Campus Santa Fe

16/11/2023

Análisis y diseño de algoritmos avanzados (Gpo 602)

Reto: Movilidad Urbana

Santiago Tena | A01781293

Iker Garcia German | A01782767

Problema y solución propuesta

La movilidad urbana es un aspecto esencial de la vida de las personas de una ciudad. Gracias a su buena planeación es que se puede propiciar el desarrollo tanto económico como social. Para este reto se necesitó implementar una simulación con la librería Mesa de Python que lograra evitar la congestión vehicular en un ambiente dado por los profesores, implementando y diseñando el comportamiento adecuado de los agentes involucrados. Finalmente conectar la simulación obtenida al motor de desarrollo de juegos Unity para poder observarla en un modelo en 3D.

Nuestra simulación cuenta con cinco diferentes agentes. Car(Agent), Traffic_Light(Agent), Road(Agent), Destination(Agent) y Obstacle(Agent). De estos cinco solo se configuraron los agentes de los coches y los semáforos ya que en realidad son los únicos que realizan acciones.

Para poder completar el objetivo de encontrar y recalcular la ruta más corta al destino se implementó el algoritmo de A* (a estrella) a través de la librería de Pathfinding de Python. De la misma librería para poder determinar las direcciones se hizo uso de la posibilidad de crear diferentes niveles de grid. Los niveles creados fueron seis. Los primeros cuatro determinan, cada uno, las direcciones de arriba, abajo, izquierda y derecha. Los últimos se usaron para poder asignar direcciones en situaciones específicas como lo son las glorietas. Dentro de cada uno de estos grids se especificó el peso o costo entre las conexiones para que el algoritmo de búsqueda de A* pueda determinar el camino de menor costo.

Diseño de los agentes

<u>Coche</u>

Car(Agent).

Objetivo: llegar al destino escogido de manera aleatoria.

Capacidad Efectora:

- Escoger destino random.
- Calcular la ruta más corta al destino.
- Avanzar en la dirección de la calle.
- Detenerse en semáforos.
- Evitar colisiones con otros agentes y obstáculos.
- Recalcular la ruta en caso de no avanzar cierto número de steps random entre 10 y 19.

Percepción: el agente del coche puede ver un espacio hacia adelante y hacia atrás para saber sí tiene otros agentes cercanos.

Proactividad y Reactividad: el agente del coche es capaz de seleccionar la ruta más corta desde el principio de su camino e incluso cuando lleva cierto tiempo esperando sin moverse

un número de steps random entre 10 y 19 para intentar explorar rutas nuevas (el valor se escoge de manera random para simular diferentes tipos de conductores). Por la parte de reactividad el agente se espera a tener dentro de las casillas disponibles de movimiento, a que se encuentren vacías para realizar el siguiente movimiento.

Tipo de agente: agente a base de objetivos.

Semáforo

Traffic_Light(Agent)

Objetivo: alternar su estado entre true (verde) y false (rojo) cada 5 segundos.

Capacidad efectora:

- Cambiar su estado (color) dependiendo de si el módulo del número de steps de la simulación de mesa con el número asignado de cambio es igual a cero.

Camino

Road(Agent)

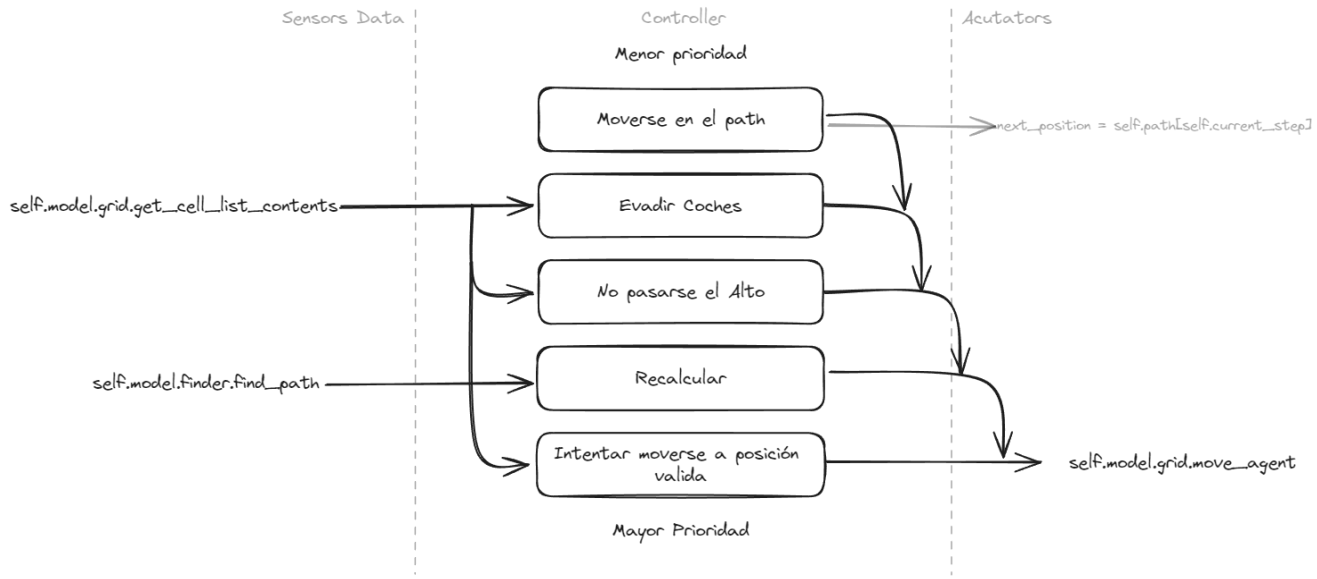
Objetivo: determinar la dirección de movimiento en sus coordenadas.

Obstáculos y destinos

Los agentes de los obstáculos y destinos no tienen un comportamiento programado dentro.

Objetivo: delimitar los espacios de movimiento en el grid.

Arquitecturas de subsunción



Características del ambiente

Inaccesible: se podría llegar a pensar que el ambiente es de tipo accesible ya que los agentes tipo coche conocen todos los caminos que pueden tomar, pero al mismo tiempo no cuentan con una información completa del ambiente ya que no conocen las posiciones de los otros coches.

No determinista: similar a la característica anterior, se podría pensar que el ambiente es determinista, ya que las acciones principales de moverse, esperar o calcular el camino más corto lo son. Sin embargo para el comportamiento al momento de elegir si va a recalcular o no la ruta depende de una variable random la cual no podemos predecir si va a ser activada o no.

Episódico: el agente decide qué acción realizar basándose únicamente en el episodio actual. No conoce las acciones de los demás agentes ni el estado actual ni a futuro.

Dinámico: el ambiente cambia constantemente ya que se generan nuevos agentes de coche cada cierto tiempo. De igual manera los agentes que cumplen con su objetivo son eliminados.

Discreto: el ambiente de la simulación es discreto ya que existe un número finito y fijo de acciones posibles a realizar por los diferentes agentes. Estas acciones son las descritas en la capacidad efectora de cada agente.

Simulación 3D en Unity

Para la simulación de Unity se utilizaron diferentes matrices 4x4 para las transformaciones geométricas como rotación, traslación y escala. Las matrices de rotación se usaron para dos acciones diferentes, la primera es la rotación de las ruedas de los modelos de los vehículos y la segunda la rotación del vehículo con sus ruedas hacia la dirección de su movimiento. La matriz de traslación se utiliza para dar posición a los agentes y por último la de escala para cambiar el tamaño de los mismos a una de acorde a todos los elementos.

El movimiento de los agentes de los coches se dio con la fórmula de Lerp o interpolación lineal para poder traducir las coordenadas que se reciben de un punto inicial a un punto final como el movimiento de los vehículos.

Todas estas matrices de transformación funcionan gracias a un controlador del agente en donde se instancian los prefabs de los vehículos y semáforos (los que si tienen algún tipo de comportamiento) a partir de las coordenadas y estados recibidos url del servidor y de los endpoints generados para poder mandar estos datos.

Conexión Mesa con Python

Para la posibilidad de usar las posiciones de Mesa Python con nuestro modelo de Unity. Tuvimos que instanciar un servidor de Flask para poder utilizar “Métodos de HTTP”, estas llamadas al servidor logran posible la conexión de ambos ya que como cada uno habla en

idiomas diferentes, la comunicación no es tan fácil como pensamos. En el servidor de Flask para iniciar el modelo de Mesa, solicitamos un método HTTP POST para solicitar los datos enviados por Unity. Entonces Unity manda el módulo (que es una variable para saber en cuantos steps del modelo vas a volver a aparecer los coches):

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

    if request.method == 'POST':
        module = int(request.form.get('module'))
        currentStep = 0
        print(request.form)
        print(module)
        cityModel = CityModel(module)

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

Luego tenemos otro método de HTTP llamado Get para que Unity reciba las informaciones que solicite, en este ejemplo está llamando a getTrafficLights para recibir el valor estado de los Traffic Lights Agents y poder cambiar la luz de color en Unity.

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

    if request.method == 'GET':
        trafficLightStates = []
        for cell_content, (x, z) in cityModel.grid.coord_iter():
            # Asumiendo que cell_content es una lista de agentes
            for agent in cell_content:
                if isinstance(agent, Traffic_Light):
                    state_info = {"id": str(agent.unique_id), "x": x, "y": 1, "z": z, "state": agent.state}
                    trafficLightStates.append(state_info)

        return jsonify({'trafficLights': trafficLightStates})
```

Conclusiones

A medida que avanzamos con el desarrollo del proyecto nos dimos cuenta de varias mejoras que se les podían aplicar a los diferentes agentes para aumentar su nivel de inteligencia. Por ejemplo, al agente de los coches se le podría agregar la capacidad de habilidad social dando y recibiendo señales de otros vehículos como lo podrían ser el claxon o las luces direccionales para indicar hacia qué dirección va a si lo está dejando pasar o no. De igual manera esta implementación podría acercar al ambiente a ser un poco más del lado de no episódico ya que el agente conocería las acciones futuras de los agentes cercanos. Por el lado del agente del semáforo, se puede mejorar su nivel de inteligencia igualmente aplicando una capacidad de habilidad y un cierto nivel de percepción. Esto para poder comunicarse con semáforos de la

zona y detectar si vienen agentes de vehículos para determinar y comunicar si tienen y pueden mantenerse en el mismo estado o si pueden o tienen que cambiar.

Un problema de optimización con el acercamiento que se tomó hacia la solución del reto es que se tienen que hacer los diferentes niveles de los grids a mano. Lo cual genera pérdidas de tiempos en algo que se podría hacer de forma automática con un poco más de tiempo de desarrollo.

Video de la simulación

https://youtu.be/Ivm1KJnWT-M