Instituto Tecnológico y de Estudios Superiores de Monterrey

M5.- Revisión de avance 3

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 *12 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.

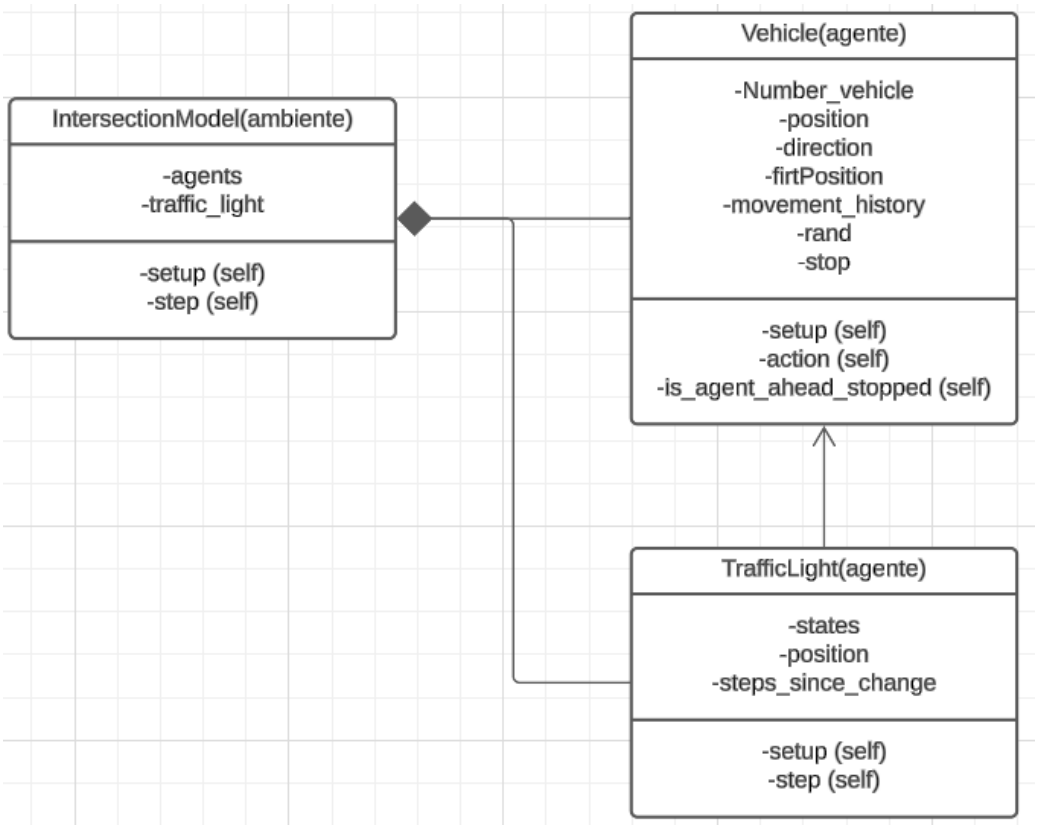
Las rutas de automóviles como en el caso de guadalajara muchas veces cuentan con altos tiempos de espera para cruzar una avenida muy concurrida, tomando como ejemplo las avenidas cercanas al Tec en campus guadalajara podemos basarnos en 2 aspectos sobre los cuáles podemos enfocar el proyecto.

- 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.

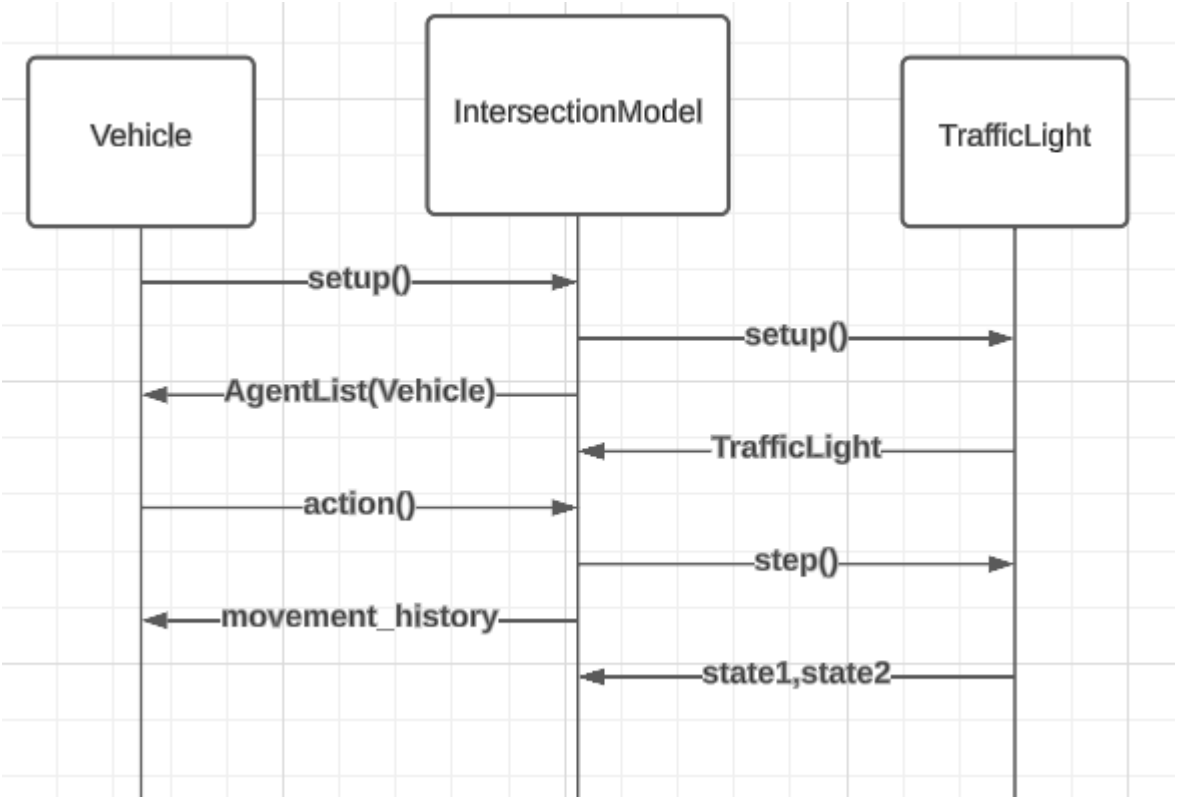
La implementación del proyecto parte de definir un agente. El agente coche y el posible agente semáforo, los cuáles estarán ejecutando en nuestro modelo que conectará con unity.

Actualización de Diagramas

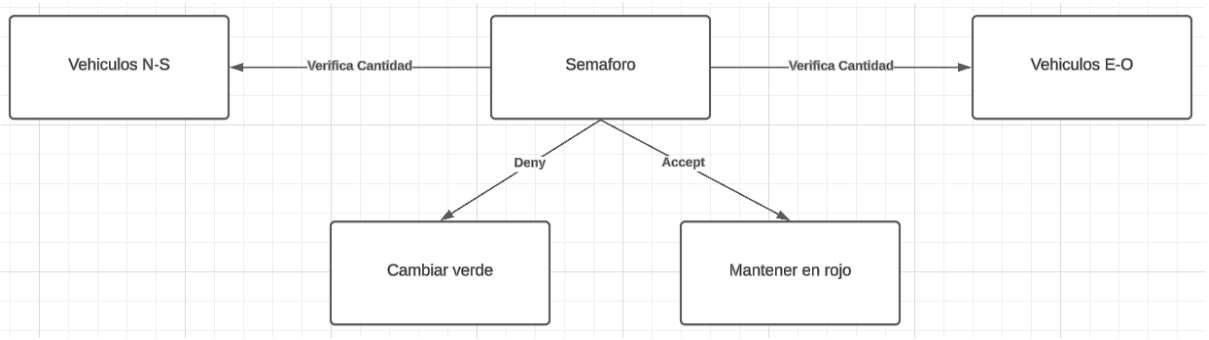
- Diagrama de clases



- Diagrama de secuencia



- Diagrama de protocolo de interacción



Códigos de unity

Dentro del motor se hacen uso de varios scripts para el movimiento de los agentes y su interacción dentro de la simulación.

- CarController

```
using UnityEngine;

Script de Unity (1 referencia de recurso) | 0 referencias
public class CarController : MonoBehaviour
{
    Rigidbody rb;

    Mensaje de Unity | 0 referencias
    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    Mensaje de Unity | 0 referencias
    void OnCollisionEnter(Collision collision)
    {
        // Verificar si la colisión es con otro carro
        if (collision.gameObject.CompareTag("Car"))
        {
            // Detener el movimiento del carro
            rb.velocity = Vector3.zero;
            // También puedes desactivar el Rigidbody para detener completamente el movimiento
            // rb.isKinematic = true;
        }
    }
}
```

- CarMovement

```
using UnityEngine;

Script de Unity (1 referencia de recurso) | 6 referencias
public class CarMovement : MonoBehaviour
{
    // Variables públicas para almacenar la información del carro
    public string direction;
    public string initialPosition;
    public string movementSequence;

    public float moveSpeed = 1f; // Velocidad de movimiento del carro en metros por segundo
    //private string movementSequence; // Secuencia de movimientos del carro
    private int currentIndex = 0; // Índice actual en la secuencia

    private float elapsedTime = 0f;
    //private float timeBetweenMoves = 1f / 60f; // Tiempo entre movimientos para cumplir con aproximadamente 60 fps

    // Método para establecer la secuencia de movimientos del carro
    1 referencia
    public void SetMovementSequence(string sequence)
    {
        movementSequence = sequence;
    }

    1 referencia
    public void SetInitialPosition(string sequence)
    {
        initialPosition = sequence;
    }

    1 referencia
    public void SetDirection(string sequence)
    {
        direction = sequence;
    }
```

```
Mensaje de Unity | 1 referencia
public void Update()
{
    if (movementSequence != null && currentIndex < movementSequence.Length)
    {
        elapsedTime += Time.deltaTime;

        // Verificar si ha pasado el tiempo suficiente para avanzar
        if (elapsedTime >= 1f) // Si ha pasado 1 segundo
        {
            char move = movementSequence[currentIndex];

            // Mover el carro según el próximo movimiento
            if (move == '1')
            {
                transform.Translate(Vector3.forward * moveSpeed);
            }

            if (move == '0')
            {
                transform.Translate(Vector3.forward * 0);
            }



            // Restablecer el tiempo acumulado y avanzar al siguiente movimiento
            elapsedTime = 0f;
            currentIndex++;
        }
    }
}
```

- Move

```
using UnityEngine;
using System.Collections.Generic;

Script de Unity (1 referencia de recurso) | 0 referencias
public class Move : MonoBehaviour
{
    public GameObject carPrefab; // Prefab del carro

    private int maxCarInstances = 50; // Máximo número de instancias de carros permitidas

    // Lista de carros instanciados
    private List<GameObject> carInstances = new List<GameObject>();

    // Update is called once per frame
    Mensaje de Unity | 0 referencias
    void Update()
    {
        string data = TCPIPServerAsync.receivedData;
        if (data != null)
        {
            string[] agentDataList = data.Split('|');

            // Iterar sobre cada cadena de datos de agente en agentDataList
            foreach (string agentData in agentDataList)
            {
                // Dividir la cadena de datos del agente en sus componentes
                string[] agentComponents = agentData.Split(',');

                // Verificar si aún se pueden instanciar más carros
                if (carInstances.Count < maxCarInstances)
                {
                    // Obtener la dirección del agente
                    string direction = agentComponents[0];

                    // Obtener la posición inicial del agente
                    string initialPosition = agentComponents[1];

                    // Obtener la secuencia de movimientos del agente
                    string movementSequence = agentComponents[2];

                    // Crear una nueva instancia de carro con los valores obtenidos
                    InstantiateCar(direction, initialPosition, movementSequence);
                }
                else
                {
                    Debug.Log("Se ha alcanzado el límite máximo de instancias de carros.");
                }
            }
        }

        // Actualizar el movimiento de los carros en cada frame
```

```
// Función para instanciar un vehículo con la dirección, posición inicial y secuencia de movimientos dados
1 referencia
private void InstantiateCar(string direction, string initialPosition, string movementSequence)
{
    // Determinar la posición inicial según la dirección
    Vector3 position;
    Quaternion rotation = Quaternion.identity;
    if (direction == "1")
    {
        position = new Vector3(10, 0, (int.Parse(initialPosition))) ;
    }
    else if (direction == "2")
    {
        position = new Vector3((int.Parse(initialPosition)), 0, 10);
        rotation = Quaternion.Euler(0, 90, 0); // Rotar 90 grados en el eje Y
    }
    else
    {
        position = Vector3.zero;
    }

    // Instanciar el carro en la posición inicial con la rotación adecuada
    GameObject car = Instantiate(carPrefab, position, rotation);

    // Asignar la secuencia de movimientos al componente adecuado del carro (si tienes uno)
    // Por ejemplo, si tienes un componente llamado CarMovement que maneja el movimiento del carro, podrías asignarle la secuencia de movimientos aquí
    if (car.GetComponent<CarMovement>() != null)
    {
        car.GetComponent<CarMovement>().SetMovementSequence(movementSequence);
        car.GetComponent<CarMovement>().SetInitialPosition(initialPosition);
        car.GetComponent<CarMovement>().SetDirection(direction);
        UpdateCarMovement();
    }

    // Agregar el carro a la lista de carros instanciados
    carInstances.Add(car);
}
```

```
// Función para actualizar el movimiento de los carros en cada frame
1 referencia
private void UpdateCarMovement()
{
    foreach (GameObject car in carInstances)
    {
        // Obtener el componente de movimiento del carro
        CarMovement carMovement = car.GetComponent<CarMovement>();
        if (carMovement != null)
        {
            // Actualizar el movimiento del carro
            carMovement.Update();
        }
    }
}
```

- TCPIPServerAsync

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Net;
using System.Net.Sockets;
using System.Threading;

Script de Unity (1 referencia de recurso) | 1 referencia
public class TCPIPServerAsync : MonoBehaviour
{

public static string receivedData;
public static string data;
// Use this for initialization

System.Threading.Thread SocketThread;
volatile bool keepReading = false;

Mensaje de Unity | 0 referencias
void Start()
{
    Application.runInBackground = true;
    startServer();
}

1 referencia
void startServer()
{
    SocketThread = new System.Threading.Thread(networkCode);
    SocketThread.IsBackground = true;
    SocketThread.Start();
}
```

```
private string getIPAddress()
{
    IPHostEntry host;
    string localIP = "";
    host = Dns.GetHostEntry(Dns.GetHostName());
    foreach (IPAddress ip in host.AddressList)
    {
        if (ip.AddressFamily == AddressFamily.InterNetwork)
        {
            localIP = ip.ToString();
        }

    }
    return localIP;
}


Socket listener;
Socket handler;
```

```
1 referencia
void networkCode()
{



    // Data buffer for incoming data.
    byte[] bytes = new Byte[1024];

    // host running the application.
    //Create EndPoint
     IPAddress IPAdr = IPAddress.Parse("127.0.0.1"); // Dirección IP
     IPEndPoint localEndPoint = new IPEndPoint(IPAdr, 1106);

    // Create a TCP/IP socket.
    listener = new Socket(AddressFamily.InterNetwork,SocketType.Stream, ProtocolType.Tcp);

    // Bind the socket to the local endpoint and
    // listen for incoming connections.

    try
    {
        listener.Bind(localEndPoint);
        listener.Listen(10);

        // Start listening for connections.
        while (true)
        {
            keepReading = true;

            // Program is suspended while waiting for an incoming connection.
            Debug.Log("Waiting for Connection");     //It works

            handler = listener.Accept();
            Debug.Log("Client Connected");     //It doesn't work
            data = null;

                byte[] SendBytes = System.Text.Encoding.Default.GetBytes("I will send key");
               handler.Send(SendBytes); // dar al cliente
```

```
            // An incoming connection needs to be processed.
            while (keepReading)
            {
                bytes = new byte[1024];
                int bytesRec = handler.Receive(bytes);

                if (bytesRec <= 0)
                {
                    keepReading = false;
                    handler.Disconnect(true);
                    break;
                }

                data += System.Text.Encoding.ASCII.GetString(bytes, 0, bytesRec);
                     //Debug.Log("Received from Server: "+data);
                     receivedData = data;

                if (data.IndexOf("<EOF>") > -1)
                {
                    break;
                }

                System.Threading.Thread.Sleep(1);
            }

            System.Threading.Thread.Sleep(1);
        }
    }
    catch (Exception e)
    {
        Debug.Log(e.ToString());
    }
}
```

```
1 referencia
void stopServer()
{
    keepReading = false;

    //stop thread
    if (SocketThread != null)
    {
        SocketThread.Abort();
    }

    if (handler != null && handler.Connected)
    {
        handler.Disconnect(false);
        Debug.Log("Disconnected!");
    }
}

Mensaje de Unity | 0 referencias
void OnDisable()
{
    stopServer();
}

Mensaje de Unity | 0 referencias
void Update(){
    receivedData = data;
}

}
```

Código de 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(1, 3)

       # 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] + self.rand)
       else:
           stop_position = (self.position[0] + self.rand, 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] -2 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] -2 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

# --environment--
class IntersectionModel(ap.Model):
    def setup(self):
        self.agents = ap.AgentList(self, 100, Vehicle)
        self.traffic_light = TrafficLight(self)

    def step(self):
        for vehicle in self.agents:
            vehicle.action()
        self.traffic_light.step()

model = IntersectionModel()
model.run(400)

# Obtener historial de movimiento de todos los vehículos


for agent in model.agents:
    vehicle_history = agent.movement_history
    movement_string = " ".join(map(str, vehicle_history))
    print("direccion: ", agent.direction, "posicion inicial:", agent.firtPosition,
"vehículo", agent.number, ":", movement_string)

agent_info = ""
for agent in model.agents:
    agent_info += f"{agent.direction},{agent.firtPosition},{''.join(map(str,
agent.movement_history))}|"



#Imprimir la información de todos los agentes sin caracteres
print(agent_info)
```

```
'''
# Crear la figura y el eje
fig, ax = plt.subplots()

# Función de inicialización de la animación
def init():
   ax.set_xlim(-10, 30)
   ax.set_ylim(-10, 30)

# Función de actualización de la animación
def update(frame):
   model.step()
   ax.clear()
   ax.set_xlim(-10, 30)
   ax.set_ylim(-10, 30)
   for agent in model.agents:
       x, y = agent.position
       ax.plot(x, y, 'ko')  # Dibujar un punto negro en la posición del agente
       ax.text(x, y, f'{agent.number}', color='black', ha='center', va='center')  #
Agregar etiqueta con el número del agente
   x1, y1 = model.traffic_light.position1
   x2, y2 = model.traffic_light.position2
   ax.plot(x1, y1, 'ro')  # Dibujar un punto rojo en la posición del primer
semáforo
   ax.plot(x2, y2, 'ro')  # Dibujar un punto rojo en la posición del segundo
semáforo

# Configurar la animación
ani = FuncAnimation(fig, update, frames=1000, init_func=init, interval=50)  # 200
frames, 1000 milisegundos (1 segundo) de intervalo entre frames

# Mostrar la animación
plt.show()

'''

# Establecer conexión con el servidor
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("127.0.0.1", 1104))

# Recibir datos del servidor
from_server = s.recv(4096)
print("Received from server:", from_server.decode("ascii"))

# Convertir el historial de movimiento a una cadena de texto
```

```
movement_str = " ".join(map(str, model.agents[0].movement_history))

# Enviar la cadena de texto codificada en bytes
s.send(agent_info.encode("ascii"))

# Cerrar la conexión
s.close()
```

Plan de trabajo y Aprendizaje Adquirido

- *Actividades pendientes:*

Tras la finalización de este avance, el equipo consiguió una conexión sólida entre nuestro sistema multiagentes y unity, se visualizan correctamente nuestras pruebas mientras que el modelado de la ciudad está completado. Las escalas son correctas y las correcciones a realizar no afectan de forma importante al funcionamiento de nuestro proyecto.
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: 12 de marzo. Intervalo de esfuerzo: Medio.
- Realizar pruebas de comunicación por medio de negociaciones para poner en práctica los protocolos de interacción. Tiempo estimado: 2 días. Integrantes tentativos: Omar, Diego y Oswaldo. Fecha tentativa de inicio: 12 de marzo. Intervalo de esfuerzo: Medio.
- Pensar e implementar la medición de tiempos o métricas del sistema para encontrar mejoras u optimizaciones que den soluciones al reto. Tiempo estimado: 2 días. Integrantes tentativos: Omar, Diego y Oswaldo. Fecha tentativa de inicio: 13 de marzo. Intervalo de esfuerzo: Medio.

- *Actividades realizadas, Avance 3:*

- Programación de un espacio gráfico en dónde visualizamos el movimiento de los coches por la intersección ya asignada una comunicación con unity.

Tiempo realizado: 3 días (9 a 12 de Marzo). Integrantes: Omar, Diego y Oswaldo. Intervalo de esfuerzo: Alto.

- Redacción del documento: Omar, Oswaldo, Diego. Tiempo realizado: 1 días (12 de Marzo), Esfuerzo: Medio
- Actualización de diagramas: Oswaldo. Tiempo realizado: 1 día (12 de Marzo). Esfuerzo: Medio.
- Actualización de contenidos del repositorio: Omar, Diego, Oswaldo. Tiempo realizado: 1 día (12 de Marzo). Esfuerzo: Bajo.
- Modelación en 3D con aplicación de texturas en unity: Diego. Tiempo realizado: 3 días (9 a 12 de Marzo). Esfuerzo: Alto.

- *Aprendizaje adquirido:*

- Diego Michel: Durante el desarrollo de este avance pude poner en práctica los ejercicios realizados en las clases de gráfica, pude aplicar texturas y modelado un poco más complejo usando proBuilder, aprendí a tener un mejor manejo del motor y el cómo funciona la instanciación de objetos al tenerlo conectado a nuestro sistema en python, también desarrollé un poco más mi lado creativo al diseñar una pequeña ciudad minimalista y colorida.
- Omar Arias: Durante este avance del proyecto desarrolle mucho mi comprensión sobre cómo desarrollar e implementar agentes, además que aprendí a desarrollar simulaciones utilizando varias herramientas como unity y python conectarlas entre sí para conseguir una simulación más detallada y profesional.
- Oswaldo Sobrevilla: En lo personal mi aprendizaje adquirido se centra en la comprensión de la problemática de movilidad urbana, la aplicación de un enfoque de simulación con agentes, la actualización de los diagramas con lo aprendido en clase, la conexión exitosa entre el sistema y Unity, y la planificación detallada de actividades futuras para mejorar el proyecto.