Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Hidalgo

M3. Actividad

Por:

Diana Guadalupe García Aguirre | A01276380

Emmanuel Bolteada Manzo | A01276310

José Herón Samperio León | A01276217

Asignatura:

Modelación de sistemas multiagentes con gráficas computacionales

Quinto semestre

Ingeniería en Tecnologías Computacionales

Profesores:

Mtro. Alfredo Israel Ramírez Mejía

Dr. Joselito Medina Marín

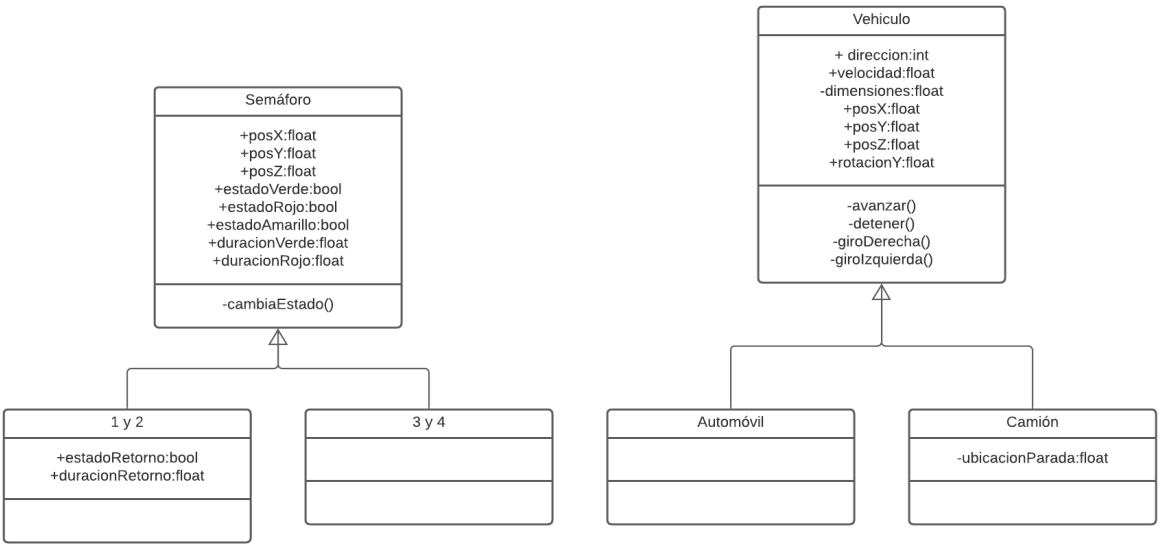
Pachuca de Soto, Hidalgo; a 30 de noviembre del 2021

Para este avance se comenzó con la implementación del servidor local utilizando Python, Flask y MESA para conectarlo con el modelo en Unity que ya estaba previamente completado. Se decidió utilizar como base el código proporcionado por el profesor la semana pasada y aplicarlo a los prefabs de automóviles con los que se contaba para hacer que se movieran en uno de los carriles. También se codificó la lógica del funcionamiento de los semáforos en el modelo de Unity y se intentó avanzar lo más posible en la detección de los mismos como obstáculos por parte de los carros.

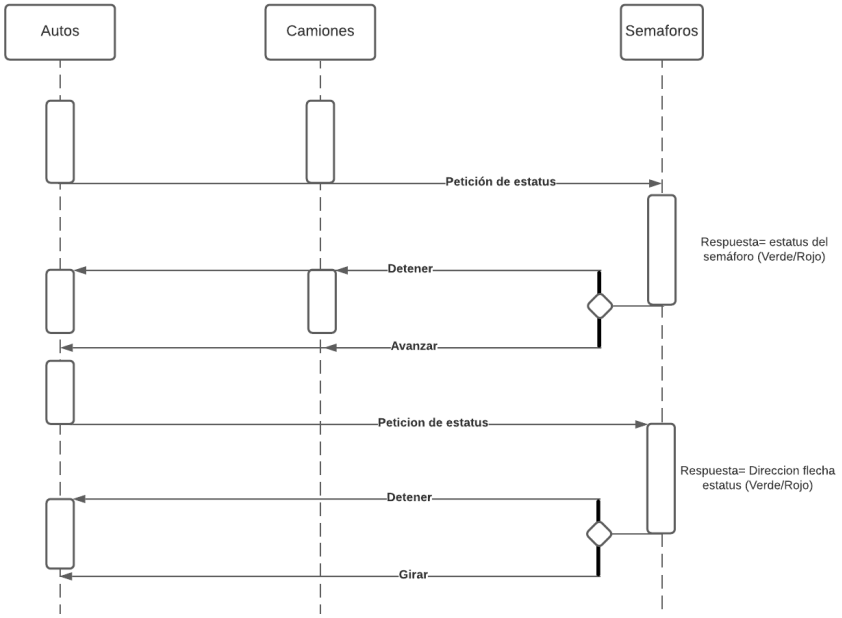
Entre las funcionalidades que se optó por dejar de lado en este avance fue el giro de los automóviles y la detección de colisiones entre ellos. Una vez que se logre tener un modelo funcional con los automóviles detectando el estado de cada semáforo y avanzando en una sola dirección en sus respectivos carriles se podrá proceder a realizar estas funcionalidades.

A continuación, se presenta la versión actualizada de los diagramas del reto tomando en consideración las modificaciones que se le han hecho por el poco tiempo disponible.

Diagrama de clases



Protocolo de interacción



Código de implementación de los agentes

RandomAgents.py

```python
from mesa import Agent, Model
from mesa.time import StagedActivation
from mesa.space import Grid
import time

# Clase que instancia los coches
class RandomAgent(Agent):

    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.direction = 1

    #Funcion que verifica si hay otro agente para moverse a ese espacio.
    def move(self):

        possible_steps = self.model.grid.get_neighborhood(
            self.pos,
            moore=True,
            include_center=True)

        freeSpaces = list(map(self.model.grid.is_cell_empty, possible_steps))

        if freeSpaces[self.direction]:
            self.model.grid.move_agent(self, possible_steps[self.direction])
            print(f"Se mueve de {self.pos} a {possible_steps[self.direction]}; direction {self.direction}")
        else:
            print(f"No se puede mover de {self.pos} en esa direccion.")

    def step(self):
        self.direction = 1
        print(f"Agente: {self.unique_id} movimiento {self.direction}")
        self.move()

# Clase que instancia los semáforos como obstaculos
class ObstacleAgent(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)

    def step(self):
        pass

# Clase que instancia el modelo de agentes
class RandomModel(Model):
    def __init__(self, N, width, height):
        self.num_agents = N
        self.grid = Grid(width,height,torus = False)
        self.schedule = StagedActivation(self)
        self.running = True
        #Posibilidad de aparicion en 2 carriles
        self.spawn1 = [631,636]
        self.semaforo1 = []
        self.semaforo2 = []
        self.semaforo3 = []
        self.semaforo4 = []

        #Coordenadas donde se encuentra el limite de parada de cada semaforo
        s1Coord=[(x,y) for y in range(471,481) for x in range(393, 394) if y in [471, 481 - 1] or x in [393, 394-1]]
        s2Coord=[(x,y) for y in range(627,637) for x in range(465, 466) if y in [627, 637 - 1] or x in [465, 466-1]]
        s3Coord=[(x,y) for y in range(642, 643) for x in range(415,425) if y in [642, 643 - 1] or x in [415, 425-1]]
        s4Coord=[(x,y) for y in range(463,464) for x in range(433, 443) if y in [463,464 - 1] or x in [433, 443-1]]

        self.spawnSemaforo(self.semaforo3, s3Coord,3)
        self.spawnSemaforo(self.semaforo4, s4Coord,4)

        # Agrega el agente a una posicion vacía
        for i in range(self.num_agents):
            a = RandomAgent(i+1000, self)
            self.schedule.add(a)

            #Cambiar posiciones de spawneo de coches
            pos_gen = lambda arr: (970, arr[self.random.randint(0,1)])
            pos = pos_gen(self.spawn1)
            while (not self.grid.is_cell_empty(pos)):
                pos = pos_gen(self.spawn1)
            self.grid.place_agent(a, pos)
            for i in range(self.random.randint(0,10)):
                self.step()
```

```python
    #Funcion que actualiza el estado de los semaforos
    def updateSemaforosStatus(self, semaforosStatus):
        print(bcolors.WARNING + "Updating Lights Status" + bcolors.ENDC)
        print(bcolors.WARNING + str(semaforosStatus) + bcolors.ENDC)

        #Coordenadas donde se encuentra el limite de parada de cada semaforo
        s1Coord=[(x,y) for y in range(471,481) for x in range(393, 394) if y in [471, 481 - 1] or x in [393, 394-1]]
        s2Coord=[(x,y) for y in range(627,637) for x in range(465, 466) if y in [627, 637 - 1] or x in [465, 466-1]]
        s3Coord=[(x,y) for y in range(642, 643) for x in range(415,425) if y in [642, 643 - 1] or x in [415, 425-1]]
        s4Coord=[(x,y) for y in range(463,464) for x in range(433, 443) if y in [463,464 - 1] or x in [433, 443-1]]

        if(semaforosStatus == 2):
            print(bcolors.OKGREEN + "Cambio a estatus 1" + bcolors.ENDC)
            self.greenlight(self.semaforo1)
            self.greenlight(self.semaforo2)
            self.spawnSemaforo(self.semaforo3, s3Coord,3)
            self.spawnSemaforo(self.semaforo4, s4Coord,4)

        elif(semaforosStatus == 1):
            print(bcolors.OKGREEN + "Cambio a estatus 2" + bcolors.ENDC)
            self.greenlight(self.semaforo3)
            self.greenlight(self.semaforo4)
            self.spawnSemaforo(self.semaforo1, s1Coord,1)
            self.spawnSemaforo(self.semaforo2, s2Coord,2)

        else:
            print(bcolors.FAIL + "No hay estatus" + bcolors.ENDC)

        self.schedule.step()

    def greenlight(self, semaforo):
        print(bcolors.OKGREEN + "Quitando semaforo" + bcolors.ENDC)
        for j in semaforo:
            self.grid.remove_agent(j)

    def spawnSemaforo(self, semaforo, coord, num):
        for pos in coord:
            obs = ObstacleAgent(pos, self)
            semaforo.append(obs)
            self.schedule.add(obs)
            self.grid.place_agent(obs, pos)
        print(bcolors.WARNING + f"Semaforo {num} creado" + bcolors.ENDC)

    def step(self):
        '''Advance the model by one step.'''
        self.schedule.step()


```

server.py

```python
from flask import Flask, request, jsonify
from RandomAgents import *

# Size of the board:
number_agents = 10
width = 28
height = 28
trafficModel = None
currentStep = 0

app = Flask("Traffic example")

# @app.route('/', methods=['POST', 'GET'])

@app.route('/init', methods=['POST', 'GET'])
def initModel():
    global currentStep, trafficModel, number_agents, width, height

    if request.method == 'POST':
        number_agents = int(request.form.get('NAgents'))
        width = int(request.form.get('width'))
        height = int(request.form.get('height'))
        currentStep = 0

        print(request.form)
        print(number_agents, width, height)
        trafficModel = RandomModel(number_agents, width, height)

        return jsonify({"message":"Parameters received, model initiated."})

@app.route('/updateLights', methods=['POST', 'GET'])
def updateLights():
    print(bcolors.WARNING + "Endpoint call" + bcolors.ENDC)
    global currentStep, trafficModel
    if request.method == 'POST':
        semaforos = int(request.form.get('semaforos'))
        print(bcolors.OKGREEN + str(semaforos) + bcolors.ENDC)
        trafficModel.updateSemaforosStatus(semaforos)
        return jsonify({"message":"Estatus de semaforo actualizado."})

@app.route('/getObstacles', methods=['GET'])
def getObstacles():
    global trafficModel

    if request.method == 'GET':
        carPositions = [{"x": x, "y":1, "z":z} for (a, x, z) in trafficModel.grid.coord_iter() if isinstance(a, ObstacleAgent)]

        return jsonify({'positions':carPositions})

@app.route('/getAgents', methods=['GET'])
def getAgents():
    global trafficModel
    if request.method == 'GET':
        carPositions = [{"x": x, "y":50, "z":z} for (a, x, z) in trafficModel.grid.coord_iter() if isinstance(a, RandomAgent)]

        return jsonify({'positions':carPositions})

@app.route('/update', methods=['GET'])
def updateModel():
    global currentStep, trafficModel
    if request.method == 'GET':
        trafficModel.step()
        currentStep += 1
        return jsonify({'message':f'Model updated to step {currentStep}.', 'currentStep':currentStep})

if __name__=='__main__':
    app.run(host="localhost", port=8585, debug=True)
```

Código de implementación de la parte gráfica

controlTrafico.cs

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

public class CarData
{
    public int uniqueID;
    public Vector3 position;
}

public class agentData
{
    public List<Vector3> positions;
}

public class ControlTrafico : MonoBehaviour
{
    // private string url = "https://boids.us-south.cf.appdomain.cloud/";
    private string getAgentsUrl = "http://localhost:8585/getAgents",
            getObstaclesUrl = "http://localhost:8585/getObstacles",
            sendConfigUrl = "http://localhost:8585/init",
            updateUrl = "http://localhost:8585/update",
            updateLightsUrl= "http://localhost:8585/updateLights";
    private agentData carsData, obstacleData;
    public GameObject carPrefab, obstaclePrefab, floor, semaforo1, semaforo2, semaforo3, semaforo4;
    public int NAgents, width, height;
    GameObject[] agents;
    public float timeToUpdate = 5.0f, timer, dt;
    public float greenLightDuration = 5.0f, redLightDuration = 5.0f;
    List<Vector3> newPositions;
    int status = 2;

    [UnityEngine.Serialization.FormerlySerializedAs("MR Main")]
    public MeshRenderer lights1, lights2, lights3,lights4;

    void Start()
    {
        carsData = new agentData();
        obstacleData = new agentData();
        newPositions = new List<Vector3>();

        agents = new GameObject[NAgents];

        floor.transform.localScale = new Vector3((float)width / 10, 1, (float)height / 10);
        timer = timeToUpdate;

        for (int i = 0; i < NAgents; i++)
        {
            agents[i] = Instantiate(carPrefab, Vector3.zero, Quaternion.identity);
            agents[i].transform.Rotate(new Vector3(0, 90, 0));
        }


        StartCoroutine(sendConfiguration());
    }
```

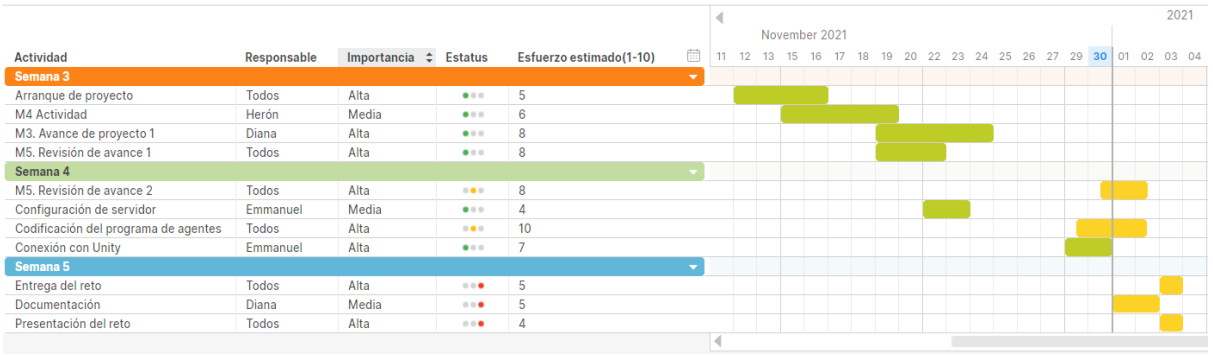
```
    private void Update()
    {
        if(status==2) greenLightDuration -= Time.deltaTime;
        else if(status==1) redLightDuration -= Time.deltaTime;

        float t = timer / timeToUpdate;
        dt = t * t * (3f - 2f * t);

        if (timer > timeToUpdate)
        {
            timer = 0;
            StartCoroutine(updateSimulation());
        }

        if (newPositions.Count > 1)
        {
            for (int s = 0; s < agents.Length; s++)
            {
                Vector3 interpolated = Vector3.Lerp(agents[s].transform.position, newPositions[s], dt);
                agents[s].transform.localPosition = interpolated;

                Vector3 dir = agents[s].transform.position - newPositions[s];

                timer += Time.deltaTime;
            }
        }

        if (greenLightDuration < 0)
        {
            semaforo1.GetComponent<SemaforoLuz>().MRChangeLeftYield(ref lights1, "Red");
            Debug.Log("Cambio a estatus 1");
            StartCoroutine(updateLightStatus(1));
            status = 1;
            greenLightDuration = 30.0f;
        }

        if (redLightDuration < 0)
        {
            semaforo1.GetComponent<SemaforoLuz>().MRChangeLeftYield(ref lights1, "Green");
            Debug.Log("Cambio a estatus 2");
            StartCoroutine(updateLightStatus(2));
            status = 2;
            redLightDuration = 30.0f;
        }
    }

    IEnumerator updateSimulation()
    {
        UnityWebRequest www = UnityWebRequest.Get(updateUrl);
        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
            Debug.Log(www.error);
        else
        {
            StartCoroutine(GetCarsData());
        }
    }

    IEnumerator sendConfiguration()
    {
        WWWForm form = new WWWForm();

        form.AddField("NAgents", NAgents.ToString());
        form.AddField("width", width.ToString());
        form.AddField("height", height.ToString());

        UnityWebRequest www = UnityWebRequest.Post(sendConfigUrl, form);
        www.SetRequestHeader("Content-Type", "application/x-www-form-urlencoded");

        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
        {
            Debug.Log(www.error);
        }
        else
        {
            Debug.Log("Configuration upload complete!");
            Debug.Log("Getting Agents positions");
            StartCoroutine(GetCarsData());
            StartCoroutine(GetObstacleData());
        }
    }
```

```
    IEnumerator updateLightStatus(int lightStatus)
    {
        WWWForm form = new WWWForm();

        form.AddField("semaforos", lightStatus.ToString());


        UnityWebRequest www = UnityWebRequest.Post(updateLightsUrl, form);
        www.SetRequestHeader("Content-Type", "application/x-www-form-urlencoded");

        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
        {
            Debug.Log("No se puede actualizar el estatus" + lightStatus +" del semaforo: " + www.error);
        }
        else
        {
            Debug.Log("Estatus de semaforos actualizado!");
        }
    }

    IEnumerator GetCarsData()
    {
        UnityWebRequest www = UnityWebRequest.Get(getAgentsUrl);
        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
            Debug.Log(www.error);
        else
        {
            carsData = JsonUtility.FromJson<agentData>(www.downloadHandler.text);

            newPositions.Clear();

            foreach (Vector3 v in carsData.positions)
                newPositions.Add(v);
        }
    }
    IEnumerator GetObstacleData()
    {
        UnityWebRequest www = UnityWebRequest.Get(getObstaclesUrl);
        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
            Debug.Log(www.error);
        else
        {
            obstacleData = JsonUtility.FromJson<agentData>(www.downloadHandler.text);

            Debug.Log(obstacleData.positions);

            foreach (Vector3 position in obstacleData.positions)
            {
                Instantiate(obstaclePrefab, position, Quaternion.identity);
            }
        }
    }
}
```

El proyecto completo y código se encuentra en el siguiente repositorio:

https://github.com/DianaA96/agentes-computacionales

Plan de trabajo actualizado



Actividad	Responsable	Importancia	Estatus	Esfuerzo estimado(1-10)
Semana 3				
Arranque de proyecto	Todos	Alta		5
M4 Actividad	Herón	Media		6
M3. Avance de proyecto 1	Diana	Alta		8
M5. Revisión de avance 1	Todos	Alta		8
Semana 4				
M5. Revisión de avance 2	Todos	Alta		8
Configuración de servidor	Emmanuel	Media		4
Codificación del programa de agentes	Todos	Alta		10
Conexión con Unity	Emmanuel	Alta		7
Semana 5				
Entrega del reto	Todos	Alta		5
Documentación	Diana	Media		5
Presentación del reto	Todos	Alta		4

Aprendizajes adquiridos

- Diana Guadalupe García Aguirre
    - Las librerías de Python para manejo de servidores y agentes, como Flask y Mesa, son de mucha utilidad en la construcción de este tipo de proyectos, ya que simplifican la enorme labor de instanciar agentes en un entorno y hacerlos interactuar.
    - En ocasiones es necesario transformar los prefabs al instanciarlos en la simulación de Unity, pues puede ser necesario rotarlos o escalarlos para que el movimiento y la escena se vean más naturales.
- José Herón Samperio León
    - La conexión por medio de python y el parentesco entre express y flask a la hora de hacer peticiones.
    - La conexión de unity y Mesa para la obtención de instrucciones de cada instancia de agentes en un entorno gráfico
- Emmanuel Bolteada Manzo
    - En ocasiones es necesario priorizar los requerimientos del proyecto para poder tener un producto funcional, siendo este el caso pues se tuvo muy poco tiempo para poder avanzar en la implementación del servidor con Unity y muchos errores surgieron.
    - Desarrollé práctica en la utilización de Flask y su integración con los métodos de MESA.
    - Sincronización de semáforos por medio de un Asset de Unity y sus GameObjects.