MF_Simulation_Camera

December 21, 2021

@author: Marcos Tulio Fermin Lopez

[ ]:
```
import random
import math
import time
import threading
import pygame
import sys
import os
import Data_Manager
import Dynamic_Traffic_Maker
```

This module simulates the Dynamic Traffic Lights with a high speed camera in each direction.

The fundamental assumption for the high speed camera techonology is to provide a highly efficient image processing method to dynamically control traffic lights by detecting the presence of vehicles at the intersection. In a real-time IoT project the cameras would be connected to a image processing module that detects the vehicles using computer vision. However, for this simulation, only the expected behavior of the traffic lights after image processing was considered.

This module assumes that the cameras are sending their live feed to a image processing unit, and the traffic lights' timing self-adjust based on the processing done by this unit.

The green light timing of each traffic light changes dynamically by providing a longer greentime to the direction that has the most vehicles. The signals are activated in a clock-wise direction.

At the end of the simulation the algorithm will print the number of cars served per techonology, the average waiting time in red at the intersection, and their efficiencies compared to each other in the form of graphs and tables; if the plotter file is run.

[ ]:
```
# Initial Parameters

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

signals = []
```

```
noOfSignals = 4
simTime = 3600  # 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
busTime = 2.5
truckTime = 2.5

# Count of cars at a traffic signal
noOfCars = 0
noOfBikes = 0
noOfBuses = 0
noOfTrucks = 0
noOfLanes = 2

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

# Average speeds of vehicles in terms of pixels per second
speeds = {
    "car": 2.25,
    "bus": 1.8,
    "truck": 1.8,
    "bike": 2.5,
}

# Coordinates of vehicles' start
x = {
    "right": [0, 0, 0],
    "down": [775, 747, 717],
    "left": [1400, 1400, 1400],
    "up": [602, 627, 657],
}
y = {
    "right": [338, 360, 388],
    "down": [0, 0, 0],
    "left": [508, 476, 446],
    "up": [800, 800, 800],
}

# Dictionary of vehicles in the simulation with lanes per direction
```

```
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 = [(493, 230), (875, 230), (875, 570), (493, 570)]
signalTimerCoods = [(530, 210), (850, 210), (850, 550), (530, 550)]
vehicleCountTexts = ["0", "0", "0", "0"]
vehicleCountCoods = [(480, 210), (910, 210), (910, 550), (480, 550)]

# Camera Coordinates
leftcameraCoords = (400, 250)
rightcameraCoords = (900, 530)
upcameraCoords = (530, 550)
downcameraCoords = (760, 210)

# Coordinates of stop lines
stopLines = {"right": 391, "down": 200, "left": 1011, "up": 665}
defaultStop = {"right": 381, "down": 190, "left": 1021, "up": 675}
stops = {
    "right": [381, 381, 381],
    "down": [190, 190, 190],
```

```
    "left": [1021, 1021, 1021],
    "up": [675, 675, 675],
}

# Coordinates of the middle line of the intersection relative to the x axis
mid = {
    "right": {
        "x": 700,
        "y": 461
    },
    "down": {
        "x": 700,
        "y": 461
    },
    "left": {
        "x": 700,
        "y": 461
    },
    "up": {
        "x": 700,
        "y": 461
    },
}
# Default rotation angle of the cars
rotationAngle = 3

# Gap between vehicles
gap = 15  # Stopping gap from vehicle to the stop line pixels per second
gap2 = 15  # Moving gap between vehicles in pixels per second

pygame.init()  # Initializes Pygame
# A container class to hold and manage multiple Sprite objects (Vehicle images).
simulation = pygame.sprite.Group()
```

[ ]:
```
""" Calculation of the Average Waiting Time for all lanes - STARTS """
# Time managers START
leftWaitTime = 0
rightWaitTime = 0
topWaitTime = 0
bottomWaitTime = 0

# Calculates the average waiting time for the simulation
def calculateAverageWaitTime():
    global leftWaitTime, rightWaitTime, topWaitTime, bottomWaitTime
    return round(
        (((leftWaitTime + rightWaitTime + topWaitTime + bottomWaitTime) / 60) /
         4), 3)  # round to 3 dp
```

```
# Tracks the waiting time for all lanes
def trackWaitTimeForAllLanes():
    global leftWaitTime, rightWaitTime, topWaitTime, bottomWaitTime, signals,␣
 ↪currentGreen

    if signals[currentGreen] != 0:
        leftWaitTime += 1

    if signals[currentGreen] != 0:
        rightWaitTime += 1

    if signals[currentGreen] != 0:
        topWaitTime += 1

    if signals[currentGreen] != 0:
        bottomWaitTime += 1

# Time managers END
""" Calculation of the Average Waiting Time for all lanes - ENDS """
```

[ ]:
```
class TrafficSignal:
    def __init__(self, red, yellow, green, minimum, maximum):
        """ Initializes the traffic lights as objects. """
        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):
        """Initializes vehicles parameters and vehicles images as sprite␣
 ↪objects."""
        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
        # Path to load vehicle images from folder based on
        # direction and vehicle class
        path = "images/" + direction + "/" + vehicleClass + ".png"
        self.originalImage = pygame.image.load(path)
        self.currentImage = pygame.image.load(path)

        if direction == "right":
            # Checks if there is more than 1 vehicle in the lanes
            # before crossing the stop lines in the right direction
            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":
            # Checks if there is more than 1 vehicle in the lanes
            # before crossing the stop lines in the left direction
            if (len(vehicles[direction][lane]) > 1 and
                    vehicles[direction][lane][self.index - 1].crossed == 0):
                # Setting stop coordinate as: stop coordinate
                # of next vehicle - width of next vehicle - gap
                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":
            # Checks if there is more than 1 vehicle in the lanes
            # before crossing the stop lines in the down direction
            if (len(vehicles[direction][lane]) > 1 and
                    vehicles[direction][lane][self.index - 1].crossed == 0):
                # Setting stop coordinate as: stop coordinate
                # of next vehicle - width of next vehicle - gap
                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]
            # Set new starting and stopping coordinate
            temp = self.currentImage.get_rect().height + gap
            y[direction][lane] -= temp
            stops[direction][lane] -= temp
        elif direction == "up":
            # Checks if there is more than 1 vehicle in
            # the lanes before crossing the stop lines in the up direction
            if (len(vehicles[direction][lane]) > 1 and
                    vehicles[direction][lane][self.index - 1].crossed == 0):
                # Setting stop coordinate as: stop coordinate
                # of next vehicle - width of next vehicle - gap
                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]
            # Set new starting and stopping coordinate
            temp = self.currentImage.get_rect().height + gap
            y[direction][lane] += temp
            stops[direction][lane] += temp
        # Adds all parameteres to the simulation object
        simulation.add(self)

    def render(self, screen):
        """Renders the vehicle images on the screen."""
        screen.blit(self.image, (self.x, self.y))

    def move(self):
        """Move the vehicles according to their direction
           after crossing the stop line."""
        if self.direction == "right":
            # Checks If the vehicle image has crossed the stop line
            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  # The vehicle has Crossed the stop line
                vehicles[self.direction]["crossed"] += 1
        if self.willTurn == 1:
            # Checks if the vehicle that just crossed
            # the stop line will turn right
            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:
                    # Checks if the vehicle that just
                    # crossed didn't turn right
                    # If it didn't turn right, then it keeps
                    # the vehicle moving forward and keep them straight
                    self.rotateAngle += rotationAngle
                    self.currentImage = pygame.transform.rotate(
                        self.originalImage, -self.rotateAngle)
                    self.x += 3
                    self.y += 2.8
                    # If the vehicle turns right at the
                    # last moment then its decision is registered
                    if self.rotateAngle == 90:
                        self.turned = 1

                else:
                    # Index represents the relative position
                    # of the vehicle among
                    # the vehicles moving in the same direction
                    # and the same lane
                    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":
            # Checks If the vehicle image has crossed the stop line
            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:
                # Checks if the vehicle that just crossed
                # the stop line will turn
                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:
                        # Checks if the vehicle that just
                        # crossed didn't turn
                        # If it didn't turn right, then it
                        # keeps the vehicle moving forward and keep them␣
↪straight
                        self.rotateAngle += rotationAngle
                        self.currentImage = pygame.transform.rotate(
                            self.originalImage, -self.rotateAngle)
                        self.x -= 2.5
                        self.y += 2
                        # If the vehicle turns right at the last
                        # moment then its decision is registered
                        if self.rotateAngle == 90:
                            self.turned = 1
                    else:
                        # Index represents the relative position
                        # of the vehicle among the vehicles
                        # moving in the same direction and the same lane
                        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)):
                    # (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.y += self.speed # move the vehicle

        elif self.direction == "left":
```

```
            # Checks If the vehicle image has crossed the stop line
            if self.crossed == 0 and self.x < stopLines[self.direction]:
                self.crossed = 1 # The vehicle has Crossed the stop line
                vehicles[self.direction]["crossed"] += 1
            if self.willTurn == 1:
                # Checks if the vehicle that just crossed
                # the stop line will turn
                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:
                        # Checks if the vehicle that just crossed didn't turn
                        # If it didn't turn right, then it keeps the
                        # vehicle moving forward and keep them straight
                        self.rotateAngle += rotationAngle
                        self.currentImage = pygame.transform.rotate(
                            self.originalImage, -self.rotateAngle)
                        self.x -= 1.8
                        self.y -= 2.5
                        # If the vehicle turns right at the last
                        # moment then its decision is registered
                        if self.rotateAngle == 90:
                            self.turned = 1

                    else:
                        # Index represents the relative position of the vehicle
                        # among the vehicles moving in the same
                        # direction and the same lane
                        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

        elif self.direction == "up":
            # Checks If the vehicle image has crossed the stop line
            if self.crossed == 0 and self.y < stopLines[self.direction]:
                self.crossed = 1  # The vehicle has Crossed the stop line
                vehicles[self.direction]["crossed"] += 1
            if self.willTurn == 1:
                # Checks if the vehicle that just crossed
                # the stop line will turn
                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:
                        # Checks if the vehicle that just crossed didn't turn
                        # If it didn't turn right, then it keeps
                        # the vehicle moving forward and keep them straight
                        self.rotateAngle += rotationAngle
```

```
                        self.currentImage = pygame.transform.rotate(
                            self.originalImage, -self.rotateAngle)
                        self.x += 2
                        self.y -= 2
                        # If the vehicle turns right at the last
                        # moment then its decision is registered
                        if self.rotateAngle == 90:
                            self.turned = 1
                    else:
                        # Index represents the relative position
                        # of the vehicle among the vehicles
                        # moving in the same direction and the same lane
                        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)):
                    # (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.y -= self.speed# move the vehicle
```

[ ]:
```
# Initialization of signals with default values
def initialize():
    """ Initializes the traffic signals with default values. """
    # TrafficSignal1 red: 0 yellow: defaultyellow green: defaultGreen
    ts1 = TrafficSignal(0, defaultYellow, defaultGreen, defaultMinimum,
                        defaultMaximum)
    signals.append(ts1)
    # TrafficSignal2 red: (ts1.red+ts1.yellow+ts1.green)
    # yellow: defaultYellow, green: defaultGreen
```

```
    ts2 = TrafficSignal(
        ts1.red + ts1.yellow + ts1.green,
        defaultYellow,
        defaultGreen,
        defaultMinimum,
        defaultMaximum,
    )
    signals.append(ts2)
    # TrafficSignal3 red: defaultRed yellow: defaultyellow
    # green: defaultGreen
    ts3 = TrafficSignal(defaultRed, defaultYellow, defaultGreen,
                        defaultMinimum, defaultMaximum)
    signals.append(ts3)
    # TrafficSignal4 red: defaultRed yellow: defaultyellow
    # green: defaultGreen
    ts4 = TrafficSignal(defaultRed, defaultYellow, defaultGreen,
                        defaultMinimum, defaultMaximum)
    signals.append(ts4)
    repeat()
```

[ ]:
```
def setTime():
    """ Sets time based on number of vehicles. """
    global noOfCars, noOfBikes, noOfBuses, noOfTrucks, noOfLanes
    global carTime, busTime, truckTime, bikeTime

    noOfCars, noOfBuses, noOfTrucks, noOfBikes = 0, 0, 0, 0
    # Counts vehicles in the next green direction
    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
    # Counts the number of vehicles for each direction based on vehicle class
    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
    # Calculate the green time of cars
```

```
    greenTime = math.ceil(
        ((noOfCars * carTime) + (noOfBuses * busTime) +
         (noOfTrucks * truckTime) + (noOfBikes * bikeTime)) / (noOfLanes + 1))

    # Set default green time value
    if greenTime < defaultMinimum:
        greenTime = defaultMinimum
    elif greenTime > defaultMaximum:
        greenTime = defaultMaximum

    # Increase the green time of signals by one
    signals[(currentGreen + 1) % (noOfSignals)].green = greenTime
```

[ ]:
```
def repeat():
    """ Changes the color of the traffic lights based on simulation timing. """
    global currentGreen, currentYellow, nextGreen
    # while the timer of current green signal is not zero
    while (signals[currentGreen].green >
           0):
        updateValues()
        # Start a thread to set the detection time of next green signal
        if (signals[(currentGreen + 1) % (noOfSignals)].red == detectionTime
            ):
            thread = threading.Thread(name="detection",
                                      target=setTime,
                                      args=())
            thread.daemon = True
            thread.start()

        time.sleep(1)
    currentYellow = 1  # set yellow signal on
    # Initializes vehicle count when traffic lights turn green
    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 the timer of current yellow signal is not zero
    while (signals[currentGreen].yellow >
           0):
        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
    # Set the red time of next to next signal as (yellow time
    # + green time) of next signal
    signals[nextGreen].red = (
        signals[currentGreen].yellow + signals[currentGreen].green
    )
    repeat()
```

[ ]:
```
def updateValues():
    """ Updates values of the signal timers after every second. """
    # Increase the green channel of all signals
    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
```

[ ]:
```
def generateVehicles():
    """ Generates vehicles in the simulation """
    global a
    while True:
        vehicle_type = random.randint(0, 3)
        if vehicle_type == 3:
            lane_number = 0
        else:
            lane_number = random.randint(0, 1) + 1
        will_turn = 0
        if lane_number == 2:
            temp = random.randint(0, 3)
            if temp <= 2:
                will_turn = 1
            elif temp > 2:
                will_turn = 0
         # Set up a random direction number
        temp = random.randint(0, 999)
        direction_number = 0
        # Distribution of vehicles across the four directions
        a = [400, 800, 900, 1000]
```

```
        # Set the direction of the vehicle to temp
        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():
    """ Main loop for simulation time. """
    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)),
            )
            print("Average waiting Time: ", calculateAverageWaitTime())

            Data_Manager.save_Camera(f"{totalVehicles}", simTime,
                                     calculateAverageWaitTime()
                                     )  # write data of the sim to the file

            os._exit(1)
```

```
[ ]: def 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 = 922
    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(
        "Marcos Fermin's Dynamic Traffic \
        Lights Simulator - EE Capstone Project - Fall 2021"
    )

    # 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)

    # Loading Cameras Illustrations
    leftcamera = pygame.image.load("images/camera/left.png")
    rightcamera = pygame.image.load("images/camera/right.png")
    upcamera = pygame.image.load("images/camera/up.png")
    downcamera = pygame.image.load("images/camera/down.png")

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

```
    thread3.daemon = True
    thread3.start()

    # Main loop to run the simulation
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        trackWaitTimeForAllLanes()

        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])

            screen.blit(leftcamera, leftcameraCoords)
            screen.blit(rightcamera, rightcameraCoords)
            screen.blit(upcamera, upcameraCoords)
            screen.blit(downcamera, downcameraCoords)

        timeElapsedText = font.render(("Simulation Time: " + 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()
```

[ ]:
```
if __name__ == '__main__':
    main()
```