Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Santa Fe

Modelación de sistemas multiagentes con gráficas computacionales
TC2008B, Grupo 301

Reporte del reto

Equipo 1:

Nombre	Matricula:
Alan Anthony Hernández Pérez	A01783347
Oswaldo Ilhuicatzi Mendizábal	A01781988

Problemática y solución propuesta

La movilidad urbana se define como la habilidad de transportarse de un lugar a otro y es fundamental para el desarrollo económico y social y la calidad de vida de los habitantes de una ciudad. Desde hace un tiempo, asociar la movilidad con el uso del automóvil ha sido un signo distintivo de progreso. Sin embargo, esta asociación ya no es posible hoy. El crecimiento y uso indiscriminado del automóvil —que fomenta políticas públicas erróneamente asociadas con la movilidad sostenible—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.

Así, para que México pueda estar entre las economías más grandes del mundo, es necesario mejorar la movilidad en sus ciudades, lo que es crítico para las actividades económicas y la calidad de vida de millones de personas.

Es por esto que en este reto se busca plantear una solución al problema de movilidad urbana en México, esto a través de un enfoque que reduzca la congestión vehicular al simular de manera gráfica el tráfico, representando la salida de un sistema multi agentes simulado en python con la biblioteca de mesa a través de Unity en un modelo 3D.

Diseño de los agentes

Dentro del cuadro del retro, los agentes, es decir, los automóviles de la simulación, deben ser capaces de llegar a su destino de manera óptima, evidentemente tomando en cuenta el comportamiento racional de evitar causar colisiones. Por otro lado, la capacidad efectora de los agentes se basa en avanzar en los sentidos de las vías principales, en donde su movimiento siempre es hacia adelante, girar izquierda o derecha dependiendo del cruce, respetar los semáforos, detección de automóviles vecinos y, de ser necesario, el cómo cambiarse de carril de manera civilizada. Es así que los agentes buscarán tomar las acciones que maximicen sus medidas de desempeño y que les ayude a llegar a su destino de manera eficiente, es en donde se entra en el concepto de su proactividad.

Por lo tanto, es de vital importancia el analizar la métrica de desempeño del sistema para medir el éxito que tengan, lo cual será visualizado en la simulación al constatar que, en efecto, no se haya generado tráfico en las vías de circulación indicando que los agentes presentan un correcto desempeño.

Arquitectura de subsunción

- Evadir obstáculo
    - Si encuentra un obstáculo, cambia de dirección.

- Esperar automóvil
    - Si encuentra un automóvil enfrente, esperar a que emprenda su camino para que avance el coche.

- Semáforo
    - Si se encuentra en un semáforo y la luz está en verde, avanzar.
    - Si se encuentra en un semáforo y la luz está en rojo, esperar en su posición actual a que cambie a verde.

- Cambio de carril
    - Si es posible, cambiar de carril.
    - Si no es posible, seguir avanzando en su mismo carril.

- Destino alcanzado
    - Si se encuentra en su destino, deja de avanzar y desaparece de la simulación.

- Conducir en vías principales
    - Si verdadero, avanzar en la dirección de su carril.

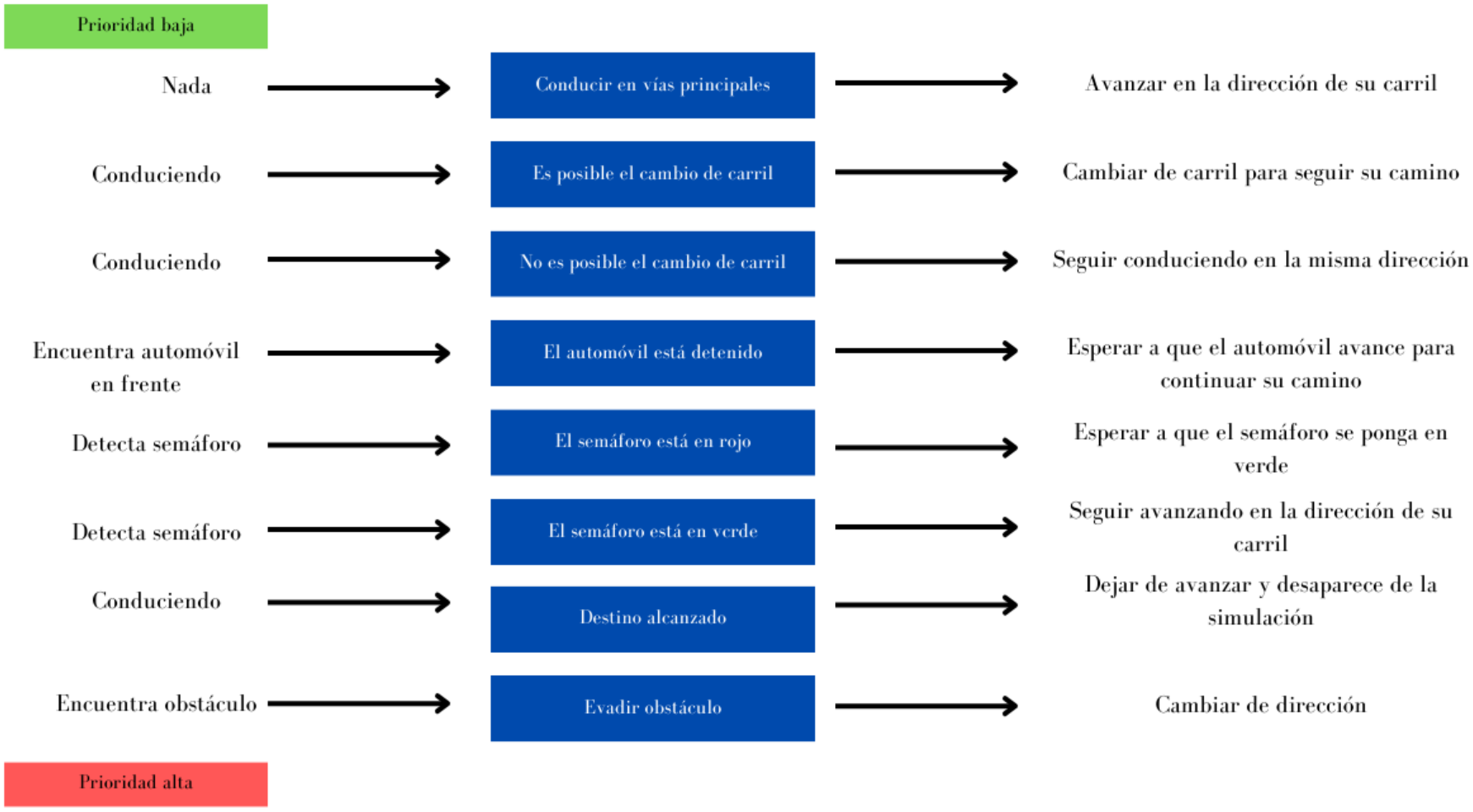

Diagrama 1. Arquitectura de subsunción

Características del ambiente

Dentro de la simulación, se trata de un ambiente inaccesible dado que los agentes no cuentan con el conocimiento completo de su entorno y su estado, por lo que no pueden saber el nivel de tráfico que existe en las calles. Cuando se habla de si el ambiente es determinista o no, depende mucho de las situaciones aleatorias que suceden en el ambiente. Dentro de este contexto, toda la simulación está programada para que se lleguen a los mismos resultados cada vez que se ejecute, es decir, que los autos lleguen a su destino. Sin embargo, las rutas que toman se producen de manera diferente y sus destinos elegidos cambian de manera aleatoria, por lo que se trata de un ambiente no determinista.

Asimismo, se trata de un ambiente episódico y discreto ya que se trabaja por steps, además de que existe una cantidad finita de acciones distintas dentro de la simulación que realizan los automóviles. Finalmente, el ambiente es dinámico dado que existen adaptaciones dentro de la simulación y las interacciones entre automóviles como el saber cuándo esperar a que avance su vecino, al igual que los cambios en el tiempo de los semáforos y la generación de múltiples agentes coexistiendo en el entorno.

Unity

Para la implementación del reto se trabajó con el motor de Unity, en específico con ayuda de los códigos proporcionados por los profesores y con modificaciones por parte del equipo de trabajo.

En cuanto al script de *AgentController,* se cambió para que utilizara las matrices de transformación en los vértices del auto y las llantas, así como la destrucción del objeto del auto cuando llega a su destino y que desaparezca de la simulación. Esto es enviado como información del endpoint por parte de mesa con *flask*.

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

                    if(!agents.ContainsKey(agent.id))
                    {
                        prevPositions[agent.id] = newAgentPosition;
                        agents[agent.id] =
Instantiate(agentPrefab[UnityEngine.Random.Range(0,agentPrefab.Length)]
, new Vector3(0,0,0), Quaternion.identity);
```

```
                        Apply_Transform applyTransform =
agents[agent.id].GetComponent<Apply_Transform>();
                        applyTransform.SetNewPos(newAgentPosition);
                        applyTransform.SetNewPos(newAgentPosition);
                        applyTransform.moveTime = timeToUpdate;
                    }
                    //if the agent is in the intermediate state, it is
destroyed
                    else if(agent.state == "intermediate"){
                        Debug.Log("wheels destroyed of the car " +
agent.id);
                        Apply_Transform applyTransform =
agents[agent.id].GetComponent<Apply_Transform>();
                        applyTransform.DestroyLlantas();
                        Destroy(agents[agent.id]);
                        Debug.Log("Agent " + agent.id + " died");
                        agents.Remove(agent.id);
                        prevPositions.Remove(agent.id);
                        currPositions.Remove(agent.id);
                    }
                    else
                    {
                        Apply_Transform applyTransform =
agents[agent.id].GetComponent<Apply_Transform>();
                        applyTransform.SetNewPos(newAgentPosition);

                    }
            }
```

Por otro lado, se tuvo que generar un nuevo endpoint para posicionar de manera adecuada los semáforos con su orientación correspondiente, es decir, en orden opuesto al sentido de las calles. Asimismo, se añadió un componente de luz para que los semáforos cambien a verde o rojo dependiendo del estado de ese agente desde *mesa*, pudiendo así modificarlo en Unity.

```
if (!agents.ContainsKey(trafficLight.id))
                {
                    if(trafficLight.type == "s")
                    {

                        prevPositions[trafficLight.id] =
newTrafficLightPosition;
```

```
                        agents[trafficLight.id] =
Instantiate(trafficLightPrefab, newTrafficLightPosition,
Quaternion.identity);
                        Debug.Log("Semaphore " + trafficLight.id + "
created");
                    }

                    else if(trafficLight.type == "S")
                    {
                        prevPositions[trafficLight.id] =
newTrafficLightPosition;
                        agents[trafficLight.id] =
Instantiate(trafficLightPrefab, newTrafficLightPosition,
Quaternion.Euler(0, 90, 0));
                        Debug.Log("Semaphore " + trafficLight.id + "
created");
                    }
                    else if(trafficLight.type == "X")
                    {
                        prevPositions[trafficLight.id] =
newTrafficLightPosition;
                        agents[trafficLight.id] =
Instantiate(trafficLightPrefab, newTrafficLightPosition,
Quaternion.Euler(0, -90, 0));
                        Debug.Log("Semaphore " + trafficLight.id + "
created");
                    }
                    else
                    {
                        prevPositions[trafficLight.id] =
newTrafficLightPosition;
                        agents[trafficLight.id] =
Instantiate(trafficLightPrefab, newTrafficLightPosition,
Quaternion.Euler(0, 180, 0));
                        Debug.Log("Semaphore " + trafficLight.id + "
created"+ "with type" + trafficLight.type);
                    }

                    if (trafficLight.light)

agents[trafficLight.id].GetComponent<Light>().color = Color.green;
                    else
```

```
agents[trafficLight.id].GetComponent<Light>().color = Color.red;
                }
                else
                {
                    if(trafficLight.light)


agents[trafficLight.id].GetComponent<Light>().color = Color.green;
                    else

agents[trafficLight.id].GetComponent<Light>().color = Color.red;
                }
```

El código de Apply transform es responsable de que los vértices del auto se muevan según la posición final asignada desde el Agent controller, además de instanciar las llantas correspondientes a cada modelo de coche. Así, cada auto tiene una posición inicial y final a manera de que se vaya actualizando su posición.

```
    public void SetNewPos(Vector3 newPos){
        startPos = stopPos;
        stopPos = newPos;
        if(startPos != stopPos){
            angle = GetAngle(stopPos-startPos);
        }
        elapsedTime = 0f;
    }
```

Después, ahora existe un diccionario que contiene las llantas, asegurando que de esta manera se eliminen de la escena de Unity cuando el coche llegue a su destino. Cabe mencionar que se cuenta con una variable de tiempo inicial, tiempo con el que se mueven y un retraso (*delay*).

```
    [SerializeField] Vector3 startPos;
    [SerializeField] Vector3 stopPos;
    [SerializeField] float ti = 0f;
    public float moveTime = 0f;
    [SerializeField] float elapsedTime = 0f;

    [Header("Transform of wheels")]
    [SerializeField] float speedRotation;
    [SerializeField] GameObject llanta;
    [SerializeField] float angle;
    [SerializeField] Vector3[] llantasPos;
```

```
    Dictionary<string, GameObject> llantas = new Dictionary<string,
GameObject>();
```

El último cambio dentro del código sería el de *DoTransform*, en el cual se configura el tiempo inicial, además de realizar ahí mismo la función de interpolación “lerp” para así poder animar el movimiento del auto junto con las llantas.

```
void DoTransform()
    {
        ti = elapsedTime / moveTime;

        Vector3 displacement = startPos + (stopPos - startPos) *
ti;//desplazamiento del auto
        elapsedTime += Time.deltaTime;

        Matrix4x4 move = HW_Transforms.TranslationMat(displacement.x ,
                                                      displacement.y ,
                                                      displacement.z );
```

Finalmente, se modificó que dentro *cityMaker* se genere una lista aleatoria de edificios para que exista diversidad en el mapa.

```
[SerializeField] GameObject[] buildingPrefab;

else if (tiles[i] == 'D') {
                position = new Vector3(x * tileSize, 0, y * tileSize);
                var RandomBuilding = Random.Range(0,
buildingPrefab.Length);
                tile = Instantiate(buildingPrefab[RandomBuilding],
position, Quaternion.Euler(0, 90, 0));
                tile.GetComponent<Renderer>().materials[0].color =
Color.red;
                tile.transform.parent = transform;
                x += 1;
            } else if (tiles[i] == '#') {
                position = new Vector3(x * tileSize, 0, y * tileSize);
                var RandomBuilding = Random.Range(0,
buildingPrefab.Length);
                tile = Instantiate(buildingPrefab[RandomBuilding],
position, Quaternion.identity);
                tile.transform.localScale = new Vector3(1,
Random.Range(0.5f, 2f), 1);
```

```
                tile.transform.parent = transform;
                x += 1;
```

Conclusiones

Al final del recorrido, se puede dar cuenta de que la simulación llevada a cabo para el reto se logró de manera satisfactoria. Sin embargo, entre más coches en la simulación, más posibilidades de fallo, por lo que existen áreas de oportunidad para mejorarla y que sea aún más óptima y realista:

Existía la posibilidad de que se generará un embotellamiento en cierto punto de la simulación, por lo tanto, sería buena opción el trabajar con un nivel de estrés con el que cuenten los autos, es decir, que tengan un tiempo de tolerancia para esperar y si no avanzan, que comiencen a recalcular su ruta con A* para llegar a su destino final, logrando así que se desocupen las calles de manera más eficiente y no existan atascos.

Por último, el aprendizaje que se obtuvo a lo largo del reto fue basta, ya que se logró comprender cómo funcionan las matrices de transformación al programarlas con los profesores y no utilizando las predeterminadas de Unity, pudiendo así entender tanto la geometría, topología e interpolación de los automóviles para el manejo de movimiento dentro de la simulación. Asimismo, se aprendió a trabajar con la biblioteca de *mesa* para generar simulaciones, así como entender el comportamiento de los agentes y las características de su ambiente.