Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Querétaro

Reporte Final

Modelación de sistemas multiagentes con gráficas computacionales

(Gpo 402)

Santiago de Querétaro, Querétaro, 03 de Diciembre del 2022

Pedro Oscar Pérez Murueta
Alejandro Fernández
Denisse L. Maldonado Flores

Josemaría Robledo Lara - A01612376
Renato Sebastían Ramirez Calva - A01275401

Descripción del Reto

Los retrasos en el transporte debido a la congestión generada por el transporte público y privado son comunes en muchas áreas urbanas del mundo. Para hacer que los sistemas de transporte sean más eficientes, actualmente se están desarrollando sistemas de transporte inteligente (ITS, Intelligent Transportation Systems). Uno de los objetivos de ITS es detectar áreas congestionadas y redirigir los vehículos lejos de ellas. Sin embargo, la mayoría de los enfoques existentes solo reaccionan una vez que se ha producido el atasco y, muchas veces, no son capaces de indicar en qué punto de la vía se encuentra. Esto se complica aún más en calles de más de un carril, cuando sería posible cambiar de vía para evitar el atasco. El reto consistirá en el desarrollo de una solución computacional que sea capaz de detectar en qué punto de una calle de varías vías se encuentra el punto de atasco y avise con anticipación a los conductores para que éstos puedan cambiar de carril con la expectativa de reducir el problema generado por ese atasco.

El reto consiste en el desarrollo de una solución computacional capaz de detectar el punto de congestión en una vía de varios carriles. Además se nos ha presentado el siguiente escenario: Se presenta una vía de 3 carriles de 1 km de largo, sin acceso lateral. En esta vía sólo transitan vehículos clasificados como de transporte particular. Además la velocidad máxima en todos los carriles es de 60 km/h.

Condiciones:

- Los agentes participantes en la simulación son vehículos de no más de 5 pasajeros
- Los vehículos empiezan en un carril aleatorio a la velocidad máxima permitida y, a menos que exista un problema en su ruta, permanecen en ese carril con aceleración constante.
- Ningún vehículo podrá tener un comportamiento errático.
- En todo momento, un vehículo conoce la posición de los vehículos que lo rodean.Por ningún motivo, un vehículo puede colisionar con otro.
- La duración del escenario será de 5 minutos. En algún momento, después de los 2 minutos, uno de los vehículos del carril central que se encuentre entre los 450 metros y 550 metros empezará a frenar hasta detenerse. Y así permanece hasta el final de la simulación.
- Los vehículos reportan en todo momento su velocidad y posición.

Modelado

Para el modelado del comportamiento de los carros utilizamos Mesa, el cual es una dependencia de python la cual nos permite diseñar sistemas de multiagentes para simulaciones de comportamiento complejo.

Generamos los agentes cada paso que da la simulación con una probabilidad del 50% de aparecer asignándoles una posición aleatoria en un pista de tres carriles y cien metros de largo, por lo que los parámetros para la creación del mundo resultan como $x = 3, y = 100$, entre estos valores de x solicitamos que el programa selecciones una posición aleatoria x_c de

cero a dos para asignarla como un carril en el cual posicionarse al auto, por lo que su posición será x_a, 0.

```
pos = np.random.choice([0, 1, 2])
agent = CarAgent(self.uids, self, pos, 0)
self.grid.place_agent(agent, (pos, 0))
self.schedule.add(agent)
self.uids += 1
```

Figura 1. Generación de agentes de manera aleatoria

Problema

Si para manejar el conteo de autos les asignamos un Id unico que inicia a partir de uno y se va sumando y asignando cada vez que se crea un auto, si dicho conteo llega a una cierta cifra entonces generamos un auto en específico que se encuentre en la posición del carril de enmedio, $x_c = 1$.

```
if self.uids == 30:
    agent = CarAgent(self.uids, self, 1, 0)
    self.grid.place_agent(agent, (1, 0))
    self.schedule.add(agent)
    self.uids += 1
```

Figura 2. Generación del auto que creará el tráfico

Entonces, este auto, al llegar a una cierta distancia empezará a disminuir su velocidad de cinco en cinco hasta llegar a cero, al alcanzar esta cantidad el auto se detendrá por completo, y la simulación tendrá la oportunidad de crear un embotellamiento.

```
if self.unique_id == 30 and self.y >= self.model.quint:

    self.speed -= 10
    self.move = False
    if self.speed <= 0:
        self.model.stop = True
        self.speed = 0
        return
    else:
        if self.check_FPath() == True:
            self.y = self.y + 1
            self.model.grid.move_agent(self, (self.x, self.y))
```

Figura 2.1. Pérdida de velocidad después de cierto recorrido

Solución

El freno por completo del auto seleccionado activa una variable booleana global declarada en el modelo de la simulación para que los demás autos al acceder a ella puedan saber desde una cierta distancia que deben cambiarse de carril para poder evitar un embotellamiento.

```
if  (self.y == (self.model.quint - 5)) and (self.x == 1) and (self.model.stop == True):
    rand = np.random.choice([1,2])
    #Muevete a la Derecha
    if rand == 1 and self.check_Rneighbors() == True:
        self.y = self.y + 1
        self.x = self.x + 1
        self.model.grid.move_agent(self, (self.x, self.y))
    #Si no muevete al izquierda
    elif self.check_Lneighbors() == True:
        self.y = self.y + 1
        self.x = self.x - 1
        self.model.grid.move_agent(self, (self.x, self.y))
    elif self.check_Lneighbors() == False and self.check_Rneighbors() == False:
        self.model.trafic.append(self)
        print("No pude cambiarme")
```

Figura 3. Cambio de carril desde una posición favorable

Movilidad

Para que los agentes pudieran moverse se diseñó un sistema de alerta en el cual pudiera estar al tanto de si la silla hacia enfrente de ellos se encontraba vacía, entonces tendría paso para moverse una posición hacia adelante.

```
def check_FPath(self):
    if self.get_out() == True and not(self in self.model.out):
        self.model.out.append(self)
        return False
    else:
        if self.get_out() == False and self.model.grid.is_cell_empty((self.x, (self.y + 1))) == True:
            return True
        else:
            self.speed == 0
            return False
```

Figura 4. Validación de paso hacia enfrente.

Además de requerir un cambio los agentes tenían la opción de verificar por medio de este sistema si es que tuvieran paso hacia la izquierda o la derecha.

```
def check_Rneighbors(self):
    if self.get_out() == True and not(self in self.model.out):
        self.model.out.append(self)
        return False
    else:
        if ((self.x + 1) <= 2) and (self.y < self.model.height):
            if  self.get_out() == False and self.model.grid.is_cell_empty(((self.x + 1), (self.y + 1))) == True:
                return True
            else:
                return False
        return False
```

Figura 4.1. Verificación de paso hacia la derecha

```python
def check_Lneighbors(self):
    if self.get_out() == True and not(self in self.model.out):
        self.model.out.append(self)
        return False
    else:
        if((self.x - 1) >= 0) and (self.y < self.model.height):
            if  self.get_out() == False and self.model.grid.is_cell_empty(((self.x - 1), (self.y + 1))) == True:
                return True
            else:
                return False
        return False
```

Figura 4.2. Verificación de paso hacia la izquierda

Resultados

Con un arreglo declarado en el modelo del mundo, le agregamos el agente que no pueda moverse ya sea hacia enfrente, a la derecha o izquierda, esto para indicar cuantas veces los agentes no ha podido moverse en total.
En una simulación de 300 iteraciones se aplicó el programa que no incluía esa variable global con las que los autos sabían cuándo moverse, el resultado fue que los autos se detuvieron por completo y no pudieron o virar hacia la derecha o izquierda o seguir moviéndose hacia enfrente un total de 57 veces, a diferencia de la implementación de dicha variable que solo no pudieron tener paso 10 veces en total.

- Primera Prueba:
  - Sin Solución:
    57 detenimientos
  - Con Solución:
    10 detenimientos

- Segunda Prueba:
  - Sin Solución:
    70 detenimientos
  - Con Solución:
    16 detenimientos
- Tercera Prueba:
  - Sin Solución:
    63 detenimientos
  - Con Solución:
    18 detenimientos

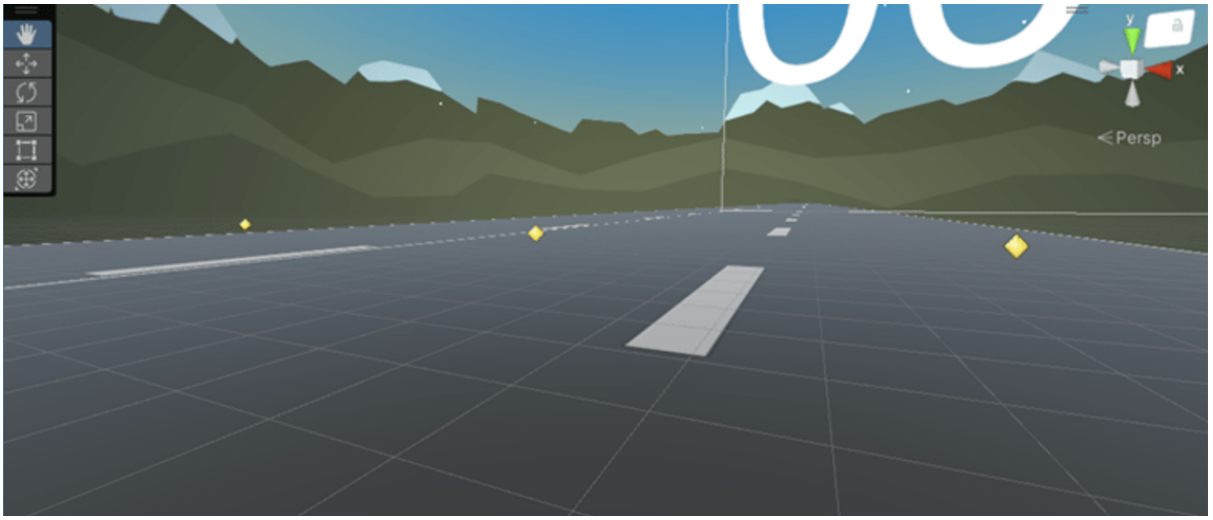
Esta diferencia va de $300\% - 600\%$ de cuando la implementación de la solución es usada.

Link de repositorio para verificar resultados:
https://github.com/Renato549/Multiagentes-tc2008b.git

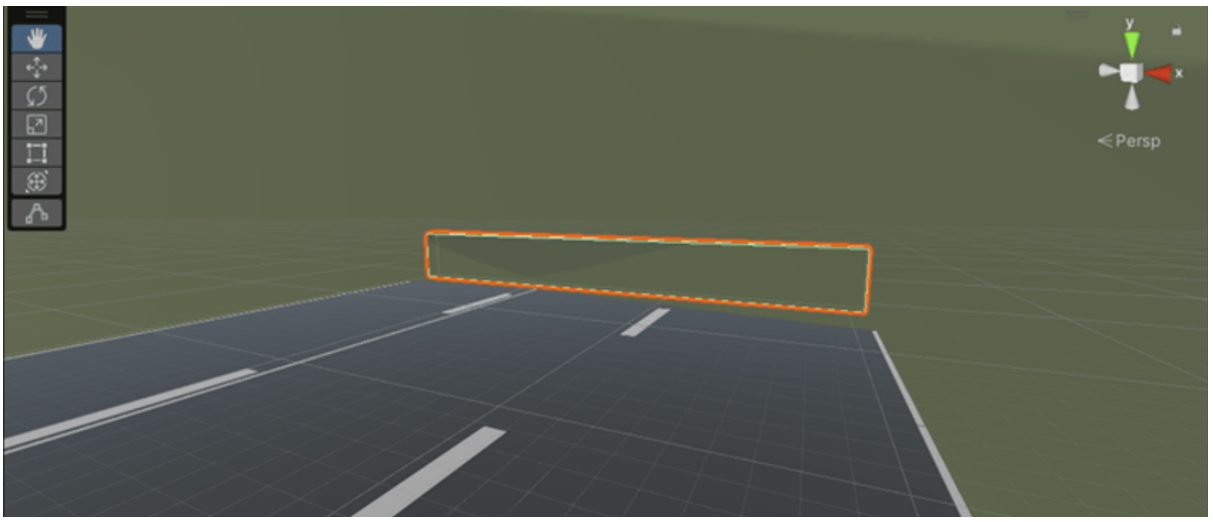
Unity

Dentro del apartado de Unity no se pudo realizar de manera correcta la conexión con la API, sólo se pudieron obtener nuestros tres datos principales del json (Pista, Velocidad e Id), pero fue imposible adaptarlos al entorno de Unity. Debido a esto se utilizó la alternativa incompleta de realizar una simulación generada directamente en Unity.

Se crearon el prefabs para los vehículos a la vez que tres gameobjects vacíos que fungían como generadores, cada uno ubicado en un carril, para que de este modo sólo se tuviese que obtener el carril propio del vehículo y asignarse dentro de la carretera real. Estos generaban vehículos de manera aleatoria en cada uno de estos tres carriles.



Al mismo tiempo al final del camino o de la carretera, se colocó un muro para que los vehículos al momento de colisionar con éste desaparecieran sin generar mayor problema por la cantidad inmensa de vehículos.



Por último también se hizo la implementación de un reloj o de un cronómetro para que después de los dos minutos de simulación generase la problemática, es decir, detuviese a uno de los carros del carril central para generar tráfico, a su vez el reloj sería el encargado de detener la simulación una vez pasados los cinco minutos. Para apreciar de mejor manera la simulación fue implementada la función de cambio de cámara, estaba ubicada a la mitad de la carretera para que fuese posible visualizar con mayor cercanía al comportamiento de los vehículos al notar que se había detenido uno.