**PROJECT REPORT**

**ON**

**Traffic Management System**

****

REPORT SUBMITTED

TO

VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE

FOR THE PBL OF **PYTHON FOR ENGINEERS**

IN

**ENGINEERING AND APPLIED SCIENCE DEPARTMENT**

By

**Mihir Patwardhan- 268/22111346**

**Aditya Adinath Patil - 269/ 22110016**

**Krishna Gupta- 270/22110152**

**Triambak Kapoor – 271/22110153**

**Class: F.Y BTECH Division: b Batch: b3**

**Batch teacher-: Aparna Barbadekar**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Contents** | **Page no.** |
| **1** | **Source Code** | **03 – 16** |
| **2** | **Output** | **17** |

**Source Code**

import random  
import time  
import threading  
import pygame  
import sys  
  
  
# Default values of signal timers  
defaultGreen = {0: 10, 1: 10, 2: 10, 3: 10}  
defaultRed = 50  
defaultYellow = 5  
  
signals = []  
noOfSignals = 4  
currentGreen = 0 # Indicates which signal is green currently  
nextGreen = (currentGreen + 1) % noOfSignals # Indicates which signal will turn green next  
currentYellow = 0 # Indicates whether yellow signal is on or off  
  
speeds = {'car': 1.0, 'bus': 0.6, 'truck': 0.3, 'bike': 1.5} # average speeds of vehicles  
  
# Coordinates of vehicles' start  
x = {'right': [0, 0, 0], 'down': [755, 727, 697], 'left': [1400, 1400, 1400], 'up': [602, 627, 657]}  
y = {'right': [348, 370, 398], 'down': [0, 0, 0], 'left': [498, 466, 436], 'up': [800, 800, 800]}  
  
vehicles = {'right': {0: [], 1: [], 2: [], 'crossed': 0}, 'down': {0: [], 1: [], 2: [], 'crossed': 0},  
 'left': {0: [], 1: [], 2: [], 'crossed': 0}, 'up': {0: [], 1: [], 2: [], 'crossed': 0}}  
vehicleTypes = {0: 'car', 1: 'bus', 2: 'truck', 3: 'bike'}  
directionNumbers = {0: 'right', 1: 'down', 2: 'left', 3: 'up'}  
  
# Coordinates of signal image, timer, and vehicle count  
signalCoods = [(530, 230), (810, 230), (810, 570), (530, 570)]  
signalTimerCoods = [(530, 210), (810, 210), (810, 550), (530, 550)]  
  
# Coordinates of stop lines  
stopLines = {'right': 590, 'down': 330, 'left': 800, 'up': 535}  
defaultStop = {'right': 580, 'down': 320, 'left': 810, 'up': 545}  
  
# Gap between vehicles  
stoppingGap = 25 # stopping gap  
movingGap = 25 # moving gap  
  
# set allowed vehicle types here  
allowedVehicleTypes = {'car': True, 'bus': True, 'truck': True, 'bike': True}  
allowedVehicleTypesList = []  
vehiclesTurned = {'right': {1: [], 2: []}, 'down': {1: [], 2: []}, 'left': {1: [], 2: []}, 'up': {1: [], 2: []}}  
vehiclesNotTurned = {'right': {1: [], 2: []}, 'down': {1: [], 2: []}, 'left': {1: [], 2: []}, 'up': {1: [], 2: []}}  
rotationAngle = 3  
mid = {'right': {'x': 705, 'y': 445}, 'down': {'x': 695, 'y': 450}, 'left': {'x': 695, 'y': 425},  
 'up': {'x': 695, 'y': 400}}  
  
  
randomGreenSignalTimer = True  
randomGreenSignalTimerRange = [10, 15]  
  
timeElapsed = 0  
simulationTime = 300  
timeElapsedCoods = (1100, 50)  
vehicleCountTexts = ["0", "0", "0", "0"]  
vehicleCountCoods = [(480, 210), (880, 210), (880, 550), (480, 550)]  
  
pygame.init()  
simulation = pygame.sprite.Group()  
  
  
class TrafficSignal:  
 def \_\_init\_\_(self, red, yellow, green):  
 self.red = red  
 self.yellow = yellow  
 self.green = green  
 self.signalText = ""  
  
  
class Vehicle(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, lane, vehicleClass, direction\_number, direction, will\_turn):  
 pygame.sprite.Sprite.\_\_init\_\_(self)  
 self.lane = lane  
 self.vehicleClass = vehicleClass  
 self.speed = speeds[vehicleClass]  
 self.direction\_number = direction\_number  
 self.direction = direction  
 self.x = x[direction][lane]  
 self.y = y[direction][lane]  
 self.crossed = 0  
 self.willTurn = will\_turn  
 self.turned = 0  
 self.rotateAngle = 0  
 vehicles[direction][lane].append(self)  
 self.index = len(vehicles[direction][lane]) - 1  
 self.crossedIndex = 0  
 path = "images/" + direction + "/" + vehicleClass + ".png"  
 self.originalImage = pygame.image.load(path)  
 self.image = pygame.image.load(path)  
  
 if (len(vehicles[direction][lane]) > 1 and vehicles[direction][lane][self.index - 1].crossed == 0):  
 if (direction == 'right'):  
 self.stop = vehicles[direction][lane][self.index - 1].stop  
  
 elif (direction == 'left'):  
 self.stop = vehicles[direction][lane][self.index - 1].stop  
  
 elif (direction == 'down'):  
 self.stop = vehicles[direction][lane][self.index - 1].stop  
  
 elif (direction == 'up'):  
 self.stop = vehicles[direction][lane][self.index - 1].stop  
  
 else:  
 self.stop = defaultStop[direction]  
  
 # Set new starting and stopping coordinate  
 if (direction == 'right'):  
 temp = self.image.get\_rect().width + stoppingGap  
 x[direction][lane] -= temp  
 elif (direction == 'left'):  
 temp = self.image.get\_rect().width + stoppingGap  
 x[direction][lane] += temp  
 elif (direction == 'down'):  
 temp = self.image.get\_rect().height + stoppingGap  
 y[direction][lane] -= temp  
 elif (direction == 'up'):  
 temp = self.image.get\_rect().height + stoppingGap  
 y[direction][lane] += temp  
 simulation.add(self)  
  
 def render(self, screen):  
 screen.blit(self.image, (self.x, self.y))  
  
 def move(self):  
 if (self.direction == 'right'):  
 if (self.crossed == 0 and self.x + self.image.get\_rect().width > stopLines[self.direction]):  
 self.crossed = 1  
 vehicles[self.direction]['crossed'] += 1  
 if (self.willTurn == 0):  
 vehiclesNotTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesNotTurned[self.direction][self.lane]) - 1  
 if (self.willTurn == 1):  
 if (self.lane == 1):  
 if (self.crossed == 0 or self.x + self.image.get\_rect().width < stopLines[self.direction] + 40):  
 if ((self.x + self.image.get\_rect().width <= self.stop or (  
 currentGreen == 0 and currentYellow == 0) or self.crossed == 1) and (  
 self.index == 0 or self.x + self.image.get\_rect().width < (  
 vehicles[self.direction][self.lane][self.index - 1].x - movingGap) or  
 vehicles[self.direction][self.lane][self.index - 1].turned == 1)):  
 self.x += self.speed  
 else:  
 if (self.turned == 0):  
 self.rotateAngle += rotationAngle  
 self.image = pygame.transform.rotate(self.originalImage, self.rotateAngle)  
 self.x += 2.4  
 self.y -= 2.8  
 if (self.rotateAngle == 90):  
 self.turned = 1  
 vehiclesTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1  
 else:  
 if (self.crossedIndex == 0 or (self.y > (  
 vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].y +  
 vehiclesTurned[self.direction][self.lane][  
 self.crossedIndex - 1].image.get\_rect().height + movingGap))):  
 self.y -= self.speed  
 elif (self.lane == 2):  
 if (self.crossed == 0 or self.x + self.image.get\_rect().width < mid[self.direction]['x']):  
 if ((self.x + self.image.get\_rect().width <= self.stop or (  
 currentGreen == 0 and currentYellow == 0) or self.crossed == 1) and (  
 self.index == 0 or self.x + self.image.get\_rect().width < (  
 vehicles[self.direction][self.lane][self.index - 1].x - movingGap) or  
 vehicles[self.direction][self.lane][self.index - 1].turned == 1)):  
 self.x += self.speed  
 else:  
 if (self.turned == 0):  
 self.rotateAngle += rotationAngle  
 self.image = pygame.transform.rotate(self.originalImage, -self.rotateAngle)  
 self.x += 2  
 self.y += 1.8  
 if (self.rotateAngle == 90):  
 self.turned = 1  
 vehiclesTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1  
 else:  
 if (self.crossedIndex == 0 or ((self.y + self.image.get\_rect().height) < (  
 vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].y - movingGap))):  
 self.y += self.speed  
 else:  
 if (self.crossed == 0):  
 if ((self.x + self.image.get\_rect().width <= self.stop or (  
 currentGreen == 0 and currentYellow == 0)) and (  
 self.index == 0 or self.x + self.image.get\_rect().width < (  
 vehicles[self.direction][self.lane][self.index - 1].x - movingGap))):  
 self.x += self.speed  
 else:  
 if ((self.crossedIndex == 0) or (self.x + self.image.get\_rect().width < (  
 vehiclesNotTurned[self.direction][self.lane][self.crossedIndex - 1].x - movingGap))):  
 self.x += self.speed  
 elif (self.direction == 'down'):  
 if (self.crossed == 0 and self.y + self.image.get\_rect().height > stopLines[self.direction]):  
 self.crossed = 1  
 vehicles[self.direction]['crossed'] += 1  
 if (self.willTurn == 0):  
 vehiclesNotTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesNotTurned[self.direction][self.lane]) - 1  
 if (self.willTurn == 1):  
 if (self.lane == 1):  
 if (self.crossed == 0 or self.y + self.image.get\_rect().height < stopLines[self.direction] + 50):  
 if ((self.y + self.image.get\_rect().height <= self.stop or (  
 currentGreen == 1 and currentYellow == 0) or self.crossed == 1) and (  
 self.index == 0 or self.y + self.image.get\_rect().height < (  
 vehicles[self.direction][self.lane][self.index - 1].y - movingGap) or  
 vehicles[self.direction][self.lane][self.index - 1].turned == 1)):  
 self.y += self.speed  
 else:  
 if (self.turned == 0):  
 self.rotateAngle += rotationAngle  
 self.image = pygame.transform.rotate(self.originalImage, self.rotateAngle)  
 self.x += 1.2  
 self.y += 1.8  
 if (self.rotateAngle == 90):  
 self.turned = 1  
 vehiclesTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1  
 else:  
 if (self.crossedIndex == 0 or ((self.x + self.image.get\_rect().width) < (  
 vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].x - movingGap))):  
 self.x += self.speed  
 elif (self.lane == 2):  
 if (self.crossed == 0 or self.y + self.image.get\_rect().height < mid[self.direction]['y']):  
 if ((self.y + self.image.get\_rect().height <= self.stop or (  
 currentGreen == 1 and currentYellow == 0) or self.crossed == 1) and (  
 self.index == 0 or self.y + self.image.get\_rect().height < (  
 vehicles[self.direction][self.lane][self.index - 1].y - movingGap) or  
 vehicles[self.direction][self.lane][self.index - 1].turned == 1)):  
 self.y += self.speed  
 else:  
 if (self.turned == 0):  
 self.rotateAngle += rotationAngle  
 self.image = pygame.transform.rotate(self.originalImage, -self.rotateAngle)  
 self.x -= 2.5  
 self.y += 2  
 if (self.rotateAngle == 90):  
 self.turned = 1  
 vehiclesTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1  
 else:  
 if (self.crossedIndex == 0 or (self.x > (  
 vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].x +  
 vehiclesTurned[self.direction][self.lane][  
 self.crossedIndex - 1].image.get\_rect().width + movingGap))):  
 self.x -= self.speed  
 else:  
 if (self.crossed == 0):  
 if ((self.y + self.image.get\_rect().height <= self.stop or (  
 currentGreen == 1 and currentYellow == 0)) and (  
 self.index == 0 or self.y + self.image.get\_rect().height < (  
 vehicles[self.direction][self.lane][self.index - 1].y - movingGap))):  
 self.y += self.speed  
 else:  
 if ((self.crossedIndex == 0) or (self.y + self.image.get\_rect().height < (  
 vehiclesNotTurned[self.direction][self.lane][self.crossedIndex - 1].y - movingGap))):  
 self.y += self.speed  
 elif (self.direction == 'left'):  
 if (self.crossed == 0 and self.x < stopLines[self.direction]):  
 self.crossed = 1  
 vehicles[self.direction]['crossed'] += 1  
 if (self.willTurn == 0):  
 vehiclesNotTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesNotTurned[self.direction][self.lane]) - 1  
 if (self.willTurn == 1):  
 if (self.lane == 1):  
 if (self.crossed == 0 or self.x > stopLines[self.direction] - 70):  
 if ((self.x >= self.stop or (  
 currentGreen == 2 and currentYellow == 0) or self.crossed == 1) and (  
 self.index == 0 or self.x > (vehicles[self.direction][self.lane][self.index - 1].x +  
 vehicles[self.direction][self.lane][  
 self.index - 1].image.get\_rect().width + movingGap) or  
 vehicles[self.direction][self.lane][self.index - 1].turned == 1)):  
 self.x -= self.speed  
 else:  
 if (self.turned == 0):  
 self.rotateAngle += rotationAngle  
 self.image = pygame.transform.rotate(self.originalImage, self.rotateAngle)  
 self.x -= 1  
 self.y += 1.2  
 if (self.rotateAngle == 90):  
 self.turned = 1  
 vehiclesTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1  
 else:  
 if (self.crossedIndex == 0 or ((self.y + self.image.get\_rect().height) < (  
 vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].y - movingGap))):  
 self.y += self.speed  
 elif (self.lane == 2):  
 if (self.crossed == 0 or self.x > mid[self.direction]['x']):  
 if ((self.x >= self.stop or (  
 currentGreen == 2 and currentYellow == 0) or self.crossed == 1) and (  
 self.index == 0 or self.x > (vehicles[self.direction][self.lane][self.index - 1].x +  
 vehicles[self.direction][self.lane][  
 self.index - 1].image.get\_rect().width + movingGap) or  
 vehicles[self.direction][self.lane][self.index - 1].turned == 1)):  
 self.x -= self.speed  
 else:  
 if (self.turned == 0):  
 self.rotateAngle += rotationAngle  
 self.image = pygame.transform.rotate(self.originalImage, -self.rotateAngle)  
 self.x -= 1.8  
 self.y -= 2.5  
 if (self.rotateAngle == 90):  
 self.turned = 1  
 vehiclesTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1  
 else:  
 if (self.crossedIndex == 0 or (self.y > (  
 vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].y +  
 vehiclesTurned[self.direction][self.lane][  
 self.crossedIndex - 1].image.get\_rect().height + movingGap))):  
 self.y -= self.speed  
 else:  
 if (self.crossed == 0):  
 if ((self.x >= self.stop or (currentGreen == 2 and currentYellow == 0)) and (  
 self.index == 0 or self.x > (  
 vehicles[self.direction][self.lane][self.index - 1].x + vehicles[self.direction][self.lane][  
 self.index - 1].image.get\_rect().width + movingGap))):  
 self.x -= self.speed  
 else:  
 if ((self.crossedIndex == 0) or (self.x > (  
 vehiclesNotTurned[self.direction][self.lane][self.crossedIndex - 1].x +  
 vehiclesNotTurned[self.direction][self.lane][  
 self.crossedIndex - 1].image.get\_rect().width + movingGap))):  
 self.x -= self.speed  
 elif (self.direction == 'up'):  
 if (self.crossed == 0 and self.y < stopLines[self.direction]):  
 self.crossed = 1  
 vehicles[self.direction]['crossed'] += 1  
 if (self.willTurn == 0):  
 vehiclesNotTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesNotTurned[self.direction][self.lane]) - 1  
 if (self.willTurn == 1):  
 if (self.lane == 1):  
 if (self.crossed == 0 or self.y > stopLines[self.direction] - 60):  
 if ((self.y >= self.stop or (  
 currentGreen == 3 and currentYellow == 0) or self.crossed == 1) and (  
 self.index == 0 or self.y > (vehicles[self.direction][self.lane][self.index - 1].y +  
 vehicles[self.direction][self.lane][  
 self.index - 1].image.get\_rect().height + movingGap) or  
 vehicles[self.direction][self.lane][self.index - 1].turned == 1)):  
 self.y -= self.speed  
 else:  
 if (self.turned == 0):  
 self.rotateAngle += rotationAngle  
 self.image = pygame.transform.rotate(self.originalImage, self.rotateAngle)  
 self.x -= 2  
 self.y -= 1.2  
 if (self.rotateAngle == 90):  
 self.turned = 1  
 vehiclesTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1  
 else:  
 if (self.crossedIndex == 0 or (self.x > (  
 vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].x +  
 vehiclesTurned[self.direction][self.lane][  
 self.crossedIndex - 1].image.get\_rect().width + movingGap))):  
 self.x -= self.speed  
 elif (self.lane == 2):  
 if (self.crossed == 0 or self.y > mid[self.direction]['y']):  
 if ((self.y >= self.stop or (  
 currentGreen == 3 and currentYellow == 0) or self.crossed == 1) and (  
 self.index == 0 or self.y > (vehicles[self.direction][self.lane][self.index - 1].y +  
 vehicles[self.direction][self.lane][  
 self.index - 1].image.get\_rect().height + movingGap) or  
 vehicles[self.direction][self.lane][self.index - 1].turned == 1)):  
 self.y -= self.speed  
 else:  
 if (self.turned == 0):  
 self.rotateAngle += rotationAngle  
 self.image = pygame.transform.rotate(self.originalImage, -self.rotateAngle)  
 self.x += 1  
 self.y -= 1  
 if (self.rotateAngle == 90):  
 self.turned = 1  
 vehiclesTurned[self.direction][self.lane].append(self)  
 self.crossedIndex = len(vehiclesTurned[self.direction][self.lane]) - 1  
 else:  
 if (self.crossedIndex == 0 or (self.x < (  
 vehiclesTurned[self.direction][self.lane][self.crossedIndex - 1].x -  
 vehiclesTurned[self.direction][self.lane][  
 self.crossedIndex - 1].image.get\_rect().width - movingGap))):  
 self.x += self.speed  
 else:  
 if (self.crossed == 0):  
 if ((self.y >= self.stop or (currentGreen == 3 and currentYellow == 0)) and (  
 self.index == 0 or self.y > (  
 vehicles[self.direction][self.lane][self.index - 1].y + vehicles[self.direction][self.lane][  
 self.index - 1].image.get\_rect().height + movingGap))):  
 self.y -= self.speed  
 else:  
 if ((self.crossedIndex == 0) or (self.y > (  
 vehiclesNotTurned[self.direction][self.lane][self.crossedIndex - 1].y +  
 vehiclesNotTurned[self.direction][self.lane][  
 self.crossedIndex - 1].image.get\_rect().height + movingGap))):  
 self.y -= self.speed  
  
  
# Initialization of signals with default values  
def initialize():  
 minTime = randomGreenSignalTimerRange[0]  
 maxTime = randomGreenSignalTimerRange[1]  
 if (randomGreenSignalTimer):  
 ts1 = TrafficSignal(0, defaultYellow, random.randint(minTime, maxTime))  
 signals.append(ts1)  
 ts2 = TrafficSignal(ts1.yellow + ts1.green, defaultYellow, random.randint(minTime, maxTime))  
 signals.append(ts2)  
 ts3 = TrafficSignal(defaultRed, defaultYellow, random.randint(minTime, maxTime))  
 signals.append(ts3)  
 ts4 = TrafficSignal(defaultRed, defaultYellow, random.randint(minTime, maxTime))  
 signals.append(ts4)  
 repeat()  
  
def repeat():  
 global currentGreen, currentYellow, nextGreen  
 while (signals[currentGreen].green > 0): # while the timer of current green signal is not zero  
 updateValues()  
 time.sleep(1)  
 currentYellow = 1 # set yellow signal on  
 # reset stop coordinates of lanes and vehicles  
 for i in range(0, 3):  
 for vehicle in vehicles[directionNumbers[currentGreen]][i]:  
 vehicle.stop = defaultStop[directionNumbers[currentGreen]]  
 while (signals[currentGreen].yellow > 0): # while the timer of current yellow signal is not zero  
 updateValues()  
 time.sleep(1)  
 currentYellow = 0 # set yellow signal off  
  
 # reset all signal times of current signal to default/random times  
 if (randomGreenSignalTimer):  
 signals[currentGreen].green = random.randint(randomGreenSignalTimerRange[0], randomGreenSignalTimerRange[1])  
 else:  
 signals[currentGreen].green = defaultGreen[currentGreen]  
 signals[currentGreen].yellow = defaultYellow  
 signals[currentGreen].red = defaultRed  
  
 currentGreen = nextGreen # set next signal as green signal  
 nextGreen = (currentGreen + 1) % noOfSignals # set next green signal  
 signals[nextGreen].red = signals[currentGreen].yellow + signals[  
 currentGreen].green # set the red time of next to next signal as (yellow time + green time) of next signal  
 repeat()  
  
  
# Update values of the signal timers after every second  
def updateValues():  
 for i in range(0, noOfSignals):  
 if (i == currentGreen):  
 if (currentYellow == 0):  
 signals[i].green -= 1  
 else:  
 signals[i].yellow -= 1  
 else:  
 signals[i].red -= 1  
  
  
# Generating vehicles in the simulation  
def generateVehicles():  
 while (True):  
 vehicle\_type = random.choice(allowedVehicleTypesList)  
 lane\_number = random.randint(1, 2)  
 will\_turn = 0  
 if (lane\_number == 1):  
 temp = random.randint(0, 99)  
 if (temp < 40):  
 will\_turn = 1  
 elif (lane\_number == 2):  
 temp = random.randint(0, 99)  
 if (temp < 40):  
 will\_turn = 1  
 temp = random.randint(0, 99)  
  
 dist = [25, 50, 75, 100]  
 if (temp < dist[0]):  
 direction\_number = 0  
 elif (temp < dist[1]):  
 direction\_number = 1  
 elif (temp < dist[2]):  
 direction\_number = 2  
 elif (temp < dist[3]):  
 direction\_number = 3  
 Vehicle(lane\_number, vehicleTypes[vehicle\_type], direction\_number, directionNumbers[direction\_number],  
 will\_turn)  
 time.sleep(1)  
  
  
def showStats():  
 totalVehicles = 0  
 for i in range(0, 4):  
 if (signals[i] != None):  
 totalVehicles += vehicles[directionNumbers[i]]['crossed']  
 print('Total time:', timeElapsed)  
 print('Total vehicles passed:', totalVehicles)  
  
  
def simTime():  
 global timeElapsed, simulationTime  
 while (True):  
 timeElapsed += 1  
 time.sleep(1)  
 if (timeElapsed == simulationTime):  
 showStats()  
  
class Main:  
 global allowedVehicleTypesList  
 i = 0  
 for vehicleType in allowedVehicleTypes:  
 if (allowedVehicleTypes[vehicleType]):  
 allowedVehicleTypesList.append(i)  
 i += 1  
 thread1 = threading.Thread(name="initialization", target=initialize, args=()) # initialization  
 thread1.daemon = True  
 thread1.start()  
  
 # Colours  
 black = (0, 0, 0)  
 white = (255, 255, 255)  
  
 # Screensize  
 screenWidth = 1400  
 screenHeight = 800  
 screenSize = (screenWidth, screenHeight)  
  
 # Setting background image i.e. image of intersection  
 background = pygame.image.load('images/intersection.png')  
  
 screen = pygame.display.set\_mode(screenSize)  
 pygame.display.set\_caption("SIMULATION")  
  
 # Loading signal images and font  
 redSignal = pygame.image.load('images/signals/red.png')  
 yellowSignal = pygame.image.load('images/signals/yellow.png')  
 greenSignal = pygame.image.load('images/signals/green.png')  
 font = pygame.font.Font(None, 30)  
 thread2 = threading.Thread(name="generateVehicles", target=generateVehicles, args=()) # Generating vehicles  
 thread2.daemon = True  
 thread2.start()  
  
 thread3 = threading.Thread(name="simTime", target=simTime, args=())  
 thread3.daemon = True  
 thread3.start()  
  
 while True:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 showStats()  
 sys.exit()  
  
 screen.blit(background, (0, 0)) # display background in simulation  
 for i in range(0,  
 noOfSignals): # display signal and set timer according to current status: green, yello, or red  
 if (i == currentGreen):  
 if (currentYellow == 1):  
 signals[i].signalText = signals[i].yellow  
 screen.blit(yellowSignal, signalCoods[i])  
 else:  
 signals[i].signalText = signals[i].green  
 screen.blit(greenSignal, signalCoods[i])  
 else:  
 if (signals[i].red <= 10):  
 signals[i].signalText = signals[i].red  
 else:  
 signals[i].signalText = "---"  
 screen.blit(redSignal, signalCoods[i])  
 signalTexts = ["", "", "", ""]  
  
 # display signal timer  
 for i in range(0, noOfSignals):  
 signalTexts[i] = font.render(str(signals[i].signalText), True, white, black)  
 screen.blit(signalTexts[i], signalTimerCoods[i])  
  
  
 # display vehicle count  
 for i in range(0, noOfSignals):  
 displayText = vehicles[directionNumbers[i]]['crossed']  
 vehicleCountTexts[i] = font.render(str(displayText), True, black, white)  
 screen.blit(vehicleCountTexts[i], vehicleCountCoods[i])  
  
 # display time elapsed  
 timeElapsedText = font.render(("Time Elapsed: " + str(timeElapsed)), True, black, white)  
 screen.blit(timeElapsedText, timeElapsedCoods)  
  
 # display the vehicles  
 for vehicle in simulation:  
 screen.blit(vehicle.image, [vehicle.x, vehicle.y])  
 vehicle.move()  
 pygame.display.update()  
Main()

**OUTPUT**

Graphical user interface

Description automatically generated with medium confidence