Tecnológico de Monterrey

Reto

Modelación de sistemas multiagentes con gráficas computacionales (Gpo 302)

Alumno

S Fernanda Colomo F - A01781983

Ian Luis Vázquez Morán - A01027225

Profesor

Octavio Navarro Hinojosa

Gilberto Echeverría Furió

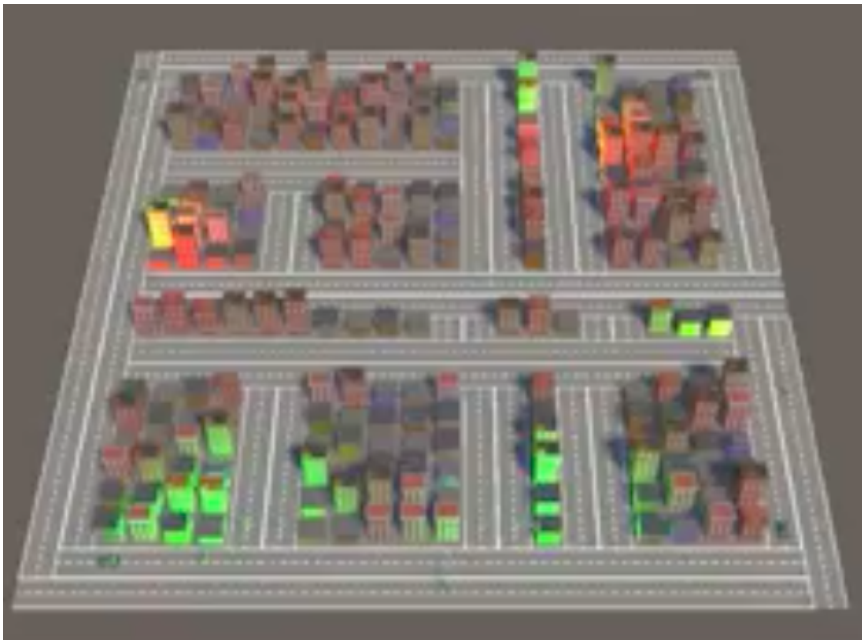
29 Nov 2023

Problema a resolver

Hoy en día el incremento de automóviles en la ciudad ha causado problemas al momento de la movilidad urbana afectando a México y a su población en diferentes aspectos como el ambiente, la parte económica, y calidad de vida. Debido a la mala movilidad en las ciudades también la cantidad de accidentes y enfermedades causadas por el daño ambiental han incrementado significativamente. Es por eso que se busca resolver este problema hallando la mejor manera posible de distribuir el flujo de vehículos y analizar sus comportamientos para así conseguir la manera óptima de solucionar este problema de la movilidad urbana en México.

Propuesta de solución

Nuestra propuesta de solución se basa en una simulación del tráfico vehicular dentro de un mapa dado en el cual los carros deben buscar la mejor manera de llegar a su destino respetando a los demás conductores y las reglas de tránsito así como el sentido de las calles y el estado de los semáforos. Para lo anterior usaremos las herramientas de Unity para la parte de *frontend* (la visualización de la simulación en 3D) junto con las librerías de python, Mesa y Flask para simular el comportamiento adecuado de los agentes involucrados en la simulación(Carro, Traffic_Light, Road, Obstacle, Destination) . Se hará un servidor en Flask para poder comunicar la parte de *backend* de Mesa con la parte de *frontend*, Unity. Mientras que Mesa tiene el comportamiento de los agentes y la construcción del modelo a simular. Cabe aclarar que para la construcción del mapa se usará un archivo de texto con la distribución de los agentes, menos los carros que serán instanciados al momento de iniciar con la simulación. Para saber que agente se debe colocar se usa un diccionario en un json que contiene a qué agente corresponde cada símbolo.



(Imagen de la simulación y construcción del mapa)

Agentes

Carro

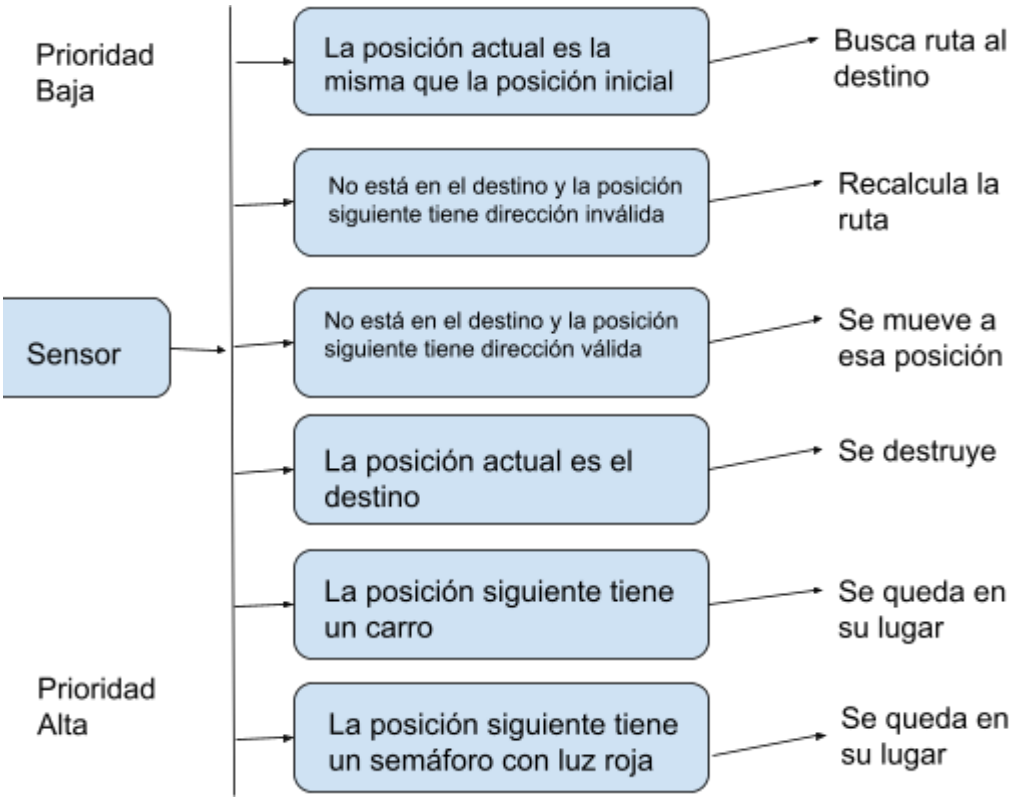
El agente carro es capaz de:

- Moverse a su destino hallando la mejor ruta (Objetivo del agente)
- Evadir obstáculos (con ayuda de Astar al momento de generar el camino posible)
- Detectar si un semáforo está en rojo y detenerse
- Detectar si hay un carro enfrente y detenerse
- Saber si se puede mover en cierta dirección
- Cambiar la ruta
- Detectar si su destino está cerca
- Destruirse si el destino es la casilla actual

Proactividad, reactividad y habilidad social

El carro es capaz de tomar la mejor decisión en base a su meta que es llegar a su destino, para lograr lo anterior el carro es capaz de analizar su entorno y saber cuál es la decisión adecuada o que cambios debe hacer en base a lo detectado en el ambiente. Los demás agentes no actúan para cumplir sus metas ni están al pendiente de algún cambio dentro de ambiente y ninguno (incluyendo al carro) se comunican entre sí.
El carro funciona de la siguiente manera, primero busca la mejor manera de llegar a su destino evitando los obstáculos (edificios), en cada step el carro analiza si se puede mover en esa dirección, de lo contrario cambia la ruta. Al mismo tiempo el carro debe estar al pendiente del estado de los semáforos, si hay algún carro enfrente o si la futura casilla de su siguiente casilla es su destino. En los primeros casos el carro es capaz de quedarse en su lugar, mientras que el último el carro le da prioridad a su futuro destino y si es necesario hace un cambio de carril. Finalmente si se llega al destino el carro se elimina de la simulación.

Arquitectura de subsunción



Posibles medidas de desempeño

- La cantidad de steps que le toma un carro llegar a su destino
- La cantidad de veces que se tuvo que recalcular una ruta

Semáforo

El agente semáforo es capaz de:

- Cambiar su estado cada tiempo determinado

El semáforo es capaz de cambiar su estado a *true* o *false* cada tiempo dado, en esta simulación los semáforos tienen un tiempo de 15 o de 7 segundos para cambiar sus estados. Eso implica que cada múltiplo de 15 o de 7 deberemos ver cambios en el estado de los semáforos.

Obstáculo

El agente obstáculo solo se coloca al inicio de la simulación pero no cuenta con otra acción.

Destino

El agente obstáculo solo se coloca al inicio de la simulación pero no cuenta con otra acción, pero le servirá al agente carro para saber a qué coordenadas ir.

Camino

El agente obstáculo solo se coloca al inicio de la simulación pero no cuenta con otra acción, pero cuenta con direcciones las cuales serán analizadas por el carro en el momento de moverse.

Características del ambiente

Mayormente inaccesible

Como sabemos un ambiente puede ser considerado accesible o inaccesible en base a la cantidad de información que el agente puede obtener del ambiente. En este caso el agente (Carro) conoce a dónde quiere ir y cómo puede llegar allá pero desconoce el estado de los semáforos o si se encontrará con un carro más adelante. Dado que solo conoce parte del ambiente, pero no todo lo decidimos clasificar como mayormente inaccesible. Cabe aclarar que los otros agentes (semáforo, destino, obstáculo y camino) no conocen nada del ambiente más que su posición en el mismo.

Mayormente determinista

Podemos considerar que nuestro ambiente no es completamente determinista dado que el comportamiento del agente carro se caracteriza por tener acciones que pueden desencadenar una acción distinta. Por ejemplo la acción de moverse hacia su destino puede causar que el carro tenga que o replantear la ruta, detenerse si detecta otro carro o si hay una luz roja por lo que cuando el carro intenta cambiar de posición no siempre causa el mismo resultado. Por otro lado, los demás agentes ayudan a que el ambiente sea considerado determinista pues el semáforo siempre cambiará su estado cada tiempo dado y los demás agentes solo son colocados al inicio de la simulación por lo que no cuentan con otras acciones.

No episódico

El ambiente no es episódico dado que el agente (carro) no solo razona qué acción debe tomar en base al episodio actual sino a episodios futuros. Esto se puede ver implementado cuando el carro analiza si la posición que sigue a su posición futura es su destino, entonces prioriza ese camino y hace un cambio de carril si es necesario. Debido a ese comportamiento se puede caracterizar el ambiente como no episódico.

Dinámico

Se considera dinámico ya que hay cambios por parte de otros agentes que están fuera del alcance de un agente (por ejemplo el carro no controla el comportamiento de los semáforos) causando diferentes acciones y cambios por parte de los agentes dentro de su entorno.

Discreto:

Se considera un ambiente discreto dado que si hay un número finito de acciones como las acciones del agente carro como: moverse, evitar carros, detenerse y destruirse. Al igual que en el agente semáforo tenemos acciones finitas que sería el cambio de su estado. A su vez, la simulación se detendrá si se llega al número máximo de carros por lo que también cuenta con un número finito de acciones que se pueden realizar.

Explicación breve del funcionamiento de algunos códigos

Move

El Código move consiste en 5 funciones: Start(), Update(), Positions(), returnPos(), DoTransform(), las cuales serán explicadas a continuación:

Start():
Esta función es la encargada de que al crear el agente car en AgentController éste cree sus llantas correspondientes y finalmente se encarga de obtener los vértices de la mesh del coche y sus llantas, las cuales serán guardadas en variables que se usarán en DoTransform()

Update():
Esta función tiene el cometido de llamar a DoTransform continuamente

Positions():
Es una función sumamente importante puesto que aunque no se llame en en el código de move, esta si es llamada en AgentController y esta se encarga de obtener las nuevas coordenadas proporcionadas por el AgentController a las cuales dirigirse y poder aplicar la transformación de matrices correspondientes

returnPos();
Es la encargada de calcular el desplazamiento del automóvil en un tiempo dado, esta función es llamada en DoTransform() para aplicar el desplazamiento

DoTransform():
Calcula las matrices de transformación, escala y rotación del automóvil y las llantas asociadas, después combina estas matrices y aplica la transformación a las coordenadas de las mallas y finalmente actualiza las mallas de las ruedas con las nuevas coordenadas transformadas.

AgentController

Es el código encargado de transmitir los datos generados en mesa, representar a los agentes en el proyecto de unity y mantener continuamente actualizado el modelo en unity lo que incluye controlar la iluminación de los semáforos, crear carros cuando un agente tipo car se cree en mesa, proporcionar las nuevas posiciones de los agentes al código move y finalmente destruir al automóvil generado junto a sus ruedas cuando el agente que representaba en mesa deje de existir

Server

Este código define e inicia un servidor de Flask que actúa como una interfaz para interactuar con el modelo de la simulación, además el código permite obtener y publicar datos de la simulación a través de endpoints como lo son el número de coches de la simulación que han llegado al destino cada 100 pasos usado para la presentación de este reto.

Agent

En este código se definen los siguientes agentes junto a su comportamiento:

- Car: Representa un vehículo que se mueve aleatoriamente en la cuadrícula.
  Este agente contiene un método move el cual implementa la lógica del movimiento del automóvil, utilizando el algoritmo A* para encontrar el camino hacia un agente destino elegido aleatoriamente.
  El automóvil tiene una posición actual (pos), una posible posición siguiente (next_position), y realiza movimientos en la cuadrícula según su entorno.
  Se programó el Astar usando la librería de Network y ahí se calcula la mejor ruta evitando edificios. Ya después el agente valida si la dirección de su next_position es válida a la posición actual en la que se encuentra el agente, si lo es se mueve, de lo contrario se llama a otro método con un nuevo grafo, sin la posición (nodo) inválida para volver a recalcular la ruta usando la misma lógica de astar.

- Traffic_Light: Representa un semáforo.
  Este agente cambia de estado (verde a rojo o viceversa) cada cierto número de pasos definidos por timeToChange.

- Destination: Representa el destino al que debe dirigirse un automóvil.
  Agente estático creado con el fin de asignarle un destino al agente car

- Obstacle: Representa un obstáculo en la cuadrícula.
  Es un agente estático el cual su único propósito de este agente es evitar el desplazamiento del agente car en ciertas locaciones

- Road: Representa una carretera donde los autos pueden moverse.
  Verifica si hay un automóvil presente en la misma celda y establece su estado occupied en consecuencia.

Posibles mejoras

Para una implementación futura se ha considerado el poner peatones o incluso agregar tolerancia a los carros creando así diferentes comportamientos en los cuales pueda cambiarse de carril si hay un carro enfrente o incluso aumentar su velocidad.

Conclusiones

Después de este reto pudimos aprender la posible aplicación de matrices de movimiento para un objeto así como los posibles cambios de posición posibles gracias a estas matrices. Al mismo tiempo pudimos conocer un poco más sobre el comportamiento de los agentes en un entorno con diferentes tipos de grids, las cuales afectan el comportamiento del agente y el ambiente por la capacidad de tener uno o más agentes en la celdas. Finalmente, también aprendimos cómo se puede implementar todo lo aprendido en semestres anteriores a este y a otros retos que ya se acercan mucho más a la vida cotidiana, como por ejemplo el uso de algoritmos como astar (usado en la simulación) en diferentes apps de tránsito.

Repositorio

Link al repositorio (el video de funcionamiento estará dentro del Readme en el repositorio.
https://github.com/Fer5929/Multiagentes_Unity