INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

SEDE: Santa Fe

Simulación de tráfico

Modelación de sistemas multiagentes con gráficas computacionales (Gpo 301)

Alumnos:

Agustín Pumarejo Ontañón	A01028997
Adriana Abella Kuri	A01329591
Fernanda Nava Moya	A01023896

.

Agentes

- **Traffic_Light**
  - Estado (booleano: verde, rojo)
  - Tiempo en que cambian entre estos valores booleanos
- **Destination**
  - Coordenadas en donde se encuentra
- **Obstacule**
  - Edificio dentro de la ciudad
- **Intersection**
  - Es un nodo
- **Road**
  - Dirección de tráfico (arriba, abajo, derecha, izquierda)
- **Car**
  - Estado (booleano: andando, parado)
  - Destino (coordenadas)
  - Intersección actual (ID de intersección)
  - Intersección próxima (ID de intersección)
  - Posición (coordenadas de vehículo)
  - Se mueve diferente en intersecciones y calles
  - Encuentra la ruta más rápida

En los carriles van a estar pasando los vehículos. El carril no se va a mover en ningún momento, este se quedará en el mismo lugar durante toda la simulación. Sin embargo, las intersecciones van a ser representadas por nodos los cuales están conectadas entre sí. Decidimos hacer esto para poder crear un grafo:

```
    adjacent = [
                [0,6,1000,6,1000,1000,1000,1000,1000,1000,1000],
                [1000,0,1000,1000,1000,15,1000,1000,1000,1000,1000],
                [15,1000,0,1000,1000,1000,1000,1000,1000,1000,1000],
                [1000,1000,6,0,1000,1000,1000,6,1000,1000,1000],
                [1000,6,1000,6,0,1000,1000,1000,1000,1000,1000],
                [1000,1000,1000,1000,6,0,1000,1000,1000,6,1000],
                [1000,1000,6,1000,1000,1000,0,6,1000,1000,1000],
                [1000,1000,1000,1000,1000,1000,1000,0,6,1000,6],
                [1000,1000,1000,1000,6,1000,1000,1000,0,6,1000],
                [1000,1000,1000,1000,1000,1000,1000,1000,1000,0,23],
                [1000,1000,1000,1000,1000,1000,15,1000,1000,1000,0]
            ]
```

Para que de esta manera se pueda sacar la ruta más eficiente entre la posición actual del vehículo y su destino utilizando el algoritmo de Floyd Warshall el cual crea una matriz de una dimensión n*n donde ‘n’ es el número de vértices, sus índices representan los vértices y sus valores es el peso de la conexión entre esos vértices. Una vez que la matriz

es creada y los valores del peso entre los vértices son insertados se toma en consideración, entonces se encuentra el vértice intermedio del camino más corto desde el origen al destino. Este vértice se guardará la variable ‘u’ para que en A[i][j] se inserte (A[i][u] + A[u][j]) if (A[i][j] > A[i][u] + A[u][j]) para que si la distancia directa desde la fuente al destino es mayor que la ruta a través del vértice ‘n’, entonces se inserta A[i][n] + A[n][j]. Se calcula la distancia desde el vértice de origen al vértice de destino a través de este vértice ‘n’. La complejidad de este algoritmo es de $O(V^3)$**:**

```
for k in range(len(self.adjacent)):
      for i in range(len(self.adjacent)):
            for j in range(len(self.adjacent)):
                  if(self.adjacent[i][j] > (self.adjacent[i][k] +
self.adjacent[k][j])):
                        self.adjacent[i][j] = self.adjacent[i][k] +
self.adjacent[k][j]
                              self.next[i][j] = self.next[i][k]
```

De igual manera, los vehículos se estarán moviendo encima de los carriles hacia una dirección y sentido, este va a cambiar su estado de “andando” a “parado” dependiendo del estado del semáforo el cual, al igual que el carril, va a permanecer en el mismo lugar desde que se empieza la simulación. Cada vez que se mueve el vehículo este lo hace en el sentido de la calle, el cual lo obtiene del atributo *direction* del agente Road, y esquivando obstáculos, coches y respetando los semáforos.

Estado de intersección:

- Si un coche detecta una intersección en su trayectoria revisa si hay alto.
    - Si no hay alto, se revisa la trayectoria del coche.
        - Si la trayectoria es la misma que su dirección actual, solo las intersecciones que están delante del coche se marcan como ocupadas con nivel de preferencia 1.
        - Si la trayectoria del coche involucra una curva a la derecha, se calculan las intersecciones por las que pasaría el coche y se marcan como ocupadas con nivel de preferencia 2.
        - Si la trayectoria del coche involucra una curva a la izquierda, se calculan las intersecciones por las que pasaría el coche y se marcan como ocupadas con nivel de preferencia 3.
    - Si hay alto, se hace el mismo proceso pero se le suma 3 al nivel de preferencia
- Conforme el coche cruza la intersección, las celdas por las que va pasando se desocupan.

Calcular cruce:

- Un coche sigue la dirección de tránsito de la calle en la que está. (norte, sur, este u oeste)
    - Si está en una intersección y su dirección de tránsito no es la dirección actual, revisa la celda en la orientación a la que quiere dirigirse (norte, sur, este u oeste)
        - Si la dicha celda es una calle, revisa la dirección de tránsito
        - Si la celda es una intersección, revisa la siguiente celda en la misma orientación
            - Repetir este paso hasta encontrar la calle, revisar su dirección de tránsito después
        - Si la dirección de tránsito coincide con la trayectoria del coche, cambiar el rumbo hacia esa dirección
    - Si la trayectoria es la misma a la dirección de tránsito actual, mantener el rumbo.

Manejar:

- Revisar próximo movimiento
    - Si hay un coche ahí, mantenerse quieto.
    - Si hay una intersección, revisar señal y calcular el cruce.
        - Si se tiene preferencia sobre el cruce, se ejecuta.
        - Si no se tiene preferencia, mantenerse quieto hasta que se tenga. Se marca la ruta con espera 1 por cada step que no se tenga preferencia.
        - Si se empata la preferencia, se revisa la espera, si es la misma el coche de la derecha tiene preferencia (N > E > S > O > N) si no, se empieza a contar la espera y se marca.
- Si hay un carril vacío, moverse a este

DIAGRAMAS DE CLASE