Instituto Tecnológico y de Estudios Superiores de Monterrey

M5.- Revisión de avance 2

Diego Alejandro Michel Castro | A01641907

Omar Arias Zepeda | A00830966

José Oswaldo Sobrevilla Vázquez | A01412742

Liga al repositorio: https://github.com/DiegoMichel14/Equipo4_Multiagentes

Modelación de sistemas multiagentes con gráficas computacionales

Docentes:

Raúl Ramírez

Iván Dounce

Guadalajara *9 de Marzo 2024*

Descripción del reto

El reto consiste en proponer una solución al problema de movilidad urbana en México, mediante un enfoque que reduzca la congestión vehicular al simular el tráfico representando su comportamiento desde un sistema multiagente.

Como parte de la solución se toma en cuenta una mejora medible en algún aspecto de la simulación como tiempo de traslado de los agentes, cantidad de coches por calle o tiempo transcurrido en las intersecciones.

Opción A: Tráfico vehicular en la ciudad

Es importante para las personas que residen o trabajan en una ciudad llegar a su destino de manera eficiente, cómoda y segura, por eso, en una ciudad grande, la cantidad de vehículos transportándose y encontrándose en intersecciones (algunas de éstas con semáforos) lleva a un problema de alto tráfico y circulación, llevando a las personas a frustrarse por los largos tiempos de trayecto, impidiendo incluso llegar puntualmente a sus respectivas citas o generando accidentes vehiculares.

Las rutas de automóviles como en el caso de guadalajara muchas veces cuentan con altos tiempos de espera para cruzar una avenida muy concurrida, tomando como ejemplo las avenidas cercanas al Tec en campus guadalajara podemos basarnos en 2 aspectos sobre los cuáles podemos enfocar el proyecto.

- Reducir el tiempo de espera en semáforos por cantidad de vehículos en lugar en que las luces para avanzar cambien cada cierto tiempo.
- Establecer un tiempo de espera fijo dependiendo de la dirección a la que se le quiera dar prioridad.

La implementación del proyecto parte de definir un agente (abierto a modificaciones dependiendo de la retroalimentación). El agente coche y el posible agente semáforo, los cuáles estarán ejecutando en nuestro modelo que conectará con unity.

Actualización de diagramas

Agente de vehículo	
tipo de agente:	híbrido (reactivo, práctico)
Funcionalidad:	recibir la señal del semáforo y de otros agentes de vehículo. Calcular la ruta óptima ajustar la velocidad de desplazamiento
Tipos de mensajes:	informar posición y velocidad solicitar información del estado del semáforo

Nuestro agente principal vehículo puede describirse como híbrido (reactivo, práctico) debido a las características que debe cumplir dentro del ambiente.

Práctico: El agente vehículo debe operar de manera práctica, tomando decisiones basadas en la información disponible y que sean factibles. Esto implica calcular la ruta más óptima, ajustar la velocidad para evitar colisiones o congestiones, y cumplir con las normativas de tráfico, como detenerse en un semáforo en rojo.

Reactivo: El agente vehículo responde de manera rápida a los cambios en el entorno, como las señales de semáforos, los cruces en las intersecciones y la distancia entre vehículos. Debe adaptarse dinámicamente a estas situaciones y tomar decisiones en tiempo real.

Forma de interacción: Los agentes están representados por la clase "Vehicle". La interacción de los agentes se define en el método "action()". Aquí se describen las condiciones bajo las cuales un vehículo debe detenerse y las acciones que realiza en cada paso de la simulación.

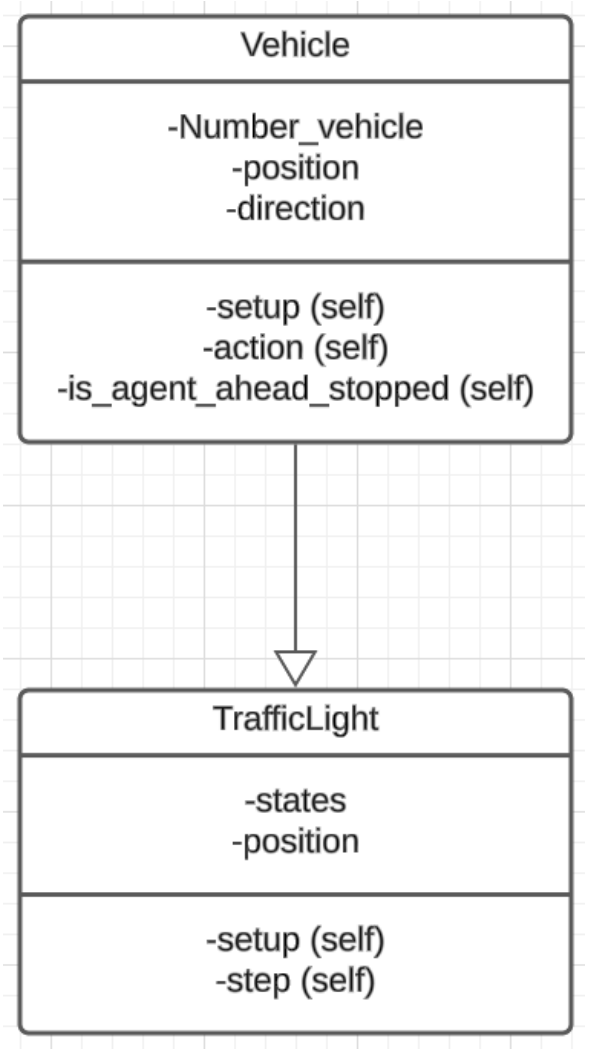
Las condiciones de detención incluyen:

- Si la luz de tráfico en la dirección del vehículo está en rojo.
- Si hay otro vehículo detenido en la posición de parada del vehículo actual.
- Si hay un vehículo detenido en la intersección frente al vehículo.

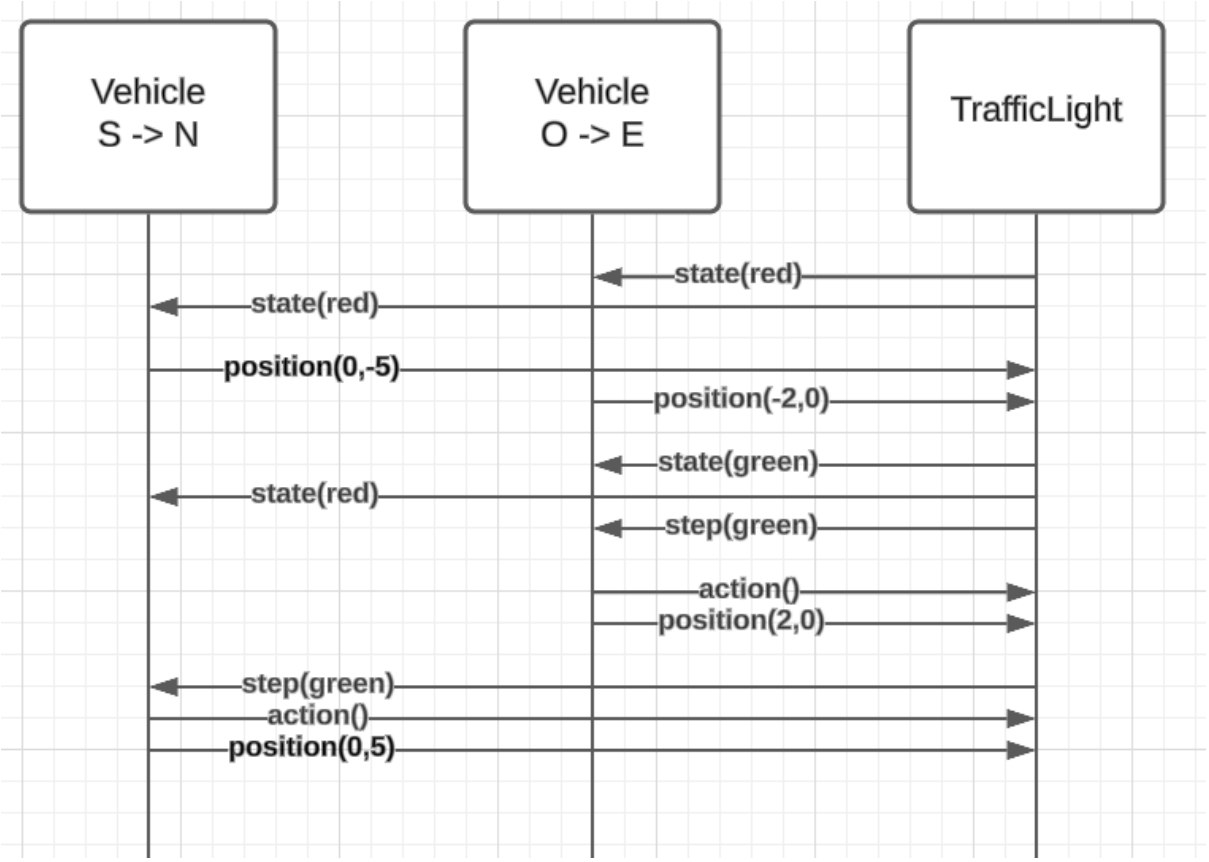
En cada paso, el vehículo realiza las siguientes acciones:

- Mueve el vehículo hacia adelante en la dirección especificada si no hay ninguna condición de detención.
- Registra la acción (movimiento hacia adelante o detención) en su historial de movimientos.
- Esta interacción se lleva a cabo para cada vehículo en cada paso de la simulación, y la clase "IntersectionModel" coordina el avance del tiempo (step()) para todos los agentes y el cambio de estado del semáforo.

- Diagrama de Clases



- Diagrama de secuencias


Vehicle
S -> N
Vehicle
O -> E
TrafficLight
state(red)
state(red)
position(0,-5)
position(-2,0)
state(green)
state(red)
step(green)
action()
position(2,0)
step(green)
action()
position(0,5)

Protocolos de interacción

En nuestro proyecto los semáforos interactuarán entre ellos haciendo uso de una *negociación* la cuál derivará en una decisión que afecte a los agentes coche.

Para poder incorporar un protocolo de negociación en la intersección debemos definir condiciones o criterios bajo los cuales se harán comparaciones.Dichos criterios fueron discutidos por los miembros del equipo y se seleccionó el primero. Las 3 opciones fueron las siguientes:

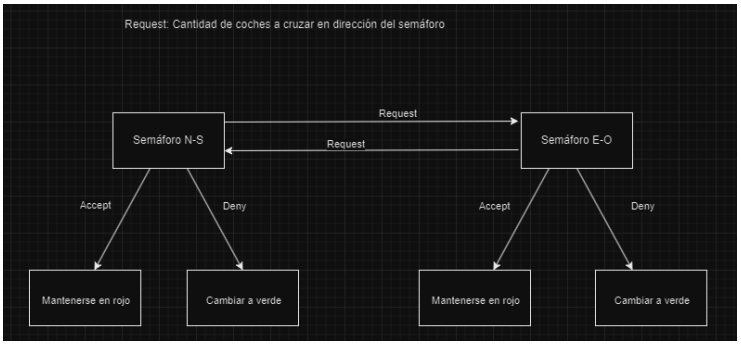
1.) Volúmen del tráfico: Uno de los semáforos puede argumentar que tiene una mayor cantidad de autos en espera para poder tener más tiempo en verde.
2.) Tiempo en rojo: Si un semáforo se excedía en cantidad de tiempo en el estado rojo podría tener prioridad para cambiar a verde.
3.) Vehículos especiales: Si había vehículos de emergencia como ambulancias o coches de policía la dirección en la que vendrían tendría prioridad.

La opción 3 quedó descartada por la inclusión de agentes extra en el modelo que no cumplen con las características que buscamos implementar.

El protocolo debe tener en cuenta 3 aspectos:
1.) Modelo los agentes: Deben definirse los semáforos como un agente en forma para su correcta comunicación.
2.) Justicia: Asignar correctamente el tiempo en verde para los semáforos que lo requieran mientras sea correspondiente a la cantidad de agentes que necesiten pasar por la intersección.
3.) Eficiencia: Establecer un balance en código adecuado que no de pie a fallos que mantengan una cantidad determinada de coches sin avanzar en caso de desahogarse la otra dirección.

La información necesaria para la negociación se puede comunicar por medio de un broadcast.

Diagrama de interacción:



Plan de trabajo

- *Actividades pendientes:*

Tras la finalización de este avance se pudo empezar a trabajar en una solución para la solución A elegida por el equipo, ya que se determinaron formalmente los agentes, actualizaron los diagramas y se establecimos la forma en la que se comunicarán un nuevo agente implementado las actividades restantes en orden de prioridad son las siguientes:

- Comenzar la programación del agente semáforo con un método de negociación más específico que permita interacción más directa entre agentes. Tiempo estimado: 2 días. Integrantes tentativos: Omar, Diego y Oswaldo. Fecha tentativa de inicio: 10 de marzo. Intervalo de esfuerzo: Medio.
- Programar un espacio gráfico en dónde podamos visualizar el movimiento de los coches por la intersección sin asignar aún la comunicación con unity. Tiempo estimado: 3 días. Integrantes tentativos: Omar. Fecha tentativa de inicio: 11 de marzo. Intervalo de esfuerzo: Alto.
- Comenzar la aplicación de texturas en objetos en unity correspondientes a la simulación en movimiento que queremos realizar incluyendo traslaciones y posible spawn para coches (agentes elegidos para el proyecto). Tiempo estimado: 2 días. Integrantes tentativos: Diego y Oswaldo. Fecha tentativa de inicio: 11 de marzo. Intervalo de esfuerzo: Medio.

- *Actividades realizadas, Avance 2:*

- Programación inicial de agentes: Omar. Tiempo realizado: 2 días (8 y 9 de Marzo), Esfuerzo “Alto”
- Redacción del documento: Omar, Oswaldo, Diego. Tiempo realizado: 2 días (8 y 9 de Marzo), Esfuerzo: Medio
- Actualización de diagramas y tipologías de agentes: Omar, Oswaldo. Tiempo realizado: 2 días (8 y 9 de Marzo). Esfuerzo: Medio.
- Actualización de contenidos del repositorio: Omar, Diego. Tiempo realizado: 1 día (9 de Marzo). Esfuerzo: Bajo.
- Redacción de protocolos de interacción y diagrama de interacción: Diego. Tiempo Realizado: 1 día (9 de Marzo). Esfuerzo: Medio.

Código del Agente en Python

```
import agentpy as ap
import random
import socket
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation


#--Agente Vehicle--
class Vehicle(ap.Agent):
   next_number = 1
   posx = 1


   def setup(self):

       self.number = Vehicle.next_number
       self.posx = Vehicle.posx
       Vehicle.next_number += 1
       Vehicle.posx -= 5
       self.firtPosition = - self.number
       self.movement_history = []
       self.direction = random.randint(1, 2)
       self.rand = random.randint(3, 6)

       # Establecer la posición inicial dependiendo de la dirección
       if self.direction == 1:
           self.position = (10, self.firtPosition)
       else:
           self.position = (self.firtPosition, 10)




   #--See function--
   def is_agent_ahead_stopped(self):
   # Calcular la posición de parada según la dirección del vehículo
       if self.direction == 1:
           stop_position = (self.position[0], self.position[1] + 2)
       else:
           stop_position = (self.position[0] + 2, self.position[1])

   # Verificar si hay un vehículo detenido en la posición de parada
       for agent in self.model.agents:
```

```
            if (abs(agent.position[0] - stop_position[0]) <= 0.1) and
(abs(agent.position[1] - stop_position[1]) <= 0.1) and agent.stop:
                return True
        return False


#-Action function--
    def action(self):
        if self.model.traffic_light.state1 == "red" and self.position[0] ==
self.model.traffic_light.position2[0] -1 and self.direction == 2:
            self.movement_history.append(0)
            self.stop = True
        elif self.model.traffic_light.state2 == "red" and self.position[1] ==
self.model.traffic_light.position2[1] -1 and self.direction == 1:
            self.movement_history.append(0)
            self.stop = True
        elif self.is_agent_ahead_stopped():
            self.movement_history.append(0)
            self.stop = True
        else:
            if self.direction == 1:
                self.position = (self.position[0], self.position[1] + 1)
                self.movement_history.append(1)
                self.stop = False
            else:
                self.position = (self.position[0] + 1, self.position[1])
                self.movement_history.append(1)
                self.stop = False


#--TrafficLight Agente--
class TrafficLight(ap.Agent):

    def setup(self):
        self.position1 = (10, 10)  # Posición del primer semáforo
        self.position2 = (10,10)  # Posición del segundo semáforo
        self.state1 = "green"  # Estado inicial del primer semáforo
        self.state2 = "red"    # Estado inicial del segundo semáforo
        self.steps_since_change = 0  # Contador de pasos desde el último cambio de
estado

    def step(self):
        # Incrementar el contador de pasos
        self.steps_since_change += 1
```

```
        # Verificar si han pasado suficientes pasos para cambiar el estado
        if self.steps_since_change >= 10:
            # Cambiar el estado de los semáforos
            if self.state1 == "red":
                self.state1 = "green"
                self.state2 = "red"
            else:
                self.state1 = "red"
                self.state2 = "green"
            self.steps_since_change = 0
```