Tecnológico de Monterrey

Modelación de sistemas multiagentes con gráficas computacionales
TC2008B.300

“Documento Descriptivo”

Prof. Jorge Mario Cruz Duarte
Prof. Alan Márquez

Equipo # 2

Nicolás Herrera Hernández A01114972

18/06/2022

Repositorio: https://github.com/NicolasHerreraHdez/Equipo2_TC2008B

Documento descriptivo

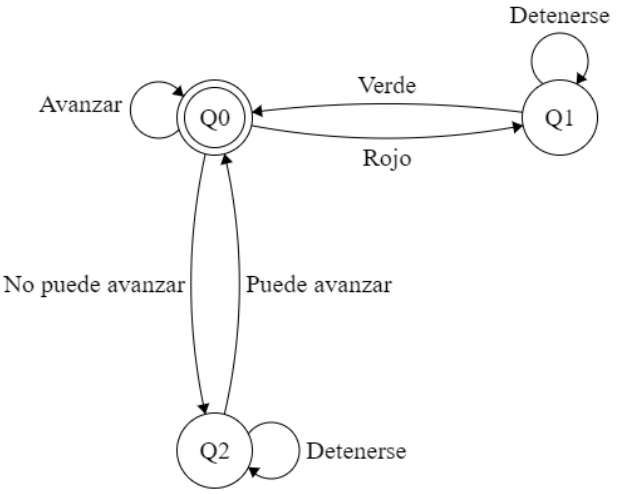
Descripción del reto:

El reto consiste en proponer una solución al problema de movilidad urbana en México, mediante un enfoque que reduzca la congestión vehicular al simular de manera gráfica el tráfico, representando la salida de un sistema multi agentes. Se seleccionó representar de manera visual un cruce de cuatro vías, con múltiples agentes interactuando, esto nos ayudará a reducir posibles accidentes, los cuales son muy comunes, en este tipo de intersecciones. Al mostrar el comportamiento óptimo de los automóviles podremos realizar una demostración visual que educará a los conductores.

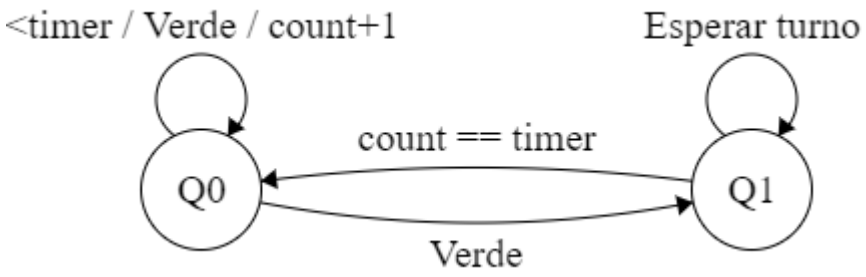
Agentes:

Al considerar que tenemos una problemática de vialidad, es de vital importancia tener en cuenta a los automóviles y a los semáforos. Esto, con el fin de determinar el comportamientos a partir de los coches, Los coches tendrán interacción unos con otros, así como con los semáforos.

- Automóviles
    - Información a detectar:
        - Agente semáforo pertinente
        - Estado del agente semáforo pertinente
        - Posición de automóviles cercanos.
    - Acciones a realizar:
        - Avanzar en caso de semáforo verde
        - Detenerse y mantener posición en caso de semáforo rojo
    - Rol:
        - Avanzar o Detenerse de acuerdo a su situación.
    - Interacciones:
        - El automóvil interactúa con el automóvil frente a él, en caso de que haya uno.
        - El automóvil interactúa con el semáforo diagonal a él.

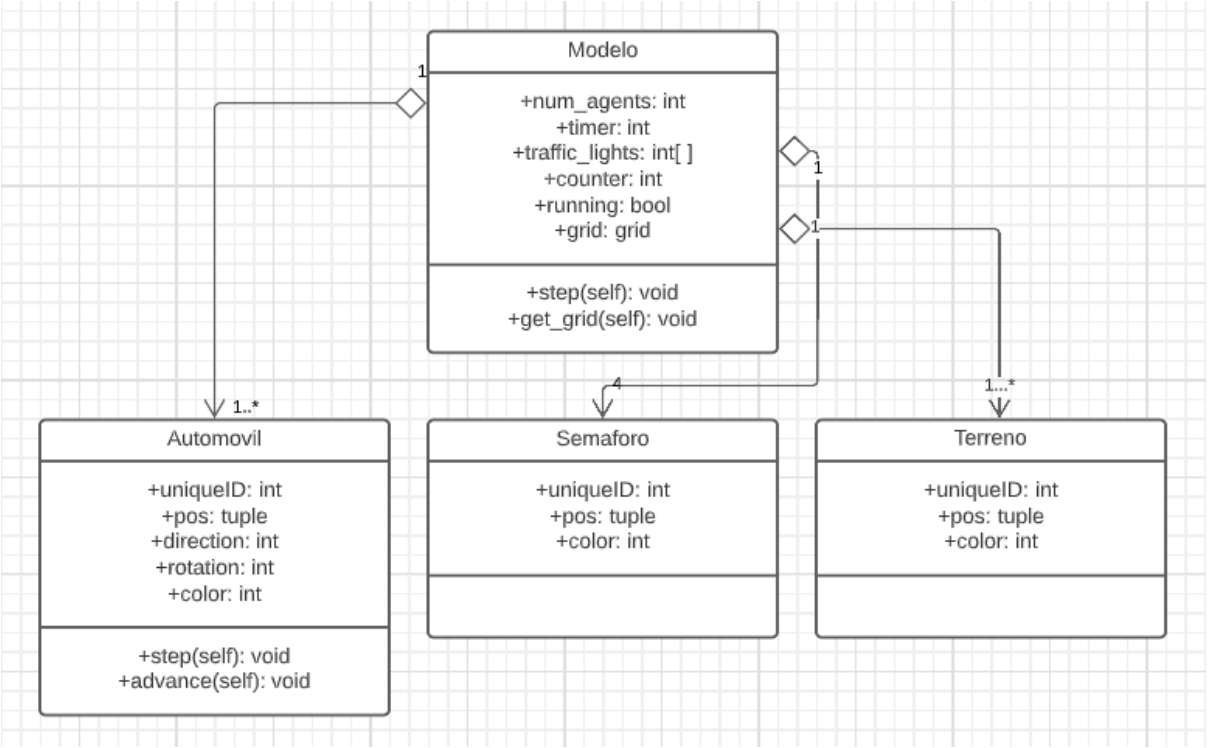


- Semáforos
  - Información a detectar:
    - Estado actual de sí mismo.
  - Acciones a realizar:
    - El semáforo se pone en rojo para que los carros se detengan
    - El semáforo se pone en verde para que los carros puedan continuar avanzando sin problema.
  - Rol:
    - Informar al automóvil su estado actual.
    - Cambiar de color despues de un tiempo determinado.
  - Interacciones:
    - El semáforo interactúa con el automóvil diagonal a él.



- Terreno:
  - Rol:
    - Su valor nos servirá para graficar y simular el modelo en Python.

Diagrama de clases para simulación Python.



Estrategia cooperativa para la solución del problema

Se realizaron los siguientes algoritmos para facilitar la implementación de la solución, en ellos se observa el comportamiento esperado de la simulación y como llegar a él.

Step de agente automóvil.

1. El automóvil siempre estará en movimiento.
2. El automóvil se detendrá cuando:
    a. Detecte que el semáforo correspondiente está en rojo.
    b. No se puede mover porque un automóvil está frente a él.
3. El automóvil se seguirá moviendo cuando:
    a. Su semáforo está en verde.
    b. Su semáforo está en rojo pero se encuentra en el cruce de la intersección.

Estado de semáforos.

1. Se selecciona un semáforo para que sea el primero en ser verde.
2. Se alternan semáforos de acuerdo al tiempo y orden especificado.

Recopilación de datos.

Se usan las funciones internas de masa con DataCollection para recopilar la información de nuestros agentes y la grilla dicha información es procesada en un archivo JSON a ser utilizado en la implementación de unity.

Implementación en Unity.

Elementos de interfaz en el proyecto en Unity.

Al estar modelando una intersección entre cuatro vías se esperan ver los siguientes elementos en la interfaz del proyecto en Unity:

- Automóviles.
- Semáforos
- Calle y señalizaciones.
- Props, decoraciones y edificios.

Se utilizaron los siguientes assets de unity para ver nuestros elementos en escena:

- CITY package: Edificios, props, calles, señalizaciones y entre otros elementos.
- Fatality FPS Gaming Font: Fuente para textos informativos y botones.
- Cartoon Low Poly City Pack Lite: Automóviles.

Componentes principales de Unity.

Animation Manager: es un objeto inicialmente vacío que cuenta con el script AnimationManager como componente. En este componente se puede especificar la duración de cada paso en la animación y el prefab que corresponde al agente que va a hacer la animación. El script AnimationManager se encarga de sincronizar las animaciones de nuestros agente y hace el movimiento con interpolaciones dicho movimiento es realizado en la clase AgentElement con la función AnimateStep y la función de Vector3 Lerp que hace la interpolación lineal de nuestro objeto y hace que se mueva de una posición a otra, dichas posiciones fueron recabadas del archivo JSON en el servidor local de python.

Custom Request: en este objeto se encuentra el script CustomRequest que nos sirve para hacer la conexión con nuestro servidor local, dentro de este servidor se hace un request a la dirección URI donde se encuentra nuestro archivo JSON. En el componente se puede especificar la dirección URI a la cual se quiere hacer la petición además se le indica el objeto de AnimationManager esto para poder generar nuestros agentes por medio de prefabs pasando la información necesaria del archivo JSON procesado con JsonUtility. El script de CustomRequest hace la petición al la dirección URL antes proporcionada, en caso de que la petición sea aceptada se obtiene la información de la página por medio de un string dicho es procesado por JsonUtility y la información se pasa una instancia de la clase

PythonSimulation finalmente se pasa esta instancia a nuestro AnimationManager