

CSE354 – Distributed Computing

Project Phase One

|  |  |
| --- | --- |
| Name | ID |
| AbdulRaouf Monir Kamal Mahmoud | 19P4442 |
| Ahmed Sameh Mohamed Mourad | 19P5861 |
| Elsaeed Ahmed Elsaeed | 19P1087 |
| Tasneem Hisham Sayed | 19P4152 |

GitHub Repo Link: <https://github.com/Ahmed-Sameh-MM/Distributed-Systems-Project>

Table of Contents

[1. Car Racing Game Implementation using Pygame: 3](#_Toc136465396)

[1.1 Pygame Code: 3](#_Toc136465397)

[1.2 Screen Shot of the pygame window: 10](#_Toc136465398)

[2.0 Chat Using Socket.io 10](#_Toc136465399)

[2.1 Server 11](#_Toc136465400)

[2.2 Client 12](#_Toc136465401)

[2.3 Screenshots 13](#_Toc136465402)

# Car Racing Game Implementation using Pygame:

## Pygame Code:

import pygame

from time import sleep

First, we import pygame module for the following reasons:

1. Control displaying the background image and displaying the racing cars.
2. Handle the events by the user to move the cars.

Second, we import time module which is used later which is used to enter the pygame process into a sleep mode when two cars crash.

pygame.font.init()

WIDTH, HEIGHT = 800, 600

WIN = pygame.display.set\_mode((WIDTH, HEIGHT))

myRoad = pygame.transform.scale(pygame.image.load("img/back\_ground.jpg"), (WIDTH, HEIGHT))

BG\_SPEED = 3

CAR\_WIDTH = 49

CAR\_HEIGHT = 100

CAR\_VEL = 5

FONT = pygame.font.SysFont("comicsans", 30)

These are some constants defined before the main code, which are mainly used for setting sizes for objects or images like racing cars, background image, etc.

There are also some other constants like CAR\_VEL which is used to move the car to any direction specified by user key event with 5 units in the specified direction.

Finally, we have the pygame font module which is used later to display some text like on the screen.

def drawGame():

    WIN.blit(myRoad, (0, 0))

def move(object, x\_coord, y\_coord):

    WIN.blit(object, (x\_coord, y\_coord))

def updateDisplay():

    pygame.display.update()

The above three functions are used for moving the new image into its new position using the *blit* provided pygame module, and *pygame.display.update()*

Is used for rendering the screen after calling the *blit* method.

run = True

    myCar = pygame.image.load("img/car.png")

    myCar\_x\_coordinate = 200

    myCar\_y\_coordinate = HEIGHT - CAR\_HEIGHT

    myRoad\_x1\_coordinate = 0

    myRoad\_x2\_coordinate = 0

    myRoad\_y1\_coordinate = 0

    myRoad\_y2\_coordinate = -600

    enemy\_car\_1 = pygame.image.load("img/enemy\_car\_1.png")

    enemy\_car\_2 = pygame.image.load("img/enemy\_car\_2.png")

    enemyOne\_x\_coordinate = 370

    enemyOne\_y\_coordinate = HEIGHT - CAR\_HEIGHT - 10

    enemyTwo\_x\_coordinate = 550

    enemyTwo\_y\_coordinate = HEIGHT - CAR\_HEIGHT - 10

    clock = pygame.time.Clock()

    drawGame()

    move(myCar, myCar\_x\_coordinate, myCar\_y\_coordinate)

    move(enemy\_car\_1, enemyOne\_x\_coordinate, enemyOne\_y\_coordinate)

    move(enemy\_car\_2, enemyTwo\_x\_coordinate, enemyTwo\_y\_coordinate)

For the above few lines of code, we have run = true which must be present in pygame script, which is used later in the while loop, and its function is to keep the code running while the QUIT button has not been pressed, or the game has ended.

Then we have some variables like myCar and others for loading the image of the racing cars into the game. Some other variables are also used for determining the starting x and y coordinates of each racing car. Afterwards, the image of the background, and the racing cars are loaded into the screen using drawGame() and move() methods. The most important part of this code is the clock module which, in short, can be used to configure the time between processing each event provided by the user.

while run:

        clock.tick(60)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                run = False

                break

        keys = pygame.key.get\_pressed()

        if keys[pygame.K\_LEFT] and myCar\_x\_coordinate - CAR\_VEL >= 0:

            myCar\_x\_coordinate -= CAR\_VEL

        if keys[pygame.K\_RIGHT] and myCar\_x\_coordinate + CAR\_VEL + CAR\_WIDTH <= WIDTH:

            myCar\_x\_coordinate += CAR\_VEL

        if keys[pygame.K\_UP] and myCar\_y\_coordinate - CAR\_VEL >= 0:

            myCar\_y\_coordinate -= CAR\_VEL

        if keys[pygame.K\_DOWN] and myCar\_y\_coordinate + CAR\_VEL + CAR\_HEIGHT <= HEIGHT:

            myCar\_y\_coordinate += CAR\_VEL

        move(myRoad, myRoad\_x1\_coordinate, myRoad\_y1\_coordinate)

        move(myRoad, myRoad\_x2\_coordinate, myRoad\_y2\_coordinate)

        myRoad\_y1\_coordinate += BG\_SPEED

        myRoad\_y2\_coordinate += BG\_SPEED

        if myRoad\_y1\_coordinate >= HEIGHT:

            myRoad\_y1\_coordinate = -600

        if myRoad\_y2\_coordinate >= HEIGHT:

            myRoad\_y2\_coordinate = -600

Here we have the while loop which runs forever until one of the events mentioned above occurs. We chose to set the number of ticks per second to be 60 as we thought it would provide normal gameplay (normal fps). Then we have the for loop that captures the event that QUIT button has been clicked.

keys = pygame.key.get\_pressed()

        if keys[pygame.K\_LEFT] and myCar\_x\_coordinate - CAR\_VEL >= 0:

            myCar\_x\_coordinate -= CAR\_VEL

        if keys[pygame.K\_RIGHT] and myCar\_x\_coordinate + CAR\_VEL + CAR\_WIDTH <= WIDTH:

            myCar\_x\_coordinate += CAR\_VEL

        if keys[pygame.K\_UP] and myCar\_y\_coordinate - CAR\_VEL >= 0:

            myCar\_y\_coordinate -= CAR\_VEL

        if keys[pygame.K\_DOWN] and myCar\_y\_coordinate + CAR\_VEL + CAR\_HEIGHT <= HEIGHT:

            myCar\_y\_coordinate += CAR\_VEL

For this part, we capture the event (key press) provided by the user, and check for some conditions regarding the position of the racing car, these conditions check for the key pressed, and makes sure that the racing after changing its x or y coordinates, does not cross the boundaries of the screen. After applying the changes to the x and y coordinates of the racing cars, the move() is called to apply these changes and render to the screen the new coordinates.

myRoad\_y1\_coordinate += BG\_SPEED

        myRoad\_y2\_coordinate += BG\_SPEED

        if myRoad\_y1\_coordinate >= HEIGHT:

            myRoad\_y1\_coordinate = -600

        if myRoad\_y2\_coordinate >= HEIGHT:

            myRoad\_y2\_coordinate = -600

This part of the code is responsible for rendering the background the image positions as if the cars are accelerating.

if myCar\_x\_coordinate + CAR\_WIDTH > enemyOne\_x\_coordinate \

           and myCar\_x\_coordinate < enemyOne\_x\_coordinate + CAR\_WIDTH \

           and myCar\_y\_coordinate < enemyTwo\_y\_coordinate :

            text = FONT.render("Game OVER!!", True, (255, 255, 255))

            WIN.blit(text, (400 - text.get\_width() // 2, 240 - text.get\_height() // 2))

            move(myCar, myCar\_x\_coordinate, myCar\_y\_coordinate)

            move(enemy\_car\_1, enemyOne\_x\_coordinate, enemyOne\_y\_coordinate)

            move(enemy\_car\_2, enemyTwo\_x\_coordinate, enemyTwo\_y\_coordinate)

            updateDisplay()

            sleep(1)

            drawGame()

            myCar\_x\_coordinate = 200

            myCar\_y\_coordinate = HEIGHT - CAR\_HEIGHT

            move(myCar, 200, HEIGHT - CAR\_HEIGHT)

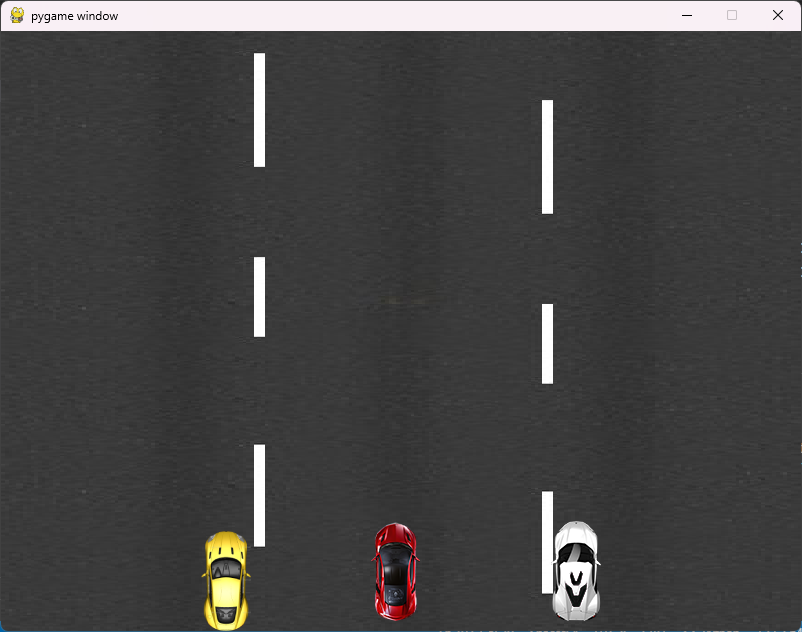
            move(enemy\_car\_1, 370, HEIGHT - CAR\_HEIGHT - 10)

            move(enemy\_car\_2, 550, HEIGHT - CAR\_HEIGHT - 10)

            updateDisplay()

The last part of the pygame code, is the part that checks for crashes. However, it is not yet fully implemented; it just checks if the x-coordinate of the first two cars are equal or not, and if they are, it will be considered, and the text “GAME OVER” will be over and the game restarts again, and the move() method is called to move the cars to its respective starting point once again.

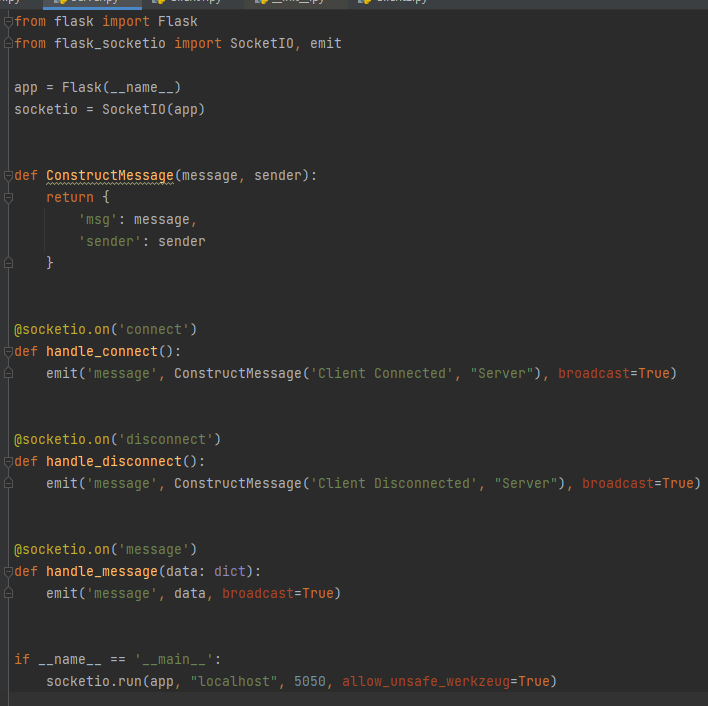
## 1.2 Screen Shot of the pygame window:



# 2.0 Chat Using Socket.io

Set up a basic Flask-SocketIO server that handles connection, disconnection, and message events. When a client connects or disconnects, a predefined message is emitted to all connected clients. Similarly, when a client sends a message event, the received data is broadcasted to all connected clients.

## 2.1 Server



## 2.2 Client



## 2.3 Screenshots

