TC2008B: Modelación de sistemas multiagentes con gráficas computacionales

Grupo 301

Movilidad Urbana

Profesores

Dr. Gilberto Echeverría Furió

Dr. Octavio Navarro Hinojosa

Integrantes

David Medina Domínguez - A01783155
Natalia Maya Bolaños - A01781992

30 de Noviembre de 2023

Introducción

El principal objetivo de este proyecto fue observar la **movilidad urbana**, la cual se define como la habilidad de transportarse de un lugar a otro, lo cual es fundamental para el desarrollo económico, social y calidad de vida de los habitantes de una ciudad.

El uso de los automóviles para incrementar la eficiencia y espectro de la movilidad ha sido una gran ayuda, sin embargo, esta asociación ya no se ha desarrollado de la misma forma por el crecimiento y uso indiscriminado del automóvil genera efectos negativos enormes en los niveles económico, ambiental y social en México.

Durante las últimas décadas, ha existido una tendencia alarmante de un incremento en el uso de automóviles en México. Los Kilómetros-Auto Recorridos (VKT por sus siglas en Inglés) se han triplicado, de 106 millones en 1990, a 339 millones en 2010. Ésto se correlaciona simultáneamente con un incremento en los impactos negativos asociados a los autos, como el smog, accidentes, enfermedades y congestión vehicular.

Este proyecto consiste en proponer una solución al problema de movilidad urbana en México, mediante la reducción de la congestión vehicular al intentar simular de manera gráfica el tráfico, representando la salida de un sistema multi agentes, observar comportamientos e intentar aproximarse a situaciones de hoy en día.

Diseño de los agentes

CarAgent

- Objetivo
    - Moverse desde su posición inicial hacia un destino designado.
- Capacidad efectora
    - move: Mueve al agente a la siguiente posición según el camino calculado.
    - canMove: Verifica si el agente puede moverse a la siguiente posición según el camino actual y el entorno (otros agentes, semáforos).
    - calculateShortestPath: Calcula el camino más corto hacia el destino utilizando el algoritmo A*.
    - changeLane: Decide si el agente puede cambiar de carril para evitar obstáculos u otros autos, dividiendo así la carga de una misma línea en dos.

- Percepción
    - Percibe la posición actual, el agente de destino, el camino y el entorno (otros agentes, semáforos) limitado a una visión de dos cuadros adelante.
- Proactividad
    - Decide de manera proactiva cuándo moverse, cambiar de carril y recalcular el camino si es necesario para llegar más rápido a su destino o dividir la carga de tráfico al cambiarse de línea .
- Métricas de desempeño
    - Atributo estado: Representa el estado del auto, si puede moverse o no.
    - Atributo path: Representa el camino calculado para el auto.
    - Método changeLane(): Contribuye al desempeño del agente al permitirle adaptarse al entorno para llegar más rápido a su destino.

TrafficLight Agent

- Objetivo
    - Controlar el flujo de coches al dictar si pueden o no avanzar los coches.
- Capacidad efectora
    - step: Cambiar el estado del semáforo (rojo y verde) para dictar si el coche puede seguir o esperar hasta que se ponga en verde.
- Percepción
    - Percibe la posición actual, y el estado de su luz.
- Proactividad
    - No hay.
- Métricas de desempeño
    - No hay.

Destination Agent

- Objetivo
    - Observar si un coche ha llegado a su destino para encargarse de eliminarlo.
- Capacidad efectora
    - step: No hay.
- Percepción
    - Su posición.

- Proactividad
    - No hay.
- Métricas de desempeño
    - No hay.

Obstacle Agent

- Objetivo
    - Eliminar posibles caminos al coche.
- Capacidad efectora
    - Estorbar.
- Percepción
    - Su posición.
- Proactividad
    - No hay.
- Métricas de desempeño
    - No hay.

Road Agent

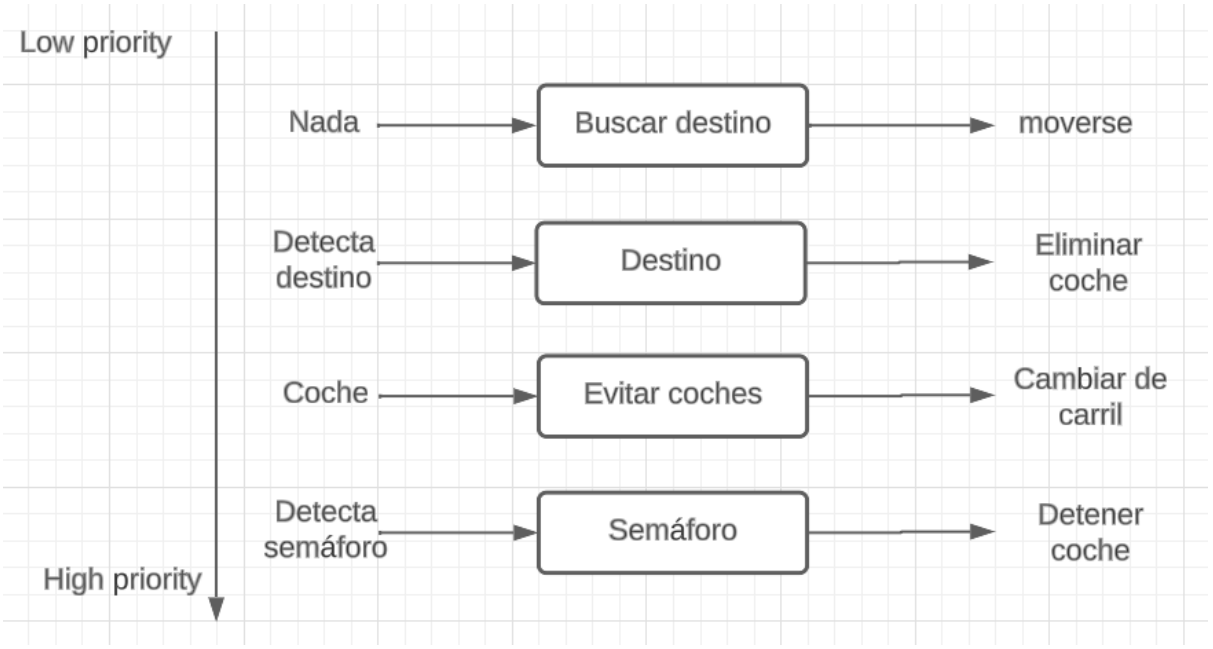
- Objetivo
    - Posibles caminos a los que puede recorrer el coche.
- Capacidad efectora
    - Dejar pasar a los agentes tipo coche.
- Percepción
    - Su posición.
- Proactividad
    - No hay.
- Métricas de desempeño
    - No hay.

Car_Generator Agent

- Objetivo
    - Posiciones en donde se van a instanciar los coches.
- Capacidad efectora

    - Crea coches cada ciertos steps.
- Percepción
    - Su posición.
- Proactividad
    - No hay.
- Métricas de desempeño
    - No hay.

Arquitectura de subsunción (agentes)



Características del ambiente

Un ambiente determinista es en el que cualquier acción tiene un único efecto garantizado: no hay incertidumbre sobre el estado que resultará de realizar una acción. En cuanto a nuestra simulación la lógica de movimiento de los agentes sigue una reglas establecidas previamente por lo que sabemos que las deben de seguir y esperamos dicho comportamiento, un ejemplo de esto es a la hora en que los coches cambian de carril por que para realizar este movimiento se basa en condiciones específicas como la longitud de camino disponible. También podemos notar que nuestra ambiente es determinista por los cambios de estado de los semáforos ya que cambian de estado cada cierto tiempo y como nosotros le

indicamos cada cuanto la transición entre estados es predecible. En general podemos decir que es un ambiente determinista ya que depende en su totalidad de las reglas y la lógica que implementamos ya que si todas las interacciones y cambios de estado pueden predecirse a partir de su estado inicial, se considera determinista.

Unity

En cuanto a la implementación visual de nuestros agentes utilizamos unity en donde con un script que genera de manera random la ciudad al igual que la altura de los edificios, básicamente usamos varios [SerializedField] lo cuál nos permite arrastrar nuestros prefabs en unity. El primero que utilizamos fue “roadprefab” donde arrastramos el prefab de piso que ya estaba en la ciudad pero le cambiamos el material, después hicimos un array del “buildingPrefab” para poder colocar diferentes prefabs de edificios para esto generamos un índice aleatorio dentro del rango del array “buildingPrefab”, este índice se utiliza para seleccionar un prefab aleatorio del array y lo mismo hacemos con “semaforoPrefab”.

Conclusión

David:

Este proyecto nos logró dar una mejor percepción de cómo trabajan los ambientes multiagentes, es decir, varias entidades decidiendo tomar acciones al mismo instante, algo que intentamos recrear en nuestro modelo, lo cual intentamos recrear en nuestro ambiente con activación random por agente. Me llevo de esta parte grandes ideas, implementaciones futuras y generalmente me abrió el cerebro a jugar más con la inteligencia artificial.

Otra de las cosas que logramos observar a fondo fue como las computadoras de hoy en día nos pueden entregar todos los GUIs, al igual que los videojuegos y otros ambientes gráficos despliegan figuras, objetos e imágenes, simulando en ellos sombras, profundidad y reflexiones de luz para llegar a dar una imagen más clara de la realidad de como lo vemos y sentimos los seres humanos.

Natalia:

El proyecto que realizamos para este bloque nos permitió familiarizarnos el trabajar con multiagentes, este reto nos demostró cómo podemos darle un uso en la vida cotidiana a proyectos como este y la importancia de ellos.

Codes

Server.py

```
from flask import Flask, request, jsonify
from legoCity_Agents.model import CityModel
from legoCity_Agents.agent import Car, Traffic_Light, Destination, Obstacle, Road, Car_Generator

app = Flask("Traffic example")

@app.route('/init', methods=['POST'])
def initModel():
    global randomModel, currentStep

    currentStep = 0

    if request.method == 'POST':
        randomModel = CityModel()

        return jsonify({"message":"Model initialized."})

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

    if request.method == 'GET':
        # carPositions = [{"id": str(s.unique_id), "x": x, "y":1, "z":z} for a, (x, z) in
randomModel.grid.coord_iter() for s in a if isinstance(s, Car)]
        carPositions = [{"id": str(a.unique_id), "x": a.pos[0], "y":1, "z":a.pos[1]} for a in
randomModel.schedule.agents if isinstance(a, Car)]

        return jsonify({'positions':carPositions})

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

    if request.method == 'GET':
        DestinationsPositions = [{"id": str(s.unique_id), "x": x, "y":1, "z":z} for a, (x, z) in
randomModel.grid.coord_iter() for s in a if isinstance(s, Destination)]
        # DestinationsPositions = [{"id": str(a.unique_id), "x": a.pos[0], "y":1, "z":a.pos[1]} for a in
randomModel.schedule.agents if isinstance(a, Destination)]
```

```
        return jsonify({'positions':DestinationsPositions})

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

    if request.method == 'GET':
        trafficLights = [{"id": str(a.unique_id), "x": a.pos[0], "y":1, "z":a.pos[1],
                        "state": a.state, "direction": a.direction,
                        "timeToChange": a.timeToChange}
                        for a in randomModel.schedule.agents if isinstance(a, Traffic_Light)]

        return jsonify({'positions':trafficLights})

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

    if request.method == 'GET':
        obstaclePositions = [{"id": str(s.unique_id), "x": x, "y":1, "z":z} for a, (x, z) in
randomModel.grid.coord_iter() for s in a if isinstance(s, Obstacle)]

        return jsonify({'positions':obstaclePositions})


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

    if request.method == 'GET':
        roadPositions = [{"id": str(s.unique_id), "x": x, "y":1, "z":z} for a, (x, z) in randomModel.grid.coord_iter()
for s in a if isinstance(s, (Road, Traffic_Light, Car_Generator))]

        return jsonify({'positions':roadPositions})

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

    if request.method == 'GET':
        arrivedCars = [{"id": str(a.unique_id)} for a in randomModel.arrivedCarsList]

        return jsonify({'positions':arrivedCars})
```

```python
@app.route('/update', methods=['GET'])
def updateModel():
    global currentStep, randomModel
    if request.method == 'GET':
        randomModel.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)
```

AgentController.cs

```
// TC2008B. Sistemas Multiagentes y Gráficas Computacionales
// C# client to interact with Python. Based on the code provided by Sergio Ruiz.
// Octavio Navarro. October 2023

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

[Serializable]
public class AgentData{
    public string id, direction;
    public float x, y, z;
    public bool state;

    // Diferentes tipos de constructores para los diferentes tipos de datos que se pueden recibir
    public AgentData(string id){
        this.id = id;
    }

    public AgentData(string id, float x, float y, float z){
        this.id = id;
        this.x = x;
        this.y = y;
        this.z = z;
    }
```

```
    public AgentData(string id, float x, float y, float z, bool state, string direction){
        this.id = id;
        this.x = x;
        this.y = y;
        this.z = z;
        this.state = state;
        this.direction = direction;
    }
}

[Serializable]
public class AgentsData{
    public List<AgentData> positions;

    public AgentsData() => this.positions = new List<AgentData>();
}

public class AgentController : MonoBehaviour{
    /*
    The AgentController class is used to control the agents in the simulation.

    Attributes:
        serverUrl (string): The url of the server.
        getAgentsEndpoint (string): The endpoint to get the agents data.
        getObstaclesEndpoint (string): The endpoint to get the obstacles data.
        sendConfigEndpoint (string): The endpoint to send the configuration.
        updateEndpoint (string): The endpoint to update the simulation.
        agentsData (AgentsData): The data of the agents.
        obstacleData (AgentsData): The data of the obstacles.
        agents (Dictionary<string, GameObject>): A dictionary of the agents.
        prevPositions (Dictionary<string, Vector3>): A dictionary of the previous positions of the agents.
        currPositions (Dictionary<string, Vector3>): A dictionary of the current positions of the agents.
        updated (bool): A boolean to know if the simulation has been updated.
        started (bool): A boolean to know if the simulation has started.
        agentPrefab (GameObject): The prefab of the agents.
        obstaclePrefab (GameObject): The prefab of the obstacles.
        floor (GameObject): The floor of the simulation.
        width (int): The width of the simulation.
        height (int): The height of the simulation.
        timeToUpdate (float): The time to update the simulation.
        timer (float): The timer to update the simulation.
        dt (float): The delta time.
    */
    // Endopoints
    string serverUrl = "http://localhost:8585";
    string sendConfigEndpoint = "/init";
    string updateEndpoint = "/update";
    string getAgentsEndpoint = "/getCars";
    string getTrafficLightsEndpoint = "/getTrafficLights";
```

```
string getRoadsEndpoint = "/getRoads";
string getDestinationsEndpoint = "/getDestinations";
string getArrivedCars = "/getArrivedCars";
string getObstaclesEndpoint = "/getObstacles";
// Posibles AgentsData types para recoleccion de información
AgentsData agentsData, obstacleData, roadsData, destinationData, ArrivedCars, trafficLightsData;
// Diccionarios
Dictionary<string, GameObject> agentsDIC, trafficLightsDIC;
Dictionary<string, Vector3> prevPositions, currPositions;

// Varaiables para el inicio y update
bool updated = false, started = false;

// Prefabs
[SerializeField] GameObject[] obstaclePrefab;
public GameObject agentPrefab, floor, trafficLightPrefab;
// Lista para borrar coches
List<GameObject> deleteCars;
// Variables para el update
public float timeToUpdate = 5.0f;
private float timer, dt;

void Start(){
    agentsData = new AgentsData();
    obstacleData = new AgentsData();
    destinationData = new AgentsData();
    roadsData = new AgentsData();

    deleteCars = new List<GameObject>();

    prevPositions = new Dictionary<string, Vector3>();
    currPositions = new Dictionary<string, Vector3>();

    agentsDIC = new Dictionary<string, GameObject>();
    trafficLightsDIC = new Dictionary<string, GameObject>();

    timer = timeToUpdate;

    // Launches a couroutine to send the configuration to the server.
    StartCoroutine(SendConfiguration());
}

private void Update(){
    if(timer < 0){
        timer = timeToUpdate;
        updated = false;
        StartCoroutine(UpdateSimulation());
    }
```

```
    if (updated){
        timer -= Time.deltaTime;
        dt = 1.0f - (timer / timeToUpdate);

        // Iterates over the agents to update their positions.
        // The positions are interpolated between the previous and current positions.
        foreach(var agent in currPositions){
            Vector3 currentPosition = agent.Value;
            Vector3 previousPosition = prevPositions[agent.Key];

            Vector3 interpolated = Vector3.Lerp(previousPosition, currentPosition, dt);
            Vector3 direction = currentPosition - interpolated;

            agentsDIC[agent.Key].GetComponent<moveCar>().ApplyTransforms(interpolated, direction);
            // agentsDIC[agent.Key].transform.localPosition = interpolated;
            // if(direction != Vector3.zero) agentsDIC[agent.Key].transform.rotation = Quaternion.LookRotation(direction);
        }

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

IEnumerator UpdateSimulation(){
    UnityWebRequest www = UnityWebRequest.Get(serverUrl + updateEndpoint);
    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
        Debug.Log(www.error);
    else{
        StartCoroutine(GetAgentsData());
        StartCoroutine(GetDestroyCars());
        StartCoroutine(ChangeTrafficLights());
        DestroyCars();
    }
}

IEnumerator SendConfiguration(){
    /*
    The SendConfiguration method is used to send the configuration to the server.

    It uses a WWWForm to send the data to the server, and then it uses a UnityWebRequest to send the form.
    */
    WWWForm form = new WWWForm();

    UnityWebRequest www = UnityWebRequest.Post(serverUrl + sendConfigEndpoint, 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");

        // Once the configuration has been sent, it launches a coroutine to get the agents data.
        // Inicializar todo el mapa
        StartCoroutine(GetTrafficLights());
        StartCoroutine(GetAgentsData());
        StartCoroutine(GetObstacleData());
        StartCoroutine(GetDestinationsData());
        StartCoroutine(GetRoadData());
    }
}

IEnumerator GetAgentsData(){
    // The GetAgentsData method is used to get the agents data from the server.

    UnityWebRequest www = UnityWebRequest.Get(serverUrl + getAgentsEndpoint);
    yield return www.SendWebRequest();

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

        foreach(AgentData agent in agentsData.positions){
            Vector3 newAgentPosition = new Vector3(agent.x, agent.y, agent.z);

            Vector3 currentPosition = new Vector3();
            // Esto es que encuentre si existe en el diccionario
            if(currPositions.TryGetValue(agent.id, out currentPosition))
                prevPositions[agent.id] = currentPosition;
            else{ // Si no existe, que lo cree y que lo agregue al diccionario
                prevPositions[agent.id] = newAgentPosition;
                agentsDIC[agent.id] = Instantiate(agentPrefab, Vector3.zero, Quaternion.identity);
                agentsDIC[agent.id].GetComponent<moveCar>().Init();
                agentsDIC[agent.id].GetComponent<moveCar>().ApplyTransforms(newAgentPosition, Vector3.zero);
            }
            currPositions[agent.id] = newAgentPosition;
        }

        updated = true;
        if(!started) started = true;
    }
}
```

```
IEnumerator GetObstacleData(){
    UnityWebRequest www = UnityWebRequest.Get(serverUrl + getObstaclesEndpoint);
    yield return www.SendWebRequest();

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

        foreach(AgentData obstacle in obstacleData.positions){
            int rand = UnityEngine.Random.Range(0, obstaclePrefab.Length);
            Instantiate(obstaclePrefab[rand], new Vector3(obstacle.x, obstacle.y, obstacle.z), Quaternion.identity);
            Instantiate(floor, new Vector3(obstacle.x, obstacle.y - 0.04f, obstacle.z), Quaternion.identity);
        }
    }
}

void DestroyCars(){
    foreach(GameObject car in deleteCars){
        // Destruir objeto
        Destroy(car);
        // Quitarlo de la lista
        deleteCars.Remove(car);
    }
}

IEnumerator GetDestroyCars(){
    UnityWebRequest www = UnityWebRequest.Get(serverUrl + getArrivedCars);
    yield return www.SendWebRequest();

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

        foreach(AgentData arrivedCar in ArrivedCars.positions){
            // Agregarlo a la lista de coches a destruir
            deleteCars.Add(agentsDIC[arrivedCar.id]);
            // Eliminarlo de los diccionarios
            agentsDIC.Remove(arrivedCar.id);
            currPositions.Remove(arrivedCar.id);
            prevPositions.Remove(arrivedCar.id);
        }
    }
}

IEnumerator GetDestinationsData(){
    UnityWebRequest www = UnityWebRequest.Get(serverUrl + getDestinationsEndpoint);
```

```
        yield return www.SendWebRequest();

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

            foreach(AgentData obstacle in destinationData.positions){
                int rand = UnityEngine.Random.Range(0, obstaclePrefab.Length);
                GameObject tile = Instantiate(obstaclePrefab[rand], new Vector3(obstacle.x, obstacle.y, obstacle.z),
Quaternion.identity);
                tile.GetComponentInChildren<Renderer>().materials[0].color = Color.red;
                Instantiate(floor, new Vector3(obstacle.x, obstacle.y - 0.04f, obstacle.z), Quaternion.identity);
            }
        }
    }

    IEnumerator GetRoadData(){
        UnityWebRequest www = UnityWebRequest.Get(serverUrl + getRoadsEndpoint);
        yield return www.SendWebRequest();

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

            foreach(AgentData obstacle in roadsData.positions){
                Instantiate(floor, new Vector3(obstacle.x, obstacle.y - 0.04f, obstacle.z), Quaternion.identity);
            }
        }
    }

    IEnumerator GetTrafficLights(){
        UnityWebRequest www = UnityWebRequest.Get(serverUrl + getTrafficLightsEndpoint);
        yield return www.SendWebRequest();

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

            foreach(AgentData tf in trafficLightsData.positions){
                if (tf.direction == "Right"){
                    trafficLightsDIC[tf.id] = Instantiate(trafficLightPrefab, new Vector3(tf.x, tf.y, tf.z - 0.5f), Quaternion.Euler(-90,
0, 0));
                    trafficLightsDIC[tf.id].GetComponentInChildren<Light>().color = Color.red;
                }
                else if (tf.direction == "Left"){
```

```
                trafficLightsDIC[tf.id] = Instantiate(trafficLightPrefab, new Vector3(tf.x, tf.y, tf.z - 0.5f), Quaternion.Euler(-90,
180, 0));
                trafficLightsDIC[tf.id].GetComponentInChildren<Light>().color = Color.red;
            }
            else if (tf.direction == "Down"){
                trafficLightsDIC[tf.id] = Instantiate(trafficLightPrefab, new Vector3(tf.x, tf.y, tf.z - 0.5f), Quaternion.Euler(-90,
0, 0));
                trafficLightsDIC[tf.id].GetComponentInChildren<Light>().color = Color.red;
            }
            else if (tf.direction == "Up"){
                trafficLightsDIC[tf.id] = Instantiate(trafficLightPrefab, new Vector3(tf.x, tf.y, tf.z - 0.5f), Quaternion.Euler(-90,
-90, 0));
                trafficLightsDIC[tf.id].GetComponentInChildren<Light>().color = Color.red;
            }
            else{
                trafficLightsDIC[tf.id] = Instantiate(trafficLightPrefab, new Vector3(tf.x, tf.y, tf.z - 0.5f), Quaternion.Euler(-90,
0, 0));
                trafficLightsDIC[tf.id].GetComponentInChildren<Light>().color = Color.red;
            }
        }
    }
  }

  IEnumerator ChangeTrafficLights(){
    UnityWebRequest www = UnityWebRequest.Get(serverUrl + getTrafficLightsEndpoint);
    yield return www.SendWebRequest();

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

        foreach(AgentData tf in trafficLightsData.positions){
            if(tf.state){
                trafficLightsDIC[tf.id].GetComponentInChildren<Light>().color = new Color32(61, 161, 27, 255);
            }
            else{
                trafficLightsDIC[tf.id].GetComponentInChildren<Light>().color = Color.red;
            }
        }
    }
  }
}
```