Project Title: Speed Racer – A game made using Python

Subject: Decision Modelling

Class: Msc.AI – Part I

Team Members and their student Id's:

Sharvaj Patil (4808540)

Dinesh Saliyar (4820242)

Sakshi Dhadwad (4505044)

Gaurav Patil (4671956)

Alex Fernandes (4810930)

1. Introduction

Aim

The primary aim of the Speed Racer project is to design and develop an engaging, interactive 2D racing game using the Python programming language and the Pygame library.

Problem Statement

In the realm of interactive entertainment and software development, creating engaging and educational projects is a significant challenge. One area that offers a comprehensive learning experience is game development, which encompasses various aspects of programming, multimedia integration, and user interaction design. The goal is to develop a simple yet entertaining 2D racing game using Python and the Pygame library

Introduction

Speed Racer is an engaging, fast-paced racing game developed using Python and the Pygame library. The objective is to navigate the car through a busy road while avoiding obstacles and collecting fuel to maintain the journey. The game keeps track of the distance traveled and the player's high score.

2. Scope

The scope of the Speed Racer project encompasses the design, development, testing, and documentation of a 2D racing game using Python and the Pygame library. The project aims to deliver a fully functional, engaging game that serves as both an educational tool and an entertainment product.

3. Objectives

1. Navigate the Car
2. Avoid Obstacles
3. Collect Fuel
4. Maximize Distance
5. Achieve High Scores
6. Maintain Fuel Levels
7. Experience Progression
8. Provide Feedback and Enjoyment

4. Motivation

The motivation behind the Speed Racer project stems from a combination of educational, creative, and entertainment-driven factors. Developing a 2D racing game using Python and Pygame offers an excellent opportunity to explore various aspects of game development, enhance programming skills, and create a fun and interactive application.

5. Literature Review

References

- Pygame Documentation
  https://www.pygame.org/docs/

- Tutorials on Pygame Game Development
  https://www.youtube.com/watch?v=XON1IA8MxQw&list=PLz7mmheDeooW795HsNzkLS6TFQSa4Idtw

6. Methodology

Pygame Library
Pygame is a set of Python modules designed for writing video games. It includes computer graphics and sound libraries.

Game Components
Car: The player-controlled vehicle that navigates the road.
Road: The track or path on which the car travels.
Obstacles: Various objects or entities that the player must avoid, such as other cars or barriers.
Fuel Pickups: Items that the player collects to replenish the car's fuel and continue driving.
User Interface (UI): Visual elements displayed on the screen to provide information to the player, including score, fuel level, and high score.
Graphics: Visual assets such as car sprites, road backgrounds, obstacles, and fuel pickups.
Audio: Sound effects for actions like collecting fuel, crashing, and background music to enhance the gaming experience.
Game Logic: Code that governs the rules and mechanics of the game, including collision detection, scoring, and fuel management.
Game States: Different states of the game, such as running, paused, and game over, managed by the game loop.
Control System: Keyboard inputs used by the player to control the car's movement, typically using arrow keys or other designated keys.
High Score System: Mechanism to track and display the highest score achieved

by the player across multiple game sessions.
Performance Optimization: Techniques employed to ensure the game runs smoothly at a consistent frame rate, even on different hardware configurations

7. Implementation

Code

The game is implemented in Python using the Pygame library. Below is a simplified version of the main game loop:

game.py

```python
import pygame
import time
import random
import os

pygame.init()
pygame.mixer.init()

gameWindow = pygame.display.set_mode((1200,700))
pygame.display.set_caption("Speed Racer")

clock = pygame.time.Clock()
fps = 60

font1 = pygame.font.SysFont("Franklin Gothic Demi Cond",50)

car = pygame.image.load("data/images/Car.png")
car = pygame.transform.scale(car,(150,150)).convert_alpha()

road = pygame.image.load("data/images/Road.png")
road = pygame.transform.scale(road,(400,700)).convert_alpha()

sand = pygame.image.load("data/images/Sand.jpg")
sand = pygame.transform.scale(sand,(150,700)).convert_alpha()

leftDisp = pygame.image.load("data/images/LeftDisplay.png")
leftDisp = pygame.transform.scale(leftDisp,(250,700)).convert_alpha()

rightDisp = pygame.image.load("data/images/RightDisplay.png")
rightDisp = pygame.transform.scale(rightDisp,(250,700)).convert_alpha()

tree = pygame.image.load("data/images/Tree.png")
tree = pygame.transform.scale(tree,(185,168)).convert_alpha()
treeLXY = [[290,0],[290,152.5],[290,305],[290,457.5],[290,610]]
treeRXY = [[760,0],[760,152.5],[760,305],[760,457.5],[760,610]]

strip = pygame.image.load("data/images/Strip.png")
strip = pygame.transform.scale(strip,(25,90)).convert_alpha()
stripXY = [[593,0],[593,152.5],[593,305],[593,457.5],[593,610]]
```

```python
explosion = pygame.image.load("data/images/Explosion.png")
explosion = pygame.transform.scale(explosion,(290,164)).convert_alpha()

fuel = pygame.image.load("data/images/Fuel.png")
fuel = pygame.transform.scale(fuel,(98,104)).convert_alpha()

comingCars,goingCars = [],[]
speedCC = [13,14,15,14,14,15,13,14,15]
speedGC = [8,6,7,5,8,7,8,6,8]

for i in range(1,10):
    CCi = pygame.image.load("data/images/Coming Cars/"+"CC"+str(i)+".png")
    CCi = pygame.transform.scale(CCi, (75, 158)).convert_alpha()
    comingCars.append([CCi,speedCC[i-1]])
    GCi                 =                pygame.image.load("data/images/Going
Cars/"+"GC"+str(i)+".png").convert_alpha()
    GCi = pygame.transform.scale(GCi,(75,158)).convert_alpha()
    goingCars.append([GCi,speedGC[i-1]])

def distance(carX,obstX,carY,obstY,isFuel = False):

    if not isFuel:
        carX += 75 # 75,75,37,79,55,130
        carY += 75
        obstX += 37
        obstY += 79

        return abs(carX - obstX) < 55 and abs(carY - obstY) < 130
    else:
        carX += 75
        carY += 75
        obstX += 98
        obstY += 104

        return abs(carX - obstX) < 70 and abs(carY - obstY) < 80

def textOnScreen(text,color,x,y,font):
    screenText = font.render(text,True,color)
    gameWindow.blit(screenText,[x,y])

def slowDown(carX,carY,dist,highscore):

    stripXY_ = [[593, 0], [593, 152.5], [593, 305], [593, 457.5], [593, 610]]
    exitScreen = False

    stripSpeed = 2

    start = time.time()
    while not exitScreen:
        if time.time() - start > 3:
            stripSpeed = 1
        if time.time() - start > 6:
            exitScreen = True
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                exitScreen = True

        gameWindow.fill((0,0,0))
        gameWindow.blit(leftDisp, (0, 0))
        textOnScreen("DISTANCE", (255, 255, 0), 27, 388, font1)
        textOnScreen(str(dist) + " Kms", (255, 0, 0), 56, 480, font1)
        textOnScreen("FUEL", (255, 255, 0), 73, 90, font1)
        textOnScreen(str(0.00) + ' %', (255, 0, 0), 75, 184, font1)
        gameWindow.blit(rightDisp, (950, 0))
        textOnScreen("HIGHSCORE", (255, 255, 0), 958, 236, font1)
        if(highscore < 10):
            disp = str(0) + str(highscore)
        else:
            disp = str(highscore)
        textOnScreen(disp + " Kms", (255, 0, 0), 1005, 342, font1)
        gameWindow.blit(road, (400, 0))
        gameWindow.blit(sand,(250,0))
        gameWindow.blit(sand,(800,0))

        for i in range(len(stripXY_)):
            stripXY_[i][1] += stripSpeed
            if stripXY_[i][1] > 700:
                stripXY_[i] = [593, -60]
        for i in range(len(treeLXY)):
            treeLXY[i][1] += stripSpeed
            if treeLXY[i][1] > 700:
                treeLXY[i] = [290,-60]
        for i in range(len(treeRXY)):
            treeRXY[i][1] += stripSpeed
            if treeRXY[i][1] > 700:
                treeRXY[i] = [760,-60]

        for X,Y in stripXY_:
            gameWindow.blit(strip,(X,Y))
        for treeX,treeY in treeLXY:
            gameWindow.blit(tree,(treeX,treeY))
        for treeX,treeY in treeRXY:
            gameWindow.blit(tree,(treeX,treeY))

        gameWindow.blit(car,(carX,carY))
        pygame.display.update()

def gameLoop():
```

```python
pygame.mixer.music.load("data/audios/lmn.mp3")
pygame.mixer.music.play()

time.sleep(1)

carX,carY = 625,540
drift = 4
carSpeedX = 0

obstacleXY = [[460,-10],[710,-300]]
c1,c2 = random.randint(0,8),random.randint(0,8)
if(c1 == c2):
    c1 = random.randint(0,8)

obstacleSpeed = [comingCars[c1][1],goingCars[c2][1]]
obstacles = [comingCars[c1][0],goingCars[c2][0]]

stripSpeed = 9

exitGame = False
gameOver = False
explode = False

fuelCount = 50
fuelX,fuelY = random.randint(420,620),-1000
fuelSpeed = 8
dist = 0

with open("data/Highscore.txt","r") as f:
    highscore = int(f.read())

slow = False
plotFuel = True

start1 = time.time()
start = [start1,start1]
start2 = start1
start3 = start1
start4 = start1
arrival = [2,3.5]

while not exitGame:
    if gameOver:

        if slow:
            slowDown(carX,carY,dist,highscore)
        time.sleep(2)
```

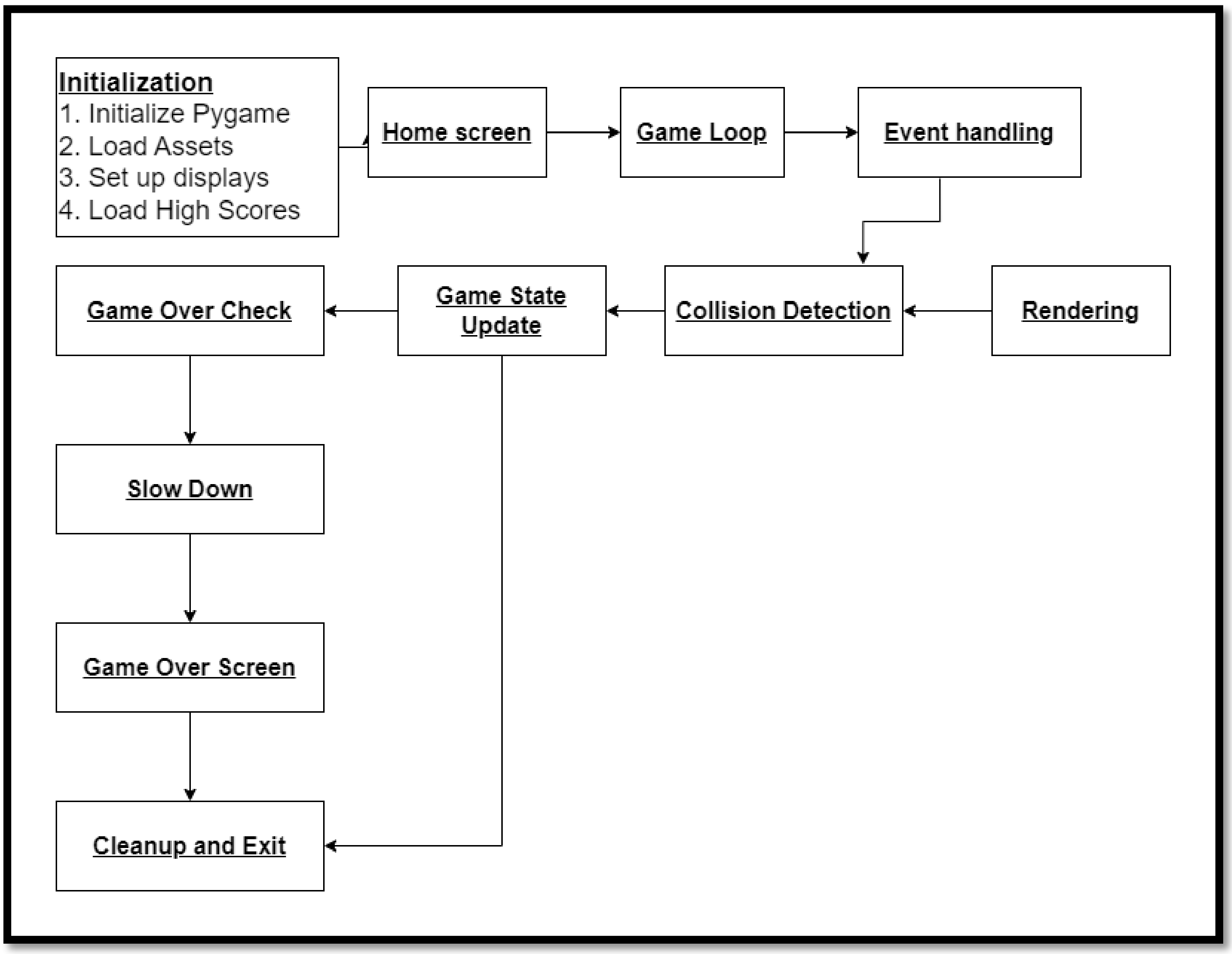
```python
            pygame.mixer.music.stop()
            pygame.mixer.music.load("data/audios/rtn.mp3")
            pygame.mixer.music.play()

            exitScreen = False
            go = pygame.image.load("data/images/GameOver.png")
            go = pygame.transform.scale(go,(1239,752)).convert_alpha()

            with open("data/Highscore.txt","w") as f:
                f.write(str(highscore))

            while not exitScreen:
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        exitScreen = True
                    elif event.type == pygame.KEYDOWN:
                        if event.key == pygame.K_RETURN:
                            pygame.mixer.music.stop()
                            homeScreen()
                gameWindow.fill((0,0,0))
                gameWindow.blit(go,(0,0))
                if(dist < 10):
                    disp = str(0) + str(dist)
                else:
                    disp = str(dist)
                textOnScreen(disp,(255,0,0),540,429,font1)
                pygame.display.update()
                clock.tick(fps)

            pygame.quit()
            quit()
        else:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    exitGame = True
                elif event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RIGHT:
                        carSpeedX = drift
                    elif event.key == pygame.K_LEFT:
                        carSpeedX = -drift
                    elif event.key == pygame.K_a:
                        obstacleXY[0][0] -= 20
                    elif event.key == pygame.K_d:
                        obstacleXY[1][0] += 20

            carX += carSpeedX
            fuelY += fuelSpeed

            if time.time() - start4 >= 2:
```

```
    dist += 1
    if dist > highscore:
        highscore = dist
    start4 = time.time()

if time.time() - start2 >= 3:
    fuelCount -= 5
    start2 = time.time()

if distance(carX,fuelX,carY,fuelY,True) and plotFuel:
    plotFuel = False
    fuelCount += 20

for i in range(len(obstacleXY)):
    obstacleXY[i][1] += obstacleSpeed[i]

fuelper = fuelCount/50
if fuelper >= 1:
    fuelper = 1

gameWindow.fill((0,0,0))
gameWindow.blit(leftDisp,(0,0))
textOnScreen("DISTANCE", (255, 255, 0),27,388,font1)
if(dist < 10):
    disp = str(0) + str(dist)
else:
    disp = str(dist)
textOnScreen(disp + " Kms",(255,0,0),56,480,font1)
textOnScreen("FUEL",(255,255,0),73,90,font1)
textOnScreen(str(fuelper*100) + ' %',(255,0,0),60,184,font1)
gameWindow.blit(rightDisp, (950, 0))
textOnScreen("HIGHSCORE",(255,255,0),958,236,font1)
if(highscore < 10):
    disp = str(0) + str(highscore)
else:
    disp = str(highscore)
textOnScreen(disp + " Kms",(255,0,0),1005,342,font1)
gameWindow.blit(road,(400,0))
gameWindow.blit(sand, (250, 0))
gameWindow.blit(sand, (800, 0))

if fuelCount == 0:
    gameOver = True
    slow = True

if carX > 720 or carX < 330:
    pygame.mixer.music.load("data/audios/Crash.mp3")
    pygame.mixer.music.play()
    gameOver = True
```

```
                explode = True

            for i in range(len(obstacleXY)):
                if distance(carX,obstacleXY[i][0],carY,obstacleXY[i][1]):
                    pygame.mixer.music.load("data/audios/Crash.mp3")
                    pygame.mixer.music.play()
                    gameOver = True
                    explode = True
                    break
            for i in range(len(stripXY)):
                stripXY[i][1] += stripSpeed
                if stripXY[i][1] > 700:
                    stripXY[i] = [593,-60]
            for i in range(len(treeLXY)):
                treeLXY[i][1] += stripSpeed
                if treeLXY[i][1] > 700:
                    treeLXY[i] = [290, -60]
            for i in range(len(treeRXY)):
                treeRXY[i][1] += stripSpeed
                if treeRXY[i][1] > 700:
                    treeRXY[i] = [760, -60]

            for stripX,stripY in stripXY:
                gameWindow.blit(strip,(stripX,stripY))

            if fuelY < 750:
                if plotFuel:
                    gameWindow.blit(fuel,(fuelX,fuelY))

            gameWindow.blit(car,(carX,carY))

            for i in range(len(obstacleXY)):
                if obstacleXY[i][1] < 750:
                    gameWindow.blit(obstacles[i],(obstacleXY[i][0],
obstacleXY[i][1]))

            for treeX, treeY in treeLXY:
                gameWindow.blit(tree, (treeX, treeY))
            for treeX, treeY in treeRXY:
                gameWindow.blit(tree, (treeX, treeY))

            if time.time() - start[0] >= arrival[0]:
                x = random.randint(430,530)
                x+=3
                obstacleXY[0] = [x,-10]
                c1 = random.randint(0,8)
                obstacles[0] = comingCars[c1][0]
                obstacleSpeed[0] = comingCars[c1][1]
                start[0] = time.time()
            if time.time() - start[1] >= arrival[1]:
                x = random.randint(620,710)
                x-=3
                obstacleXY[1] = [x,-10]
                c2 = random.randint(0,8)
                obstacles[1] = goingCars[c2][0]
                obstacleSpeed[1] = goingCars[c2][1]
                start[1] = time.time()
            if time.time() - start3 >= 15:
                fuelX,fuelY = random.randint(420,710),-500
                plotFuel = True
                start3 = time.time()
            if explode:
                gameWindow.blit(explosion,(carX - 63,carY))

            pygame.display.update()
            clock.tick(fps)


    pygame.quit()
    quit()

def homeScreen():

    pygame.mixer.music.load("data/audios/rtn.mp3")
    pygame.mixer.music.play()

    if not os.path.exists("data/Highscore.txt"):
        with open("data/Highscore.txt","w") as f:
            f.write("0")
            highscore = 0
    else:
        with open("data/Highscore.txt","r") as f:
            highscore = int(f.read())


    background = pygame.image.load("data/images/Background.png")
    background = pygame.transform.scale(background,(1213,760)).convert_alpha()

    exitScreen = False
    while not exitScreen:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                exitScreen = True
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RETURN:
                    pygame.mixer.music.stop()
                    gameLoop()
        gameWindow.blit(background,(-6,-32))
        if(highscore < 10):
            disp = str(0) + str(highscore)
        else:
            disp = str(highscore)
        textOnScreen(disp,(255,0,0),980,9,font1)
        pygame.display.update()
        clock.tick(fps)

    pygame.quit()
    quit()


homeScreen()
```

Game Architecture

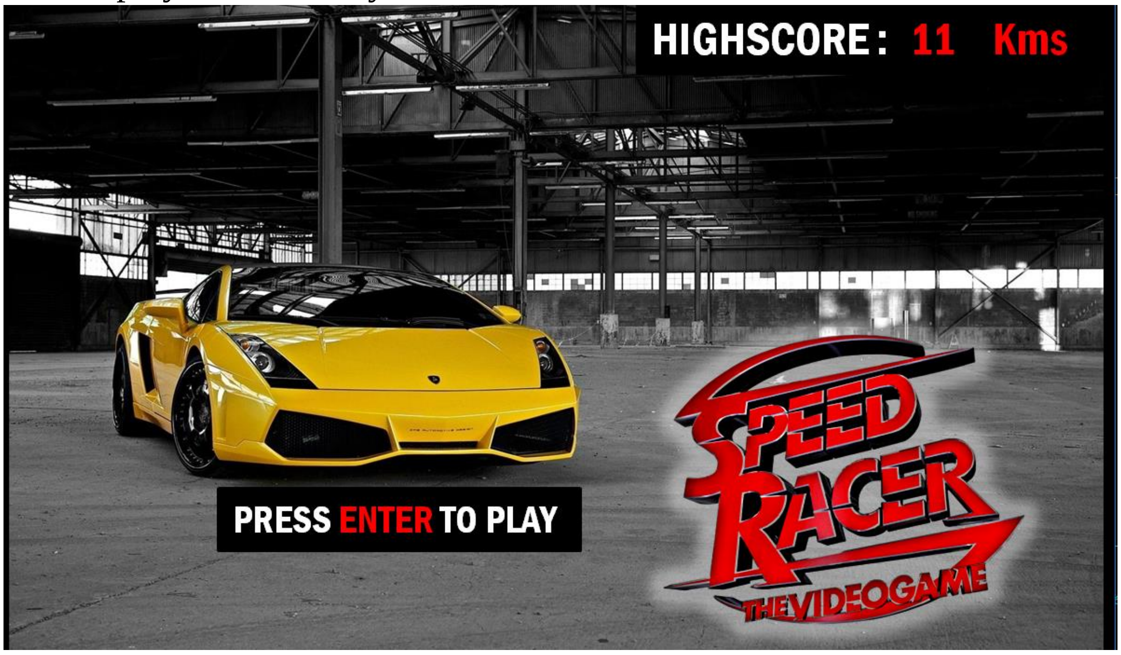
The game architecture comprises player-controlled car navigation on a road filled with obstacles and fuel pickups, with visual and auditory feedback managed through Pygame libraries and user input controlled via keyboard events.

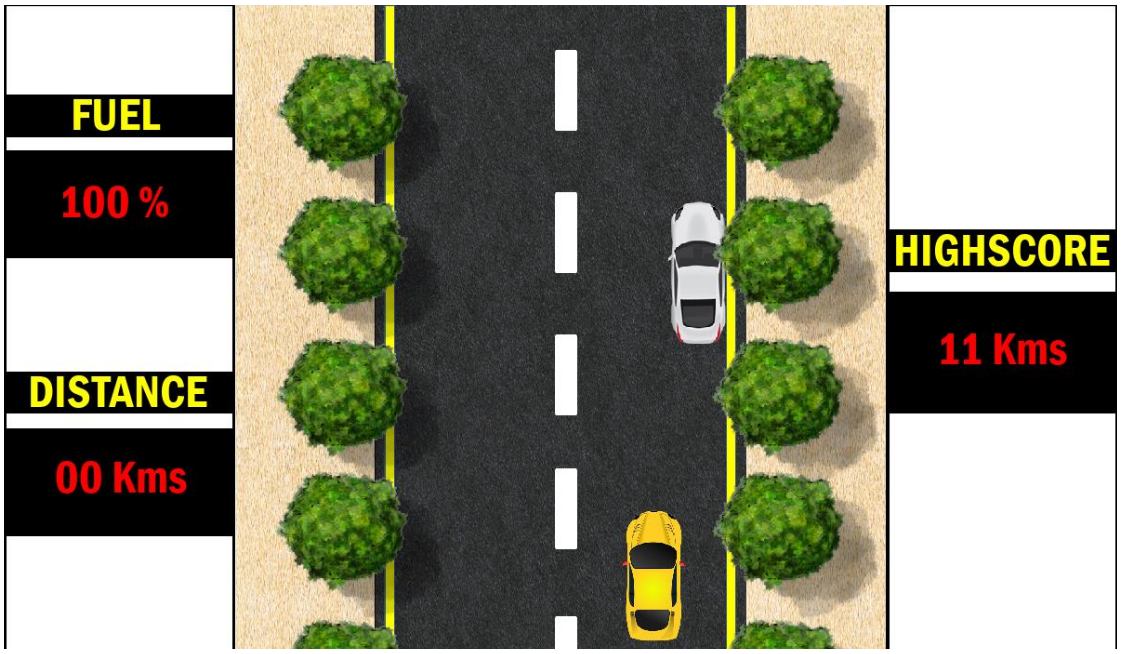


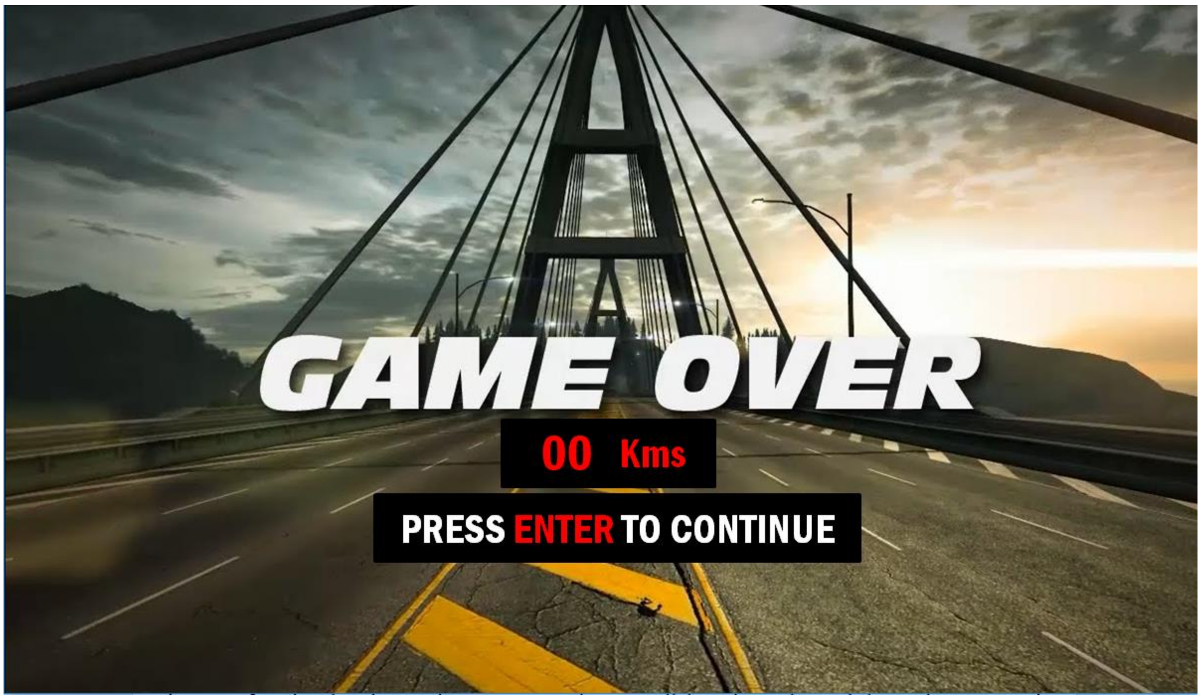
8. Results

Output Screenshot

The game successfully recreates the core mechanics of Car. The Car moves smoothly as intended and other assets like trees, coming cars, going cars, fuel and other surroundings are correctly and timely rendered. The music behind the game also plays successfully.






GAME OVER
00 Kms
PRESS ENTER TO CONTINUE

9. Discussion

The discussion of the project typically involves an analysis of various aspects, including the game's design, development process, challenges encountered, lessons learned, and potential future enhancements.

10. Conclusion

In conclusion, the Speed Racer project has been a rewarding journey in game development, combining learning, creativity, and technical skill. Through the creation of a 2D racing game using Python and Pygame, participants gained practical experience in programming, design, and problem-solving. The resulting game offers an engaging experience for players and serves as a testament to the dedication and talent of its creators. While the project marks a significant achievement, it also opens doors to future opportunities for expansion and enhancement

11. Future Work

Level Design: Introduce multiple levels with increasing difficulty.
Additional Features: Add features such as power-ups, different car models, and more complex obstacle patterns.
Multiplayer Mode: Implement a multiplayer mode where players can race against each other.

12. References

Pygame Documentation. (n.d.). Retrieved from https://www.pygame.org/docs/

https://www.geeksforgeeks.org/car-race-game-in-pygame/

https://www.youtube.com/watch?v=XON1IA8MxQw&list=PLz7mmheDeooW795HsNzkLS6TFQSa4Idtw

13. Decision Modeling in Game Design

Decision modeling in this game includes:

Player Input: Determine the keyboard controls for car movement and interaction with game elements.
Obstacle Behavior: Define the movement patterns and spawning logic for obstacles to create challenging gameplay.
Fuel Mechanics: Establish rules for fuel pickup appearance, depletion rate, and impact on gameplay dynamics.
Scoring System: Design a scoring mechanism based on distance traveled and fuel collected to incentivize player engagement.
Game Over Conditions: Specify conditions such as collision with obstacles or fuel depletion that trigger the end of the game.
User Interface Design: Plan the layout and elements of the user interface to convey essential information effectively.
Art Style and Theme: Decide on the visual style, theme, and aesthetics to create a cohesive and engaging game world.
Audio Integration: Determine the use of sound effects and background music to enhance immersion and feedback for player actions.
Performance Optimization: Identify areas for optimization to ensure smooth gameplay performance on various devices.
Testing Strategy: Establish testing procedures to evaluate game mechanics, balance, and overall player experience.