 3 lives given to each player to progress throughout the level.

The score is displayed on top and it represents the distance that the player has travelled.

Once the player has lost all of his/her lives the game is over.

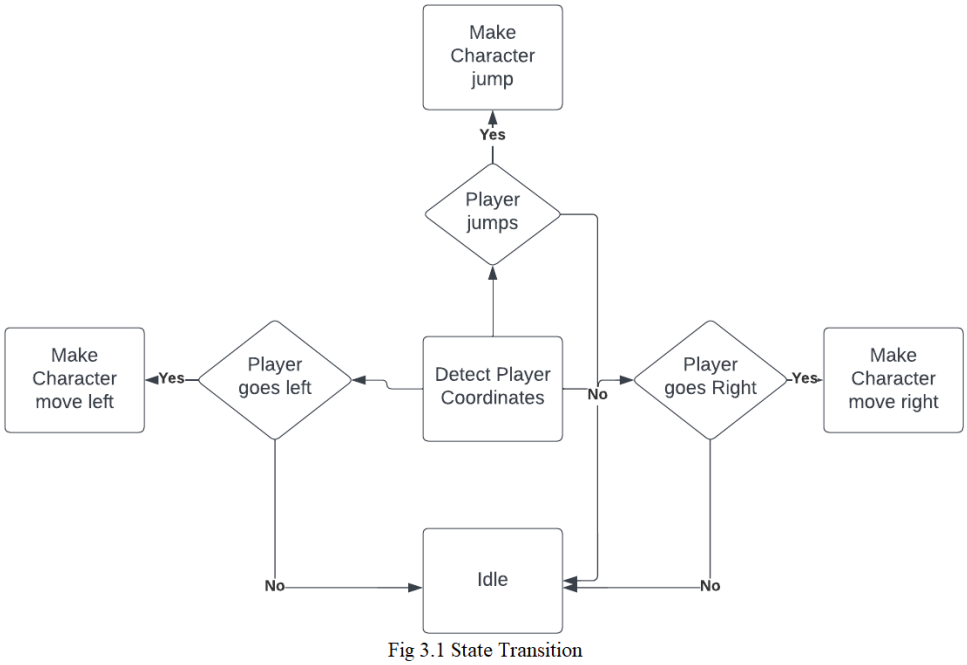

Fig 3.1 State Transition

Chapter 4

IMPLEMENTATION

Main.cpp

```
#include<stdio.h>
#include <cstdlib>
#include<time.h>
#include<string.h>
#include <string>
#define GL_SILENCE_DEPRECATION
#include<GLUT/glut.h>
#include<cmath>
using namespace std;
//int axis = 0;
//float theta[3] = { 0,0,0 };
bool start = false;
bool gameover = false;
bool jump = false;
unsigned int m_screenWidth = 750;
unsigned int m_screenHeight = 750;
int color_var = 0;
void* m_glutFont = GLUT_BITMAP_TIMES_ROMAN_24;
float player_y = 0.35;
float viewer[3] = { 0,0,2 };
float player[3] = { 0,player_y,0 };
float jumpsin = 0;
float enemypos[100];
float enemypos2[100];
float enemypos3[100];
float y_bounce=0;
float map_len = 400;
int immunity=0;
int lives=3;
GLfloat mat_cube[] = { 1.0f,1.0f,1.0f };
GLfloat mat_enemy[] = { 0.8f,0.1f,0.1f };
GLfloat mat_floor[] = { 0.6f,0.6f,0.6f };

GLfloat mat_amb[] = { 0.7f,0.7f,0.7f,1.0f };
```

```
GLfloat mat_diff[] = { 0.7f,0.7f,0.7f,1.0f };
GLfloat mat_spec[] = { 0.7f,0.7f,0.7f,1.0f };
GLfloat mat_shine[] = { 50.0f };


//GLfloat lightintensity[] = { 0.9f,0.9f,0.9f,1.0f };
GLfloat lightintensity[] = { 0.0f,0.0f,1.0f,1.0f };
GLfloat lightintensity1[] = { 0.0f,0.9f,0.9f,1.0f };
//GLfloat lightposition[] = { viewer[0],viewer[1],viewer[2],0.0f };
GLfloat lightposition[] = { 0,2,-5,1.0f };
GLfloat lightposition1[] = { viewer[0],viewer[1],viewer[2],0.0f };
GLfloat lightpositionarr[100][4];
//float v[8][3] = { {-1,-1,-1},{-1,1,-1},{1,1,-1},{1,-1,-1},{-1,-1,1},{-1,1,1},{1,1,1},{1,-1,1}
};
void display();
void drawStair(float x, float y);
void drawSpectator(float x, float y, float z);

void printStairs()
{
   drawStair(6, 0);
   drawStair(-6, 0);
   drawStair(8, 0.5);
   drawStair(-8, 0.5);
   drawStair(10, 1);
   drawStair(-10, 1);
}
void printSpectators()
{
   for (int i = -3; i > -map_len / 2; i -= 2)
   {

      drawSpectator(4.5, 0.5+y_bounce, i);
      drawSpectator(-4.5, 0.5+y_bounce, i);
      drawSpectator(6, 1+y_bounce, i+1);
      drawSpectator(-6, 1+y_bounce, i+1);
      drawSpectator(7.5, 1.5+y_bounce, i);
      drawSpectator(-7.5, 1.5+y_bounce, i);
   }
}
void drawStair(float x, float y)
{
   glPushMatrix();
   glTranslated(x, y, 0);
```

```
    glScaled(6, 1, map_len);
    glutSolidCube(1.0);
    glPopMatrix();
}
void drawSpectator(float x, float y, float z)
{
    glPushMatrix();
    glTranslated(x, y, z);
    glScaled(0.3, 1, 0.3);
    glutSolidCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslated(x, y+0.76, z+0.45);
    glRotatef(30, 1, 0, 0);
    glScaled(0.1, 0.3, 0.1);
    glutSolidCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslated(-x, y+0.76, z-0.45);
    glRotatef(-30, 1, 0, 0);
    glScaled(0.1, 0.3, 0.1);
    glutSolidCube(1.0);
    glPopMatrix();

    glPushMatrix();
    glTranslated(x, y+0.8, z);
    //glScaled(0.3, 1, 0.3);
    glutSolidSphere(0.25,16,16);
    glPopMatrix();
}
void drawText(const std::string& text, const unsigned int x, const unsigned int y)
{
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0, m_screenWidth, 0, m_screenHeight, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    glColor3f(1.0f, 1.0f, 1.0f);
    glRasterPos2i(x, y);
    for (const char c : text)
```

```
        glutBitmapCharacter(m_glutFont, (int)c);
    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
}
void printText(string s)
{
    //printf("Entering");
    glColor3f(1.0, 0.0, 1.0);
    glRasterPos2f(0.1f, 0.1f);
    int len = int(s.length());
    for (int i = 0; i < len; i++) {
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, s[i]);
    }
}
void keyboard(unsigned char key, int x, int y)
{
    printf("key = %c", key);
    if (key == 's')
    {
        start = true;
        printf("Game Started");
    }
}
void keys(int key, int x, int y)
{
    if (start)
    {
        if (key == GLUT_KEY_LEFT) player[0] -= 1.4;
        if (key == GLUT_KEY_UP) if (!jump) jump = true;
        if (key == GLUT_KEY_RIGHT) player[0] += 1.4;
    }
}
void init()
{
    srand(time(0));
    for (int i = 0; i < 100; i++)
    {
        enemypos[i] = (rand() % 3 - 1) * 1.4;
        enemypos2[i] = (rand() % 3 - 1) * 1.4;
        enemypos3[i] = (rand() % 3 - 1) * 1.4;
    }
}
```

```
void move()
{
   if (start)
   {
      player[2] -= 0.1;
      immunity--;
      printf("%d\n",lives);
   }
   if(lives<1)
   {
      start=false;
      gameover = true;
   }
   display();
}

void changeMaterial(float rgb[3], float sh)
{
   for (int i = 0; i < 3; i++)
   {
      mat_amb[i] = rgb[i];
      mat_diff[i] = rgb[i];
      mat_spec[i] = rgb[i];
   }
   mat_shine[0] = sh;
   glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb);
   glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diff);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shine);
}
void ground(double thickness)
{
   glPushMatrix();
   glTranslated(0, 0, 0);
   glScaled(5, thickness, map_len);
   glutSolidCube(1.0);
   glPopMatrix();
}
void drawPlayer(double size)
{
   glPushMatrix();
   glTranslated(player[0], player[1], player[2]);
   glScaled(1, 0.7, 1);
```

```
    glutSolidCube(1.0);
    glPopMatrix();
}
void drawEnemyCube(float pos_x, float pos_z)
{
    glPushMatrix();
    glTranslated(pos_x, player_y, pos_z);
    glScaled(1, 0.7, 1);
    glutSolidCube(1.0);
    glPopMatrix();
}
void jumpfun()
{
    if (jumpsin > 180)
    {
        jumpsin = 0;
        player[1] = player_y;
        jump = false;
    }
    player[1] = player_y + sin(jumpsin * 3.1415 / 180);
    //printf("%lf    %lf\n",sin(jumpsin),player[1]);
    jumpsin += 3;
}
void drawPointLight(GLfloat p[], GLfloat li[])
{
    //printf("%f %f %f\n", p[0], p[1], p[2]);
    glPushMatrix();
      //glTranslatef(p[0], p[1], p[2]);
        glLightfv(GL_LIGHT1, GL_POSITION, p);
        glLightfv(GL_LIGHT1, GL_DIFFUSE, li);
        glLightfv(GL_LIGHT1, GL_AMBIENT, li);
        glLightfv(GL_LIGHT1, GL_SPECULAR, li);
        const GLfloat c[] = { 100 };
        glLightfv(GL_LIGHT1, GL_SPOT_EXPONENT, c);
        GLfloat a[] = {0,0,-5};
        glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, a);
        const GLfloat b[] = {30};
        glLightfv(GL_LIGHT1, GL_SPOT_CUTOFF, b);


    glPopMatrix();

}
void drawEnemies()
```

```
{
   for (int i = -6, j = 0; i > -map_len / 2; i -= 12, j++)
   {

      if (i < player[2] + 0.5)
      {
         drawEnemyCube(enemypos[j], float(i));
         if ((int(player[0]) == int(enemypos[j]) && i > player[2]) && (player[1] - player_y <
0.55))
         {
            if(immunity<0)
            {
               lives--;
               immunity=7;
            }
         }

         if (j % 6 == 0)
         {
            drawEnemyCube(enemypos3[j], float(i));
            if ((int(player[0]) == int(enemypos3[j]) && i > player[2]) && (player[1] -
player_y < 0.55))
            {
               if(immunity<0)
               {
                  lives--;
                  immunity=7;
               }

            }

         }
         if (j % 3 == 0)
         {
            drawEnemyCube(enemypos2[j], float(i));
            if ((int(player[0]) == int(enemypos2[j]) && i > player[2]) && (player[1] -
player_y < 0.55))
            {
               if(immunity<0)
               {
                  lives--;
                  immunity=7;
               }
            }
```

```
        }

      }

   }

}
void drawLightPoles()
{
   for (int i = -6, j = 0; i > -map_len / 2; i -= 12, j++)
   {

      if (i < player[2] + 0.5)
      {
         //drawPointLight(lightposition);
         glLightfv(GL_LIGHT0, GL_DIFFUSE, lightintensity);
      }
   }
}

void changeColor()
{

    if (mat_floor[color_var] >= 1.0)
      {
         mat_floor[color_var] = 1;
         color_var++;
         if (color_var == 2)
         {
            color_var = 0;
            mat_floor[0] =0;
            mat_floor[1] =0;
            mat_floor[2] = 0;
            mat_floor[rand() % 3]=1;
         }

      }

   mat_floor[color_var] += 0.01;

}

void display()
{
```

```
//lightposition[0] = viewer[0];
//lightposition[1] = viewer[1];
//lightposition[2] = viewer[2];
//glLightfv(GL_LIGHT0, GL_POSITION, lightposition);



glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-2, 2, -2, 2, 1, 10);
//glOrtho(-4, 4, -4, 4, 4, 100.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
viewer[0] = player[0];
viewer[1] = player[1] + 1;
viewer[2] = player[2] + 1.5;
//gluLookAt(0, viewer[1], viewer[2],0, player[1], player[2], 0.0, 1.0, 0.0);
gluLookAt(viewer[0], viewer[1], viewer[2], player[0], player[1], player[2], 0.0, 1.0, 0.0);

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
//drawPointLight(lightposition, lightintensity);

//changeColor();
changeMaterial(mat_floor, 50);
glPushMatrix();
ground(0.02);
glPopMatrix();

changeMaterial(mat_floor, 50);
printStairs();
printSpectators();

changeMaterial(mat_cube, 80);
if (jump)
{
   jumpfun();
}
glPushMatrix();
drawPlayer(1);
glPopMatrix();
//printf("%f", player[2]);
changeMaterial(mat_enemy, 80);
glPushMatrix();
```

```
    drawEnemies();
    glPopMatrix();
    //changeMaterial(mat_cube, 80);
    glDisable(GL_LIGHTING);
    if (!start&&!gameover)
    {
        drawText("Press 's' or Raise your right hand to start", 180, 600);
    }
    if(gameover)
    {
        drawText("GAME OVER", 300, 600);
    }
    string score = "Score: "+to_string(-int(player[2])) ;
    drawText(score, 325, 700);
    string life = "Lives: ";
    for (int i=0; i<lives; i++) {
        life+="O";
    }
    drawText(life,50, 700);
    glEnable(GL_LIGHTING);
    glutSwapBuffers();
    glFlush();

}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(m_screenWidth, m_screenHeight);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("ABHIGYAN AND DUBEY ENDLESS RUNNER");

    //int x = 1000, y = 1000, n = 4, ok;
    //ok = stbi_info("a.png", &x, &y, &n);
    //printf("%d", ok);
    init();
    glutDisplayFunc(display);

    glutSpecialFunc(keys);
    glutIdleFunc(move);
    glutKeyboardFunc(keyboard);

    glEnable(GL_LIGHTING);
```

```
    glEnable(GL_LIGHT0);
    //glEnable(GL_LIGHT1);

    glEnable(GL_NORMALIZE);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    return 0;
}
```

Body_tracking.py

```
import cv2
import mediapipe as mp
import pyautogui

mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_pose = mp.solutions.pose

boundaries = {"left":0.6,"right":0.4,"bottom":0.6,"top":0.4}
states = ("left","right","up","down","center")
x,y = 0.5,0.5
cap = cv2.VideoCapture(0)
cur_state = "idle"
new_state = "idle"
tutorial = True
flag = False

def move(key):
    print(key)
    pyautogui.press(key)

def transition():
    global cur_state
    if new_state!= cur_state:
        if cur_state == states[4] and (new_state == states[1] or new_state == states[0]):
            move(new_state)
        elif new_state == states[4] and (cur_state == states[1] or cur_state == states[0]):
            if cur_state == states[1]:
                move(states[0])
            else:
                move(states[1])
        elif new_state == states[3] or new_state == states[2]:
```

```
            move(new_state)

        cur_stsate = new_state

def check():
    global cur_state
    global new_state
    #print(state)
    if y < boundaries["top"]:
            new_state = states[2]
    elif y > boundaries["bottom"]:
            new_state = states[3]
    elif x >boundaries["left"]:
        new_state = states[0]
    elif x <boundaries["right"]:
            new_state = states[1]
    else:
        new_state = "center"
    transition()
def start(y):
    global flag
    if(y<boundaries["top"]):
        flag = True
        #pyautogui.click()
        pyautogui.press('s')
        print("start")

cap = cv2.VideoCapture(0)
with mp_pose.Pose(
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as pose:
  while cap.isOpened():
    success, image = cap.read()
    cap.set(3,640)
    cap.set(4,480)
    if not success:
      print("Ignoring empty camera frame.")
      # If loading a video, use 'break' instead of 'continue'.
      continue
    h, w, c = image.shape
    #print(h, w, c)
    if (tutorial):
        cv2.rectangle(image, (0, int(boundaries["top"]*h)), (int(w*boundaries["right"]),
int(boundaries["bottom"]*h)), (0, 255, 0), 1)
```

```
        cv2.rectangle(image, (w, int(boundaries["top"]*h)), (int(w*boundaries["left"]),
int(boundaries["bottom"]*h)), (0, 255, 0), 1)
        cv2.rectangle(image, (0, 0), (w, int(boundaries["top"]*h)), (0, 255, 0), 1)
        cv2.rectangle(image, (0, int(boundaries["bottom"]*h)), (w, h), (0, 255, 0), 1)
    # To improve performance, optionally mark the image as not writeable to
    # pass by reference.
    image.flags.writeable = False
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = pose.process(image)

    # Draw the pose annotation on the image.
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    if results.pose_landmarks:
        mp_drawing.draw_landmarks(
            image,
            results.pose_landmarks,
            mp_pose.POSE_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles.get_default_pose_landmarks_style())
        cv2.rectangle(image, (int(x * w) - 1, int(y * h) - 1), (int(x * w) + 1, int(y * h) + 1), (0,
255, 0), 2)
        x = (results.pose_landmarks.landmark[mp_pose.PoseLandmark.LEFT_SHOULDER].x
+ results.pose_landmarks.landmark[
            mp_pose.PoseLandmark.RIGHT_SHOULDER].x) / 2
        y = (results.pose_landmarks.landmark[mp_pose.PoseLandmark(11).value].y +
results.pose_landmarks.landmark[
            mp_pose.PoseLandmark(24).value].y) / 2

        if flag:
            check()
        if not flag:
            start(results.pose_landmarks.landmark[mp_pose.PoseLandmark(16).value].y)
    # Flip the image horizontally for a selfie-view display.
    cv2.imshow('MediaPipe Pose', cv2.flip(image, 1))
    if cv2.waitKey(5) & 0xFF == 27:
      break
cap.release()
```

Chapter 5

DISCUSSIONS AND SCREENSHOTS

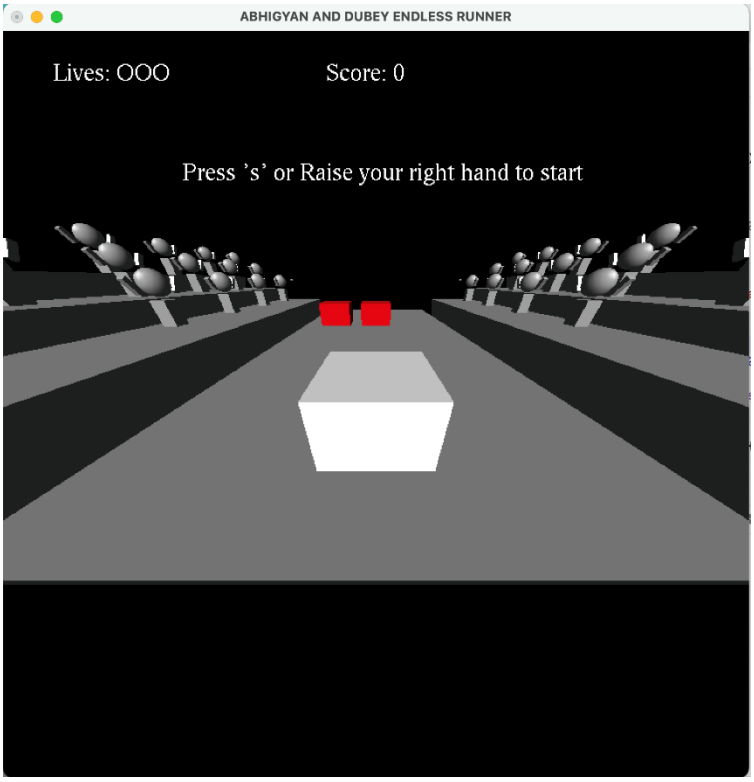

Fig 5.1 Start Game

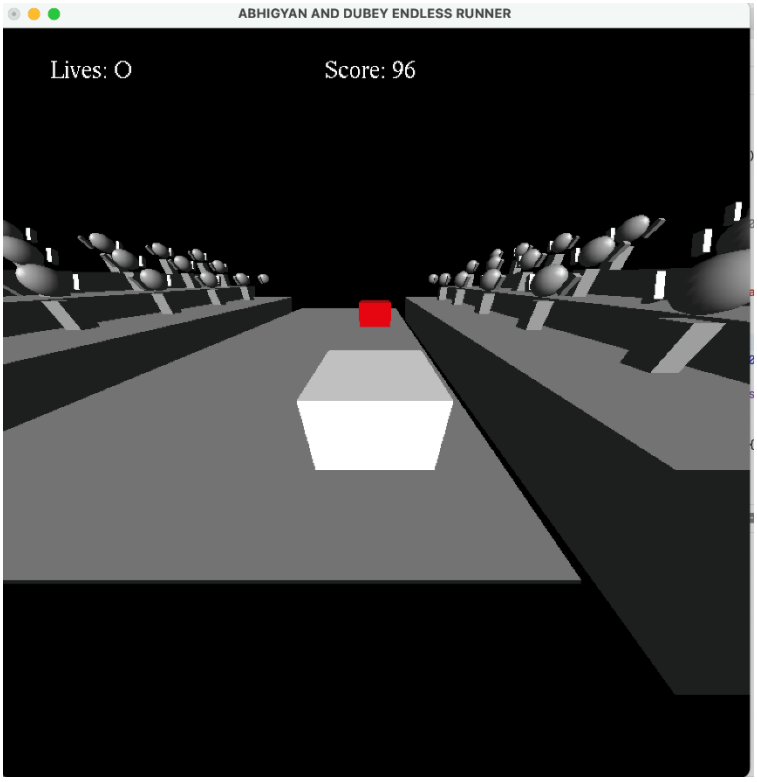

Fig 5.2 Gameplay

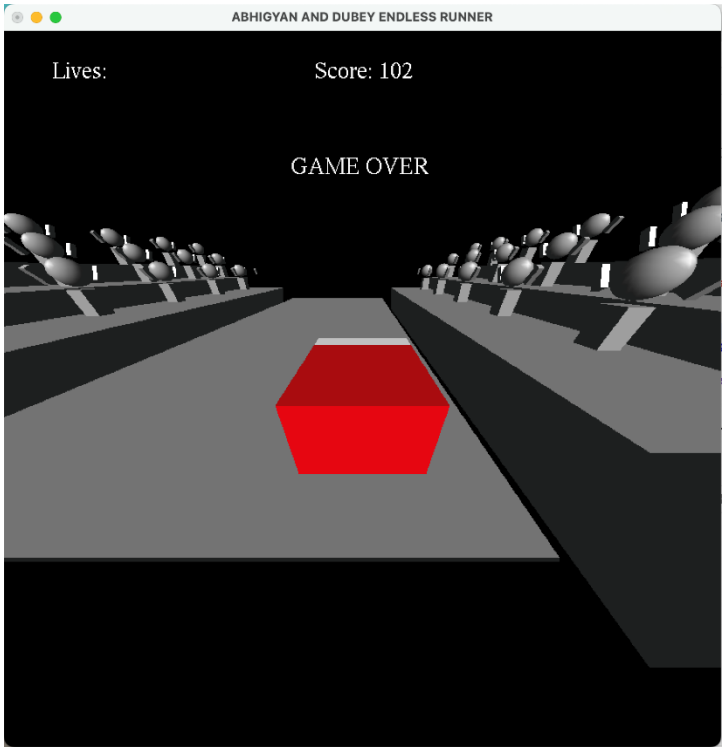

Fig 5.3 Gameplay

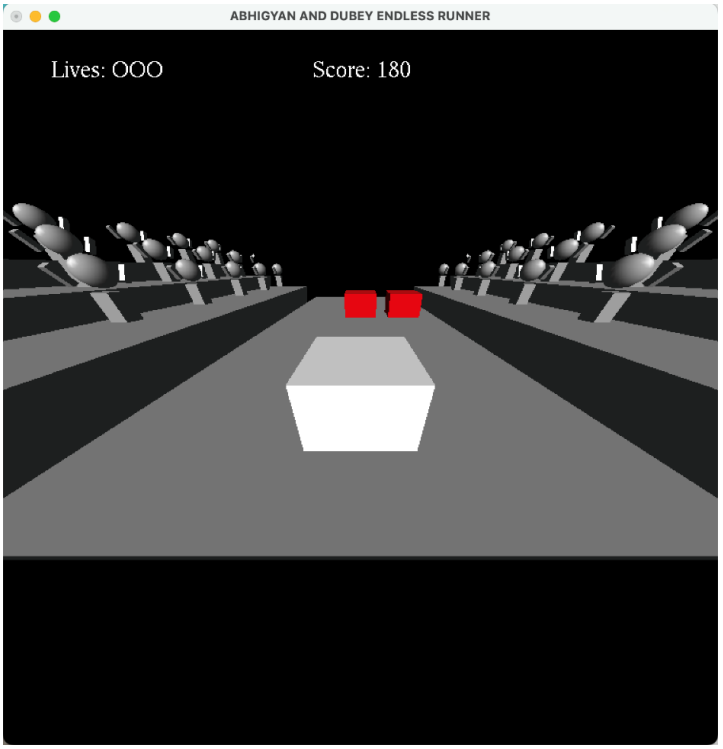

Fig 5.4 GameOver

Fig 5.5 BodyTracking

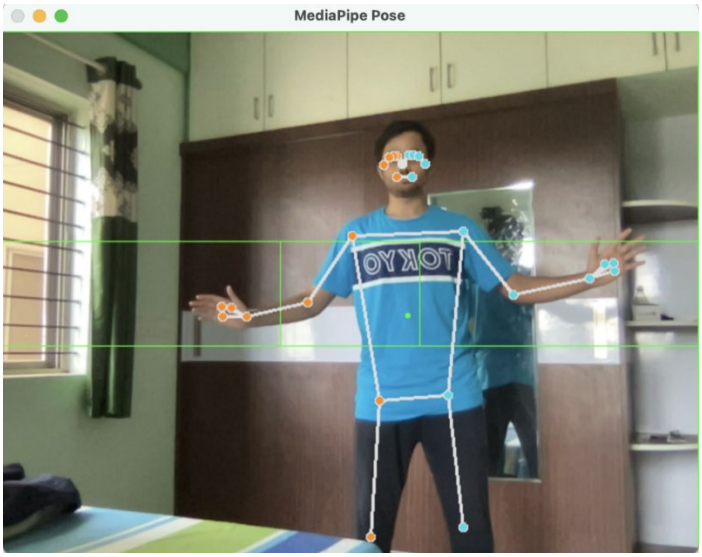

Fig 5.6 BodyTracking

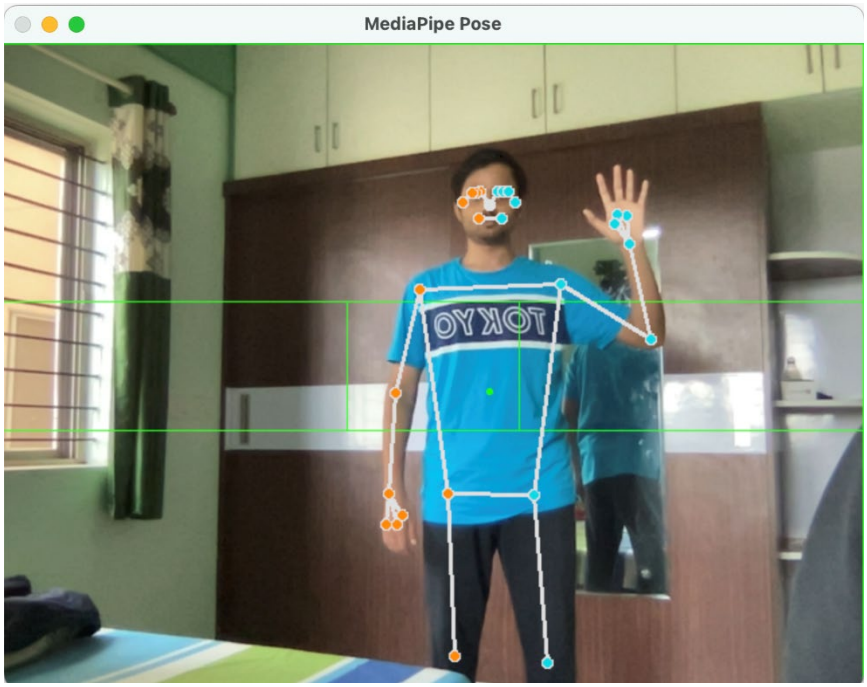

Fig 5.7 BodyTracking

Chapter 6

CONCLUSION AND FUTURE SCOPE

The project on Computer Graphics allowed us to think creatively and develop our own application using the languages C++, Python and Technologies like OpenGL and OpenCV.

Future Scope:

i. Multiplayer Control
ii. Expanding the map and introducing new obstacles
iii. Adding more levels and maps
iv. Making the UI more Playful and Interactive.

BIBILOIGRAPHY

[1] https://www.khronos.org/registry/OpenGL-Refpages/

[2] https://docs.microsoft.com/en-us/windows/win32/opengl/opengl

[3] https://docs.opencv.org/4.x/

[4] https://stackoverflow.com/

[5] https://google.github.io/mediapipe/

[6] https://pyautogui.readthedocs.io/en/latest/