DEPARTMENT OF COMPUTER ENGINEERING

A. Y. 2025-26 Semester-I

MINI PROJECT REPORT

Subject: Data Structures

Group No. :

Title of the Project: Traffic light controller

Group Members:

Sr. No.	PRN No.	Name of the Student
1	B24CE1034	Abhijeet B Hake
2	B24CE1038	Pruthviraj Kokate
3	B24CE1040	Soham Badarkhe
4	B24CE1041	Shivtej Wagare

Data Structures Used: Array, Circular linked list, Circular Queue

Mini-Project Idea: Intelligent Traffic Signal System Using YOLO and Python

Input:

- Real-time or pre-captured traffic images/videos.
- Each frame is processed to detect vehicles like cars, buses, trucks, bikes, and rickshaws.

Output:

- Dynamically controlled traffic lights that adjust green time based on the number of vehicles detected.
- A graphical simulation showing vehicle movement and signal transitions using Pygame.

Techniques Used:

- Object Detection: YOLOv2 model via Darkflow to detect vehicles.
- Image Processing: OpenCV for reading and annotating detected vehicles.
- Simulation: Pygame for visualizing intersection and signal behavior.
- Multithreading: Used to handle simultaneous processes — detection, signal timing, and simulation updates.
- Dynamic Time Calculation: Signal time changes according to traffic density using mathematical estimation.

Program:

```
# LAG
# NO. OF VEHICLES IN SIGNAL CLASS
# stops not used
# DISTRIBUTION
# BUS TOUCHING ON TURNS
# Distribution using python class

# *** IMAGE XY COOD IS TOP LEFT
import random
import math
import time
import threading
# from vehicle_detection import detection
import pygame
import sys
import os

# options={
#    'model':'./cfg/yolo.cfg',     #specifying the path of model
#    'load':'./bin/yolov2.weights',   #weights
#    'threshold':0.3     #minimum confidence factor to create a box, greater than 0.3 good
# }

# tfnet=TFNet(options)    #READ ABOUT TFNET

# Default values of signal times
defaultRed = 150
defaultYellow = 5
defaultGreen = 20
defaultMinimum = 10
defaultMaximum = 60

signals = []
noOfSignals = 4
simTime = 300       # change this to change time of simulation
timeElapsed = 0

currentGreen = 0   # Indicates which signal is green
nextGreen = (currentGreen+1)%noOfSignals
currentYellow = 0   # Indicates whether yellow signal is on or off

# Average times for vehicles to pass the intersection
carTime = 2
bikeTime = 1
rickshawTime = 2.25
busTime = 2.5
truckTime = 2.5

# Count of cars at a traffic signal
noOfCars = 0
```

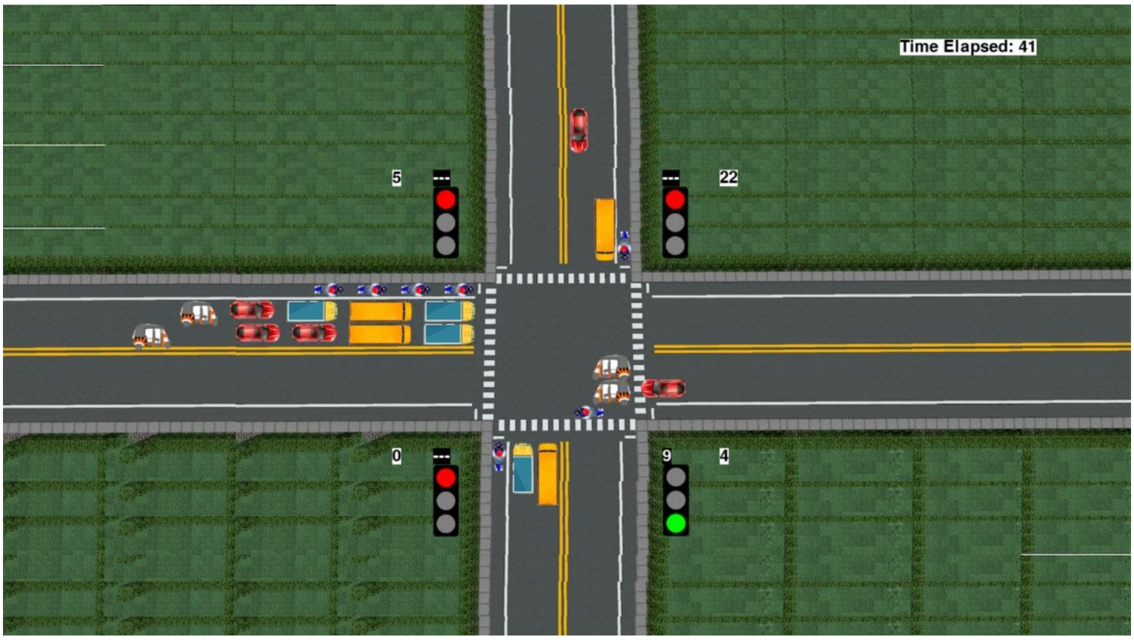
```
noOfBikes = 0
noOfBuses =0
noOfTrucks = 0
noOfRickshaws = 0
noOfLanes = 2

# Red signal time at which cars will be detected at a signal
detectionTime = 5

speeds = {'car':2.25, 'bus':1.8, 'truck':1.8, 'rickshaw':2, 'bike':2.5}  # average speeds of vehicles

# Coordinates of 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:'rickshaw', 4:'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)]
vehicleCountCoods = [(480,210),(880,210),(880,550),(480,550)]
vehicleCountTexts = ["0", "0", "0", "0"]

# Coordinates of stop lines
stopLines = {'right': 590, 'down': 330, 'left': 800, 'up': 535}
defaultStop = {'right': 580, 'down': 320, 'left': 810, 'up': 545}
stops = {'right': [580,580,580], 'down': [320,320,320], 'left': [810,810,810], 'up': [545,545,545]}

mid = {'right': {'x':705, 'y':445}, 'down': {'x':695, 'y':450}, 'left': {'x':695, 'y':425}, 'up': {'x':695, 'y':400}}
rotationAngle = 3

# Gap between vehicles
gap = 15    # stopping gap
gap2 = 15   # moving gap

pygame.init()
simulation = pygame.sprite.Group()

class TrafficSignal:
    def __init__(self, red, yellow, green, minimum, maximum):
        self.red = red
        self.yellow = yellow
        self.green = green
        self.minimum = minimum
        self.maximum = maximum
        self.signalText = "30"
        self.totalGreenTime = 0

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.stop = stops[direction][lane]
        self.index = len(vehicles[direction][lane]) - 1
        path = "images/" + direction + "/" + vehicleClass + ".png"
        self.originalImage = pygame.image.load(path)
        self.currentImage = pygame.image.load(path)



        if(direction=='right'):
            if(len(vehicles[direction][lane])>1 and vehicles[direction][lane][self.index-1].crossed==0):    # if more than 1 vehicle in the lane of vehicle before it has crossed stop line
                self.stop = vehicles[direction][lane][self.index-1].stop - vehicles[direction][lane][self.index-1].currentImage.get_rect().width - gap         # setting stop coordinate as: stop coordinate of next vehicle - width of next vehicle - gap
            else:
                self.stop = defaultStop[direction]
            # Set new starting and stopping coordinate
            temp = self.currentImage.get_rect().width + gap
            x[direction][lane] -= temp
            stops[direction][lane] -= temp
        elif(direction=='left'):
            if(len(vehicles[direction][lane])>1 and vehicles[direction][lane][self.index-1].crossed==0):
                self.stop = vehicles[direction][lane][self.index-1].stop + vehicles[direction][lane][self.index-1].currentImage.get_rect().width + gap
            else:
                self.stop = defaultStop[direction]
            temp = self.currentImage.get_rect().width + gap
            x[direction][lane] += temp
            stops[direction][lane] += temp
        elif(direction=='down'):
            if(len(vehicles[direction][lane])>1 and vehicles[direction][lane][self.index-1].crossed==0):
                self.stop = vehicles[direction][lane][self.index-1].stop - vehicles[direction][lane][self.index-
1].currentImage.get_rect().height - gap
            else:
                self.stop = defaultStop[direction]
            temp = self.currentImage.get_rect().height + gap
            y[direction][lane] -= temp
            stops[direction][lane] -= temp
        elif(direction=='up'):
            if(len(vehicles[direction][lane])>1 and vehicles[direction][lane][self.index-1].crossed==0):
                self.stop = vehicles[direction][lane][self.index-1].stop + vehicles[direction][lane][self.index-
1].currentImage.get_rect().height + gap
            else:
                self.stop = defaultStop[direction]
            temp = self.currentImage.get_rect().height + gap
            y[direction][lane] += temp
            stops[direction][lane] += temp
        simulation.add(self)

    def render(self, screen):
        screen.blit(self.currentImage, (self.x, self.y))

    def move(self):
        if(self.direction=='right'):
            if(self.crossed==0 and self.x+self.currentImage.get_rect().width>stopLines[self.direction]):   # if the
image has crossed stop line now
                self.crossed = 1
                vehicles[self.direction]['crossed'] += 1
            if(self.willTurn==1):
                if(self.crossed==0 or self.x+self.currentImage.get_rect().width<mid[self.direction]['x']):
                    if((self.x+self.currentImage.get_rect().width<=self.stop or (currentGreen==0 and currentYellow==0)
or self.crossed==1) and (self.index==0 or
self.x+self.currentImage.get_rect().width<(vehicles[self.direction][self.lane][self.index-1].x - gap2) or
vehicles[self.direction][self.lane][self.index-1].turned==1)):
                        self.x += self.speed
                else:
                    if(self.turned==0):
                        self.rotateAngle += rotationAngle
                        self.currentImage = pygame.transform.rotate(self.originalImage, -self.rotateAngle)
                        self.x += 2
                        self.y += 1.8
                        if(self.rotateAngle==90):
                            self.turned = 1
                            # path = "images/" + directionNumbers[((self.direction_number+1)%noOfSignals)] + "/" + 
self.vehicleClass + ".png"
                            # self.x = mid[self.direction]['x']
                            # self.y = mid[self.direction]['y']
                            # self.image = pygame.image.load(path)
                    else:
                        if(self.index==0 or 
self.y+self.currentImage.get_rect().height<(vehicles[self.direction][self.lane][self.index-1].y - gap2) or 
self.x+self.currentImage.get_rect().width<(vehicles[self.direction][self.lane][self.index-1].x - gap2)):
                            self.y += self.speed
            else: 
                if((self.x+self.currentImage.get_rect().width<=self.stop or self.crossed == 1 or (currentGreen==0 and 
currentYellow==0)) and (self.index==0 or 
self.x+self.currentImage.get_rect().width<(vehicles[self.direction][self.lane][self.index-1].x - gap2) or 
(vehicles[self.direction][self.lane][self.index-1].turned==1))):
                # (if the image has not reached its stop coordinate or has crossed stop line or has green signal) and (it is 
either the first vehicle in that lane or it is has enough gap to the next vehicle in that lane)
                    self.x += self.speed  # move the vehicle



        elif(self.direction=='down'):
            if(self.crossed==0 and self.y+self.currentImage.get_rect().height>stopLines[self.direction]):
                self.crossed = 1
                vehicles[self.direction]['crossed'] += 1
            if(self.willTurn==1):
                if(self.crossed==0 or self.y+self.currentImage.get_rect().height<mid[self.direction]['y']):
                    if((self.y+self.currentImage.get_rect().height<=self.stop or (currentGreen==1 and 
currentYellow==0) or self.crossed==1) and (self.index==0 or 
self.y+self.currentImage.get_rect().height<(vehicles[self.direction][self.lane][self.index-1].y - gap2) or 
vehicles[self.direction][self.lane][self.index-1].turned==1)):
                        self.y += self.speed
                else: 
                    if(self.turned==0):
                        self.rotateAngle += rotationAngle
                        self.currentImage = pygame.transform.rotate(self.originalImage, -self.rotateAngle)
                        self.x -= 2.5
                        self.y += 2
                        if(self.rotateAngle==90):
                            self.turned = 1
                    else:
                        if(self.index==0 or self.x>(vehicles[self.direction][self.lane][self.index-1].x + 
vehicles[self.direction][self.lane][self.index-1].currentImage.get_rect().width + gap2) or 
self.y<(vehicles[self.direction][self.lane][self.index-1].y - gap2)):
                            self.x -= self.speed
            else:
                if((self.y+self.currentImage.get_rect().height<=self.stop or self.crossed == 1 or (currentGreen==1 and 
currentYellow==0)) and (self.index==0 or 
self.y+self.currentImage.get_rect().height<(vehicles[self.direction][self.lane][self.index-1].y - gap2) or 
(vehicles[self.direction][self.lane][self.index-1].turned==1))):
                    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==1):
                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].currentImage.get_rect().width + gap2) or 
vehicles[self.direction][self.lane][self.index-1].turned==1)):
                        self.x -= self.speed
                else:
                    if(self.turned==0):
                        self.rotateAngle += rotationAngle
                        self.currentImage = pygame.transform.rotate(self.originalImage, -self.rotateAngle)
                        self.x -= 1.8
                        self.y -= 2.5
                        if(self.rotateAngle==90):
                            self.turned = 1
                            # path = "images/" + directionNumbers[((self.direction_number+1)%noOfSignals)] + "/" + 
self.vehicleClass + ".png"
                            # self.x = mid[self.direction]['x']
                            # self.y = mid[self.direction]['y']
                            # self.currentImage = pygame.image.load(path)
                    else:
                        if(self.index==0 or self.y>(vehicles[self.direction][self.lane][self.index-1].y + 
vehicles[self.direction][self.lane][self.index-1].currentImage.get_rect().height +  gap2) or 
self.x>(vehicles[self.direction][self.lane][self.index-1].x + gap2)):
                            self.y -= self.speed
            else:
                if((self.x>=self.stop or self.crossed == 1 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].currentImage.get_rect().width + gap2) or 
(vehicles[self.direction][self.lane][self.index-1].turned==1))):
                # (if the image has not reached its stop coordinate or has crossed stop line or has green signal) and (it is 
either the first vehicle in that lane or it is has enough gap to the next vehicle in that lane)
```

```
                    self.x -= self.speed  # move the vehicle
            # if((self.x>=self.stop or self.crossed == 1 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].currentImage.get_rect().width + gap2))):
            #     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==1):
                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].currentImage.get_rect().height +  gap2) or
vehicles[self.direction][self.lane][self.index-1].turned==1)):
                        self.y -= self.speed
                else:
                    if(self.turned==0):
                        self.rotateAngle += rotationAngle
                        self.currentImage = pygame.transform.rotate(self.originalImage, -self.rotateAngle)
                        self.x += 1
                        self.y -= 1
                        if(self.rotateAngle==90):
                            self.turned = 1
                    else:
                        if(self.index==0 or self.x<(vehicles[self.direction][self.lane][self.index-1].x -
vehicles[self.direction][self.lane][self.index-1].currentImage.get_rect().width - gap2) or
self.y>(vehicles[self.direction][self.lane][self.index-1].y + gap2)):
                            self.x += self.speed
            else:
                if((self.y>=self.stop or self.crossed == 1 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].currentImage.get_rect().height + gap2) or
(vehicles[self.direction][self.lane][self.index-1].turned==1))):
                    self.y -= self.speed

# Initialization of signals with default values
def initialize():
    ts1 = TrafficSignal(0, defaultYellow, defaultGreen, defaultMinimum, defaultMaximum)
    signals.append(ts1)
    ts2 = TrafficSignal(ts1.red+ts1.yellow+ts1.green, defaultYellow, defaultGreen, defaultMinimum,
defaultMaximum)
    signals.append(ts2)
    ts3 = TrafficSignal(defaultRed, defaultYellow, defaultGreen, defaultMinimum, defaultMaximum)
    signals.append(ts3)
    ts4 = TrafficSignal(defaultRed, defaultYellow, defaultGreen, defaultMinimum, defaultMaximum)
    signals.append(ts4)
    repeat()

# Set time according to formula
def setTime():
    global noOfCars, noOfBikes, noOfBuses, noOfTrucks, noOfRickshaws, noOfLanes
    global carTime, busTime, truckTime, rickshawTime, bikeTime
    os.system("say detecting vehicles, "+directionNumbers[(currentGreen+1)%noOfSignals])
#     detection_result=detection(currentGreen,tfnet)
#     greenTime = math.ceil(((noOfCars*carTime) + (noOfRickshaws*rickshawTime) + (noOfBuses*busTime) +
(noOfBikes*bikeTime))/(noOfLanes+1))
#     if(greenTime<defaultMinimum):
#         greenTime = defaultMinimum
#     elif(greenTime>defaultMaximum):
#         greenTime = defaultMaximum
    # greenTime = len(vehicles[currentGreen][0])+len(vehicles[currentGreen][1])+len(vehicles[currentGreen][2])
    # noOfVehicles =
len(vehicles[directionNumbers[nextGreen]][1])+len(vehicles[directionNumbers[nextGreen]][2])-
vehicles[directionNumbers[nextGreen]]['crossed']
    # print("no. of vehicles = ",noOfVehicles)
    noOfCars, noOfBuses, noOfTrucks, noOfRickshaws, noOfBikes = 0,0,0,0,0
    for j in range(len(vehicles[directionNumbers[nextGreen]][0])):
        vehicle = vehicles[directionNumbers[nextGreen]][0][j]
        if(vehicle.crossed==0):
            vclass = vehicle.vehicleClass
            # print(vclass)
            noOfBikes += 1
    for i in range(1,3):
        for j in range(len(vehicles[directionNumbers[nextGreen]][i])):
            vehicle = vehicles[directionNumbers[nextGreen]][i][j]
            if(vehicle.crossed==0):
                vclass = vehicle.vehicleClass
                # print(vclass)
                if(vclass=='car'):
                    noOfCars += 1
                elif(vclass=='bus'):
                    noOfBuses += 1
                elif(vclass=='truck'):
                    noOfTrucks += 1
                elif(vclass=='rickshaw'):
                    noOfRickshaws += 1
    # print(noOfCars)
    greenTime = math.ceil(((noOfCars*carTime) + (noOfRickshaws*rickshawTime) + (noOfBuses*busTime) +
(noOfTrucks*truckTime)+ (noOfBikes*bikeTime))/(noOfLanes+1))
```

```
    # greenTime = math.ceil((noOfVehicles)/noOfLanes)
    print('Green Time: ',greenTime)
    if(greenTime<defaultMinimum):
        greenTime = defaultMinimum
    elif(greenTime>defaultMaximum):
        greenTime = defaultMaximum
    # greenTime = random.randint(15,50)
    signals[(currentGreen+1)%(noOfSignals)].green = greenTime

def repeat():
    global currentGreen, currentYellow, nextGreen
    while(signals[currentGreen].green>0):   # while the timer of current green signal is not zero
        printStatus()
        updateValues()
        if(signals[(currentGreen+1)%(noOfSignals)].red==detectionTime):    # set time of next green signal
            thread = threading.Thread(name="detection",target=setTime, args=())
            thread.daemon = True
            thread.start()
            # setTime()
        time.sleep(1)
    currentYellow = 1   # set yellow signal on
    vehicleCountTexts[currentGreen] = "0"
    # reset stop coordinates of lanes and vehicles
    for i in range(0,3):
        stops[directionNumbers[currentGreen]][i] = defaultStop[directionNumbers[currentGreen]]
        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
        printStatus()
        updateValues()
        time.sleep(1)
    currentYellow = 0   # set yellow signal off

    # reset all signal times of current signal to default times
    signals[currentGreen].green = defaultGreen
    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()

# Print the signal timers on cmd
```

```
def printStatus():
    for i in range(0, noOfSignals):
        if(i==currentGreen):
            if(currentYellow==0):
                print(" GREEN TS",i+1,"-> r:",signals[i].red," y:",signals[i].yellow," g:",signals[i].green)
            else:
                print("YELLOW TS",i+1,"-> r:",signals[i].red," y:",signals[i].yellow," g:",signals[i].green)
        else:
            print("   RED TS",i+1,"-> r:",signals[i].red," y:",signals[i].yellow," g:",signals[i].green)
    print()

# 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
                signals[i].totalGreenTime+=1
            else:
                signals[i].yellow-=1
        else:
            signals[i].red-=1

# Generating vehicles in the simulation
def generateVehicles():
    while(True):
        vehicle_type = random.randint(0,4)
        if(vehicle_type==4):
            lane_number = 0
        else:
            lane_number = random.randint(0,1) + 1
        will_turn = 0
        if(lane_number==2):
            temp = random.randint(0,4)
            if(temp<=2):
                will_turn = 1
            elif(temp>2):
                will_turn = 0
        temp = random.randint(0,999)
        direction_number = 0
        a = [400,800,900,1000]
        if(temp<a[0]):
            direction_number = 0
        elif(temp<a[1]):
            direction_number = 1
        elif(temp<a[2]):
            direction_number = 2
        elif(temp<a[3]):
            direction_number = 3
        Vehicle(lane_number, vehicleTypes[vehicle_type], direction_number,
directionNumbers[direction_number], will_turn)
        time.sleep(0.75)

def simulationTime():
    global timeElapsed, simTime
    while(True):
        timeElapsed += 1
        time.sleep(1)
        if(timeElapsed==simTime):
            totalVehicles = 0
            print('Lane-wise Vehicle Counts')
            for i in range(noOfSignals):
                print('Lane',i+1,':',vehicles[directionNumbers[i]]['crossed'])
                totalVehicles += vehicles[directionNumbers[i]]['crossed']
            print('Total vehicles passed: ',totalVehicles)
            print('Total time passed: ',timeElapsed)
            print('No. of vehicles passed per unit time: ',(float(totalVehicles)/float(timeElapsed)))
            os._exit(1)


class Main:
    thread4 = threading.Thread(name="simulationTime",target=simulationTime, args=())
    thread4.daemon = True
    thread4.start()

    thread2 = threading.Thread(name="initialization",target=initialize, args=())    # initialization
    thread2.daemon = True
    thread2.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/mod_int.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)

    thread3 = threading.Thread(name="generateVehicles",target=generateVehicles, args=())    # Generating
vehicles
    thread3.daemon = True
    thread3.start()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                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):
                    if(signals[i].yellow==0):
                        signals[i].signalText = "STOP"
                    else:
                        signals[i].signalText = signals[i].yellow
                    screen.blit(yellowSignal, signalCoods[i])
                else:
                    if(signals[i].green==0):
                        signals[i].signalText = "SLOW"
                    else:
                        signals[i].signalText = signals[i].green
                    screen.blit(greenSignal, signalCoods[i])
            else:
                if(signals[i].red<=10):
                    if(signals[i].red==0):
                        signals[i].signalText = "GO"
                    else:
                        signals[i].signalText = signals[i].red
                else:
                    signals[i].signalText = "---"
                screen.blit(redSignal, signalCoods[i])
        signalTexts = ["","","",""]
        # display signal timer and vehicle count
        for i in range(0,noOfSignals):
            signalTexts[i] = font.render(str(signals[i].signalText), True, white, black)
            screen.blit(signalTexts[i],signalTimerCoods[i])
            displayText = vehicles[directionNumbers[i]]['crossed']
            vehicleCountTexts[i] = font.render(str(displayText), True, black, white)
            screen.blit(vehicleCountTexts[i],vehicleCountCoods[i])

        timeElapsedText = font.render(("Time Elapsed: "+str(timeElapsed)), True, black, white)
        screen.blit(timeElapsedText,(1100,50))

        # display the vehicles
        for vehicle in simulation:
            screen.blit(vehicle.currentImage, [vehicle.x, vehicle.y])
            # vehicle.render(screen)
            vehicle.move()
        pygame.display.update()

Main()
```

Output:



Analysis:
Time Complexity:
Best case: O(n)
Average case: O(n x m)
Worst case: O(n x m x d)

Space Complexity:
Best case: O(1)
Average case: O(n)
Worst case: O(n + d)

Where,
n = number of frames processed (simulation steps).
m = number of vehicles per frame.
d = number of detections (YOLO bounding boxes).