Tecnológico de Monterrey

Juan Pablo Moreno Robles Arenas - A01374091
Juan Pablo Cruz Rodriguez - A01783208

Modelación de sistemas multiagentes con gráficas computacionales

Reto Movilidad Urbana

Introducción:

Para esta problemática tuvimos que diseñar una simulación de movilidad urbana, utilizando Unity y la librería Mesa en Python. Para llevar a cabo este proyecto, dirigimos nuestra atención inicial hacia la definición del comportamiento de los agentes, que representan los vehículos en tránsito dentro de la simulación. Estos agentes reciben instrucciones detalladas que abarcan desde cómo deben desplazarse por el entorno en forma de cuadrícula hasta las leyes de tráfico que deben respetar para así poder navegar de manera apropiada por la ciudad.

Las leyes incluyen el respeto a las luces de los semáforos, la adecuada circulación en el sentido de las calles y la habilidad para maniobrar entre otros vehículos, todo implementado con la librería Mesa. Esta herramienta nos brinda la flexibilidad necesaria para modelar y simular comportamientos de los agentes que se aproximen a la vida real, permitiendo a nuestros agentes tomar decisiones informadas en tiempo real y adaptarse a las cambiantes condiciones del entorno.

Paralelamente, en Unity nos dedicamos a la construcción del entorno virtual que albergará nuestra simulación. Aprovechando nuestro conocimiento en gráficas computacionales, damos forma a una ciudad digital que refleja a una ciudad real. Además utilizamos matrices de transformación para dictar la orientación de nuestro modelo del carro.

En síntesis, nuestra labor abarca desde la programación del comportamiento de los agentes con la librería Mesa hasta la creación de un entorno realista en Unity, todo con el objetivo de ofrecer una simulación de movilidad urbana que no solo sea técnicamente sólida, sino también fiel a la complejidad del tráfico en entornos urbanos.

Diseño de Agentes:

El objetivo del agente Carro, que es nuestro agente principal, es navegar por la ciudad hasta llegar a su destino, tomando en cuenta las instrucciones que se les dio para poder cumplir con este objetivo. Para esto el agente ya sabe que como llegar a su destino por medio del algoritmo de A*, el cual le ayudará a encontrar el camino óptimo. Además le dimos reglas que debe cumplir para poder navegar por la ciudad. Una de ellas siendo que debe estarse moviendo de acuerdo al sentido de la calle en la que está. También tiene que saber maniobrar con respecto a lo que encuentra a su entorno, puede ver dos celdas adelante en su entorno, es decir si ve algún semáforo este lo debe de identificar y saber en qué estado está, si es rojo el carro debe pararse o verde que le indica que puede seguir moviéndose. También si ve que otro carro está parado en un semáforo, este lo debe de detectar y quedarse parado atrás. Sí ve que la celda que está al lado del

carro parado está vacía, el agente debe poder reconocer esto y moverse ahí para que puedan caber más carros dentro de la simulación. Y finalmente el carro al llegar a su destino debe de desaparecer al momento de llegar para así que la simulación pueda tener un flujo bien estructurado.

Objetivo:

- Su objetivo es navegar por la ciudad respetando las reglas que le damos de comportamiento.

Capacidad efectora:

- La capacidad efectora de este agente se encuentra principalmente en sus métodos move, **move_around**, stop, **see_enviroment** y **see_traffic_light**. Estos métodos determinan cómo el agente se mueve en el entorno, reacciona ante otros agentes como carros o semáforos, y evalúa su visión del entorno para tomar decisiones.

Percepción:

- El agente percibe su entorno a través de la visión de sus vecinos y determina su dirección actual. Estos elementos de percepción se utilizan posteriormente en otras funciones para que el agente tome decisiones basadas en la información que percibe en su entorno.

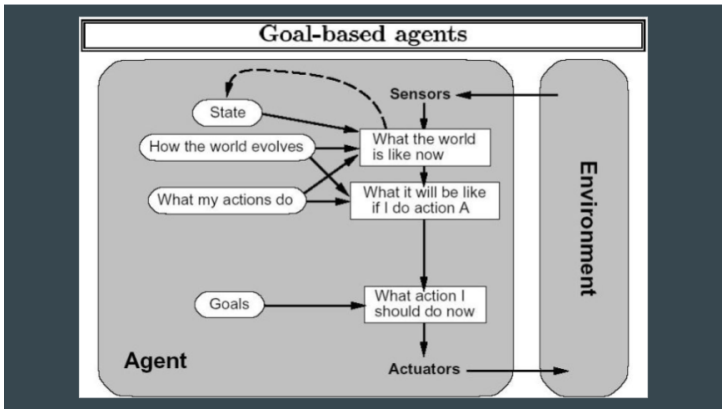
Proactividad:

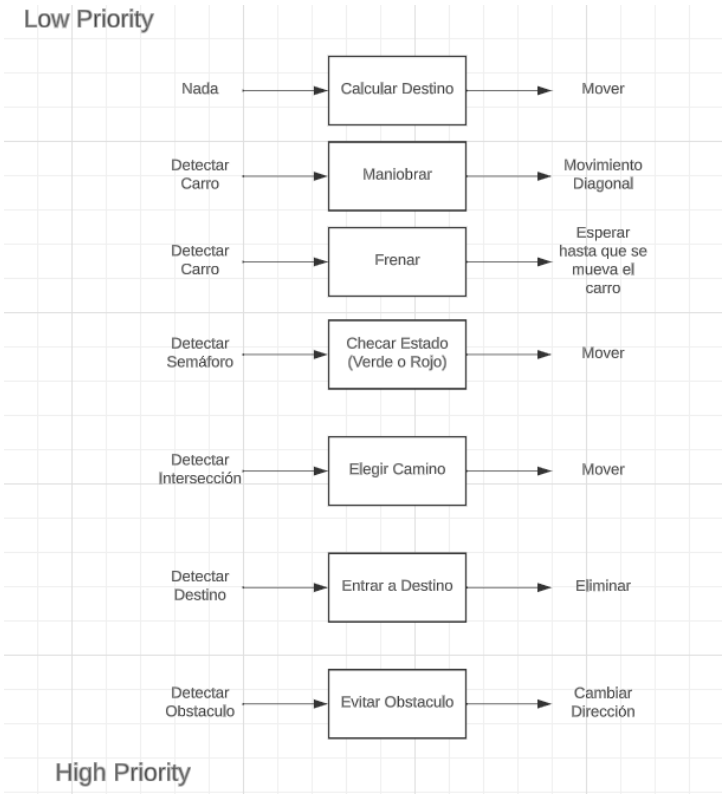
- El agente recalcula su ruta con A* cada vez que esquiva un carro. También cada vez que se mueve fuera de su ruta predeterminada recalcula su ruta principal
  métricas de desempeño, etc.

Métricas de Desempeño:

- Nuestras métricas de desempeño son que tan efectivo fue nuestra simulación para el tránsito fluido de la simulación, es decir que no chocaran nuestros carros y que llegaran bien a su destino.

Arquitectura de Subsunción de los Agentes





Optamos por diseñar la arquitectura del agente de coche basada en objetivos, ya que su principal meta es llegar a su destino de manera eficiente. Para lograrlo, utiliza el Algoritmo de A* para encontrar la ruta óptima, al tiempo que respeta las normas de tránsito para evitar colisiones con otros vehículos, detenerse en semáforos y evadir obstáculos. Además, el agente monitorea el estado de la calle en la que se encuentra, evaluando las condiciones de tráfico y otros factores relevantes. Todo esto se logra mediante sensores que proporcionamos al agente, permitiéndole

discernir la acción óptima a tomar con base en la información que recopila de las celdas dentro de su rango de visión.

Características del Ambiente

El entorno en el que opera nuestro agente de coche es accesible, ya que posee conocimiento sobre la ruta que debe seguir para alcanzar su destino y comprende el comportamiento que debe adoptar al encontrarse con distintos tipos de agentes a lo largo de su trayecto. Se caracteriza como un entorno determinista, dado que cada acción emprendida por el agente reduce progresivamente la distancia que le resta para alcanzar su objetivo. Además, estas acciones también impactan en los demás agentes de coche presentes en la cuadrícula que se mueven en su entorno. También es considerado como Discreto por qué tiene una cantidad finita de acciones que puede tomar el agente hasta llegar a su destino.

Este entorno se considera episódico, ya que las decisiones del agente se basan en el paso específico en el que se encuentra. Finalmente, el entorno es estático, pues permanece sin cambios a menos que el agente realice acciones específicas.

Unity

Para la parte Unity, se utilizó este ambiente para darle forma a nuestra simulación, donde aplicamos lo visto en el apartado de la unidad de formación que era Gráficas Computacionales.
Todo esto se logró dentro del script de **AgentController** que utilizamos como base para llamar a los endpoints que se crean en nuestro servidor de Flask que está recibiendo la información de Mesa. Además se utilizaron matrices de transformación para los vértices del auto y las llantas. También llamamos estos endpoints dentro de nuestro código para poder dar forma a nuestra ciudad tomando los edificios, destino, calles, semáforos etc. Por ejemplo para los semáforos instanciamos un prefab que tiene un hijo Light que va estar recibiendo el estado agente de mesa para así cambiar el color del semáforo al apropiado. Al de Agente iteramos sobre la lista de posiciones de agentes, para así crear vectores de posiciones que se inicializan en la posición (0,0,0). Se realizan una serie de verificaciones y actualizaciones en función si el agente existe.Y finalmente los obstacle agents toman una lista de prefabs que escoge uno de manera aleatoria y lo instancia. También le dimos orientación a las calles para que se vea de manera correcta en la dirección de la calle.

```
IEnumerator GetAgentsData()
{


    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);
        // Debug.Log(agentsData.positions);

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

                if(agents.ContainsKey(agent.id))
                {
                    if(agent.Direction == "Destino Alcanzado"){
                        Destroy(agents[agent.id]);
                        agents.Remove(agent.id);
                    }
                    else{
                        Vector3 currentPosition = new Vector3();
                        if(currPositions.TryGetValue(agent.id, out currentPosition))
                            prevPositions[agent.id] = currentPosition;
                            currPositions[agent.id] = newAgentPosition;
```

```
                }
                else
                {
                    //? Aqui en el Quaternion.identity toca inicializar con respecto a su direccion
                    prevPositions[agent.id] = newAgentPosition;
                    agents[agent.id] = Instantiate(agentPrefab, new Vector3(0,0,0), Quaternion.identity);
                }
        }
        //Activamos el update y el start
        updated = true;
        if(!started) started = true;
    }
}
```

Enumerator que instancia el Agente.

```
1 reference
IEnumerator GetObstacleData()//Inicializa los edificios
{
    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);

        // Debug.Log(obstacleData.positions);

        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);
        }
    }
}
```

Enumerator para los Obstáculos y le da una lista de prefabs.

```
foreach(AgentData trafficLight in trafficLightsData.positions)
{
    bool state = trafficLight.state;
    if(!agents.ContainsKey(trafficLight.id)){
        if(trafficLight.Direction == "Left"){
            agents[trafficLight.id] = Instantiate(trafficLightPrefab, new Vector3(trafficLight.x, trafficLight.y, trafficLight.z), Quaternion.Euler(0,360,0));
            Instantiate(roadPrefab, new Vector3(trafficLight.x, trafficLight.y, trafficLight.z), Quaternion.Euler(0,360,0));
            Light lightComponent = agents[trafficLight.id].GetComponentInChildren<Light>();
        }
        else if(trafficLight.Direction == "Right"){
            agents[trafficLight.id] = Instantiate(trafficLightPrefab, new Vector3(trafficLight.x, trafficLight.y, trafficLight.z), Quaternion.Euler(0,180,0));
            Instantiate(roadPrefab, new Vector3(trafficLight.x, trafficLight.y, trafficLight.z), Quaternion.Euler(0,180,0));
            Light lightComponent = agents[trafficLight.id].GetComponentInChildren<Light>();
        }
        else if(trafficLight.Direction == "Up"){
            agents[trafficLight.id] = Instantiate(trafficLightPrefab, new Vector3(trafficLight.x, trafficLight.y, trafficLight.z), Quaternion.Euler(0,90,0));
            Instantiate(roadPrefab, new Vector3(trafficLight.x, trafficLight.y, trafficLight.z), Quaternion.Euler(0,90,0));
            Light lightComponent = agents[trafficLight.id].GetComponentInChildren<Light>();
        }
        else if(trafficLight.Direction == "Down"){
            agents[trafficLight.id] = Instantiate(trafficLightPrefab, new Vector3(trafficLight.x, trafficLight.y, trafficLight.z), Quaternion.Euler(0,270,0));
            Instantiate(roadPrefab, new Vector3(trafficLight.x, trafficLight.y, trafficLight.z), Quaternion.Euler(0,270,0));
            Light lightComponent = agents[trafficLight.id].GetComponentInChildren<Light>();
        }
    }
```

```
            else
            {
                if (state == true){
                    Light lightComponent = agents[trafficLight.id].GetComponentInChildren<Light>();
                    lightComponent.color = Color.green;
                }
                else{
                    Light lightComponent = agents[trafficLight.id].GetComponentInChildren<Light>();
                    lightComponent.color = Color.red;
                }
            }
        }
    }
```

Enumerator para dar la orientación a los semáforos y manejar el estado que está recibiendo desde mesa por el servidor flask.

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

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

        // Debug.Log(roadData.positions);
        foreach(AgentData road in roadData.positions)
        {
            if(road.Direction == "Left"){      Saikou17, 4 hours ago • Unity Final/Mesa v.2
                Instantiate(roadPrefab, new Vector3(road.x, road.y, road.z), Quaternion.Euler(0,360,0));
            }
            else if(road.Direction == "Right"){
                Instantiate(roadPrefab, new Vector3(road.x, road.y, road.z), Quaternion.Euler(0,180,0));
            }
            else if(road.Direction == "Up"){
                Instantiate(roadPrefab, new Vector3(road.x, road.y, road.z), Quaternion.Euler(0,90,0));
            }
            else if(road.Direction == "Down"){
                Instantiate(roadPrefab, new Vector3(road.x, road.y, road.z), Quaternion.Euler(0,270,0));
            }
             else if(road.Direction == "IntersectionUp"){
                Instantiate(roadPrefab, new Vector3(road.x, road.y, road.z), Quaternion.Euler(0,90,0));
            }
            else if(road.Direction == "IntersectionDown"){
                Instantiate(roadPrefab, new Vector3(road.x, road.y, road.z), Quaternion.Euler(0,270,0));
            }
```

Enumerator que se utiliza para darle sentidos a las calles.

Conclusiones

En resumen, este proyecto representó un desafío significativo y enriquecedor. La implementación de la amplia gama de información adquirida a lo largo de esta unidad de formación requirió un esfuerzo intenso, pero los resultados finales demostraron ser exitosos al lograr generar una simulación que cumplía con los requisitos establecidos.

Uno de los aspectos más desafiantes de nuestro trabajo fue la consolidación de la información, especialmente al integrar los componentes desarrollados por separado en Mesa y Unity. La complejidad radica en comprender a fondo el proceso de utilización de los "gets" desde nuestro servidor para invocar a los agentes implementados. Además, enfrentamos el reto de aplicar estos conceptos de manera efectiva en nuestro código de Unity. La coordinación entre los distintos elementos del proyecto demandó una comprensión profunda de la interconexión entre Mesa y Unity, así como la habilidad para integrar de manera fluida las funciones del servidor.

En general, creemos que fue un reto bastante interesante y que nos dio una pequeña visión de lo que es la programación agentes y el uso de gráficas computacionales, que hoy en día se utilizan en todo tipo de entretenimiento que se utiliza hoy en día. Por lo cuál nos llevamos aprendizaje basto que podemos utilizar a futuro en nuestra carrera.